NATURAL LANGUAGE PROCESSING FOR RESUME RANKING USING LABELLING

By

OLIVER KRIEGER
URN:6664919

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2024

School of Computer Science and Electronic Engineering
University of Surrey
Guildford GU2 7XH

Supervised by: Stella Kazamia

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Oliver Krieger
May 2024

Abstract

The purpose of this research paper is to build an effective resume parsing solution with natural language processing, capable of retrieving valid job applications to a given query. The principal topic to achieving this will be using labelled data to improve upon retrieval of key information over using conventional methods of text parsing or passive similarity calculations. This data is used to provide scoring to find the best matches for a job description or a user query.

Alongside the research, a simplistic app will be created to both visualise and improve upon datasets, as well as help systematically process data. This will serve both as a testing hub, as well as introduce how the above approach could be applied to a real life application that could be used by someone with superficial knowledge of the data and the artificial intelligence.

Acknowledgements

I am extremely grateful to my supervisor Stella for her continued support, encouragement and positive attitude throughout the entire process of this project! I can say with certainty that I was very lucky to have her as my supervisor!

I will not be able to thank every single person over the course of this very long period that I have been in university, but I do want to express that I am extremely grateful for the amazing collective that I have been part of not only for this final year, but my entire 4 years as a student at the Surrey school of Computer Science. To all these amazing people that I have interacted with over the years, I would like to extend my deepest gratitude!

Glossary

AI: Artificial Intelligence, field of computer science that deals with computers being able to perform "intelligent" tasks usually associated with humans.

BERT: Bidirectional Encoder Representations from Transformers, a deep learning model, where every output is connected to every input, with the weights dynamically connected

GUI: Graphical User Interface, graphical interface for the user to interact with the software.

ML: Machine Learning, field of study in artificial intelligence, concerned with development and study of probabilistic algorithms.

NER: Name Entity Recognition, extraction and classification of named entities/labels within texts.

NLP: Natural Language Processing, field of artificial intelligence that deals with making read and written human language to be understandable by machines.

NLTK: Natural Language Toolkit, tool to preprocess texts for further analysis.

Node.js: Open source JavaScript runtime environment, often used for server side programming.

POS Tagging: Part-of-Speech Tagging, technique to assign labels to words in their specific part of speech

spaCy: Open Source Library in Python that deals with NLP.

Vue.js: JavaScript framework built to extend HTML, CSS and JavaScript into one templating language.

Contents

1 Introduction

1.1 Motivation

1.1.1 AI in Resume Ranking

The role of AI (Artificial Intelligence) has always been important in the field of recruiting. In the early 2000s, due to the ever increasing influx of new hires, applicant tracking systems became popular, streamlining the process of managing the candidates resumes electronically and laying the groundwork for the AI-driven recruitment technology [12]. Because of this, around 2010, machine learning algorithms were ready to start offering advanced features, such as resume parsing, keyword matching and automated candidate screening. Using predictive analytics, it allowed companies to go through vast sums of candidates in record time.

However, to this day these algorithms are being improved, due to the fact that artificial intelligence often struggles to "perfectly" assess unstructured data that often comes with resumes. With the number of applicants rising due to population growth [76], AI in recruitment remains prevalent today to keep up with increasing demand of hiring the best possible candidate.

Not only will AI help with efficiency of reducing repetitive tasks in the candidate screening, but it can also save companies money and time during the process and increase their chance of finding the best possible candidate. This, all the while improving the ability for a company to increase their recruitment input to any size, thus improving scalability [6].

1.1.2 Risks and Benefits

As with most things related to artificial intelligence, there are benefits and drawbacks to it. AI tools can automate tasks, save recruiters time and effort, analyse large volumes of data that would be infeasible for a human to go through, improve hiring quality, mitigate biases by focusing solely on the qualifications of the candidate and enhance the applicants experience while applying.

However, while there is an effective reasoning why artificial intelligence should be used, there are also dangers to be kept in mind. It is true that an AI can focus on qualifications of the individual, but should it include historical data, it can perpetuate and amplify biases, as well as take out the human factor that might decide that culturally (in the office) a less knowledgeable candidate is a better fit to the collective. To that end, the aim of this project makes sure not to tackle data that is historically important and focus solely on the candidate qualifications themselves.

Taking this into account, the architecture explored in this paper will be reliant on having accurate and well defined data for labelling and ground truths that

is needed to find named entities. This means a significant amount of time has to already have been invested by humans to gain pre-labelled data from open source datasets. This will also require expertise in data science or machine learning to ensure that the labels are accurate and concise. A combination of this is difficult to find. This research will also attempt to find methods to improve generation of this data through the means of user interfaces and text generation.

1.1.3 Approach

With the above in mind, to reduce the workload on the recruiters as well as create a sustainable model for use in the future, an AI solution is required. While there are many options out there to rank resumes by classification or topics, these options often lack the fine tuning required to consistently improve the ranking model. In the case of topic classification [64], it is difficult to capture the type of every single job on the planet as well as make it relevant for the specific company. In terms of topic modelling it often relies on word frequency and thus can be misleading on candidates specialisation.

During the start of 2023, a paper was released from the Datta Meghe College of Engineering in India called "Resume Parser using hybrid approach to enhance the efficiency of Automated Recruitment Process" [2], outlining a "hybrid" approach, using both spaCy's transformer and NLP methodology. This paper will try to follow a similar hybrid approach, using spaCy NER (Named Entity Recognition) to label relevant data onto the resumes and apply that data later in the ranking process via rule based methodology or a BERT (Bidirectional Encoder Representations from Transformers) model.

1.2 Aims and Objectives

The aim of this report is to tackle solutions to two separate tasks:

1. **GUI** - Create a GUI for visualisation and user interactions for an application that can be used to apply labelling and ranking methods, as well as visualise the data.

2. **AI** - Implement and evaluate two separate AI models, one tasked with recognition of named entities and one to rank resumes to a search query.

As such, to realise these aims, objectives can be set for both tasks. The GUI will be capable of the following:

1. **Data Extraction** - Uploading data in a PDF or JSON file format and extracting data into relevant JSON format.

2. **User Interactions** - Providing a visual GUI for the users to interact with as to reduce complexity of running commands, as well as providing a reliable system to run tasks in.

3. **Label Editing** - The capability to add and remove labels from resumes for training ground truths. The system should also be capable of recognising faulty labels (labels that overlap with other labels or do not start or end with a space).

4. **Find Labels** - Find and label entities within texts that will be used for ranking.

5. **Ranking** - Rank resumes from user input or job description

6. **Model Accuracy** - Provide accuracy metric for finding labels from ground truths to see how well a model is performing.

The AI models will be capable of the following:

1. **Label Training** - Given a train/test/validation split of data, a model capable of training named entities and producing metrics of how well the model has learned.

2. **Entity Recognition** - Given a text in string format, the model is capable of recognising entities and displaying them.

3. **Ranking** - Given a job description or a search query, the model is capable of finding best resulting resumes.

2 Literature Review

2.1 Current Hiring Process

While the average volume of applicants that a job receives can depend on various factors, such as specific requirements for the job, location and company reputation, the average applicant volume for a job can range from anywhere from tens to hundreds of applicants. A survey conducted by Stand Out CV suggests that the average number of applications for a "low skill" job is around 506, whereas for "high skill" jobs the number is on average 55 applications [19].

For well paid or entry-level roles, this volume can be much higher, often reaching even thousands. The more specialised the role, the fewer applications there are. With the rise of online job boards such as LinkedIn, the number of applicants has been steadily increasing.

It is important to note that not all applicants meet the requirements for the jobs they apply for. However, on the flip side of this, a really qualified applicant could be passed over because their application is at the bottom of the resume stack and with too many applications, their application could never be seen. With an artificial intelligence solution the latter would never be a problem and the former applicants would be immediately scored low for not having enough relevant experience to what the job posters are looking for.

As recruiters and hiring managers often use applicant tracking systems and other user interface tools to screen large volumes of applicants, a GUI is required, as well as to make the resume parsing easier.

2.2 Building a GUI

In order to develop an app that would be capable of filling requirements listed in the objectives, not only would it have to be easy to manage, but also have to be able to work on multiple platforms. As the purpose of the GUI would be to visualise and parse data, something that would run natively on the user's device made sense from a computational power standpoint. As such, a cross platform desktop application was the most suitable resolution to the problem.

Implementation from scratch remains an inefficient strategy in terms of time and code robustness and as such, something that could handle multiple platforms had to be chosen. WPF (Windows Presentation Foundation) [72] exists on the windows operating system (and can be run as a web application) to create such executables, but would not easily support other platforms. Scythe Studios blog post from 2022 about best frameworks for cross platform desktop applications lists QT, Flutter, Electron and JavaFx as some of the most viable candidates [1]. As something that matches closely to WPF and can also run JavaScript, Electron.js [11] was chosen to fill that role.

To avoid difficulties with real time updates when interacting with the application, a two-way binding layer would have to be implemented on top of it. Due to wide usage, as well as integration with Electron-Vite-Vue, Vue.js [67] was chosen for this.

2.3 Labelling Strategies

2.3.1 Labelling as a Ranking Strategy

As mentioned in the approach section of this report, a hybrid approach to rank resumes could improve upon the existing methods currently available. Techniques like topic classification, word frequency, similarity algorithms (euclidean, cosine, etc) are already used to rank texts to a search query. It is important to understand the advantages and limitations of each to see how labelling could improve upon this.

As a method for ranking, topic classification can be an effective approach on its own. When ranking, resumes are first classified by a topic and the topic closeness is matched to the query topic (user input or a job description). However, topic classification can be limiting, as it requires large volumes of data to be trained into predefined topics and will expect similar texts in the future.

A more general approach to this would be to rank the entire text to the closeness of the topic, by using embeddings or vectors, using either sentence transformation or classification. That way every word within the corpus of a text could be verified to the topic and tested for closeness. The text similarity will remove the issue of having predefined topics, as texts would be matched to their closeness and if a topic is added later, it can be a visual aid to understand why a text was matched to another text. A well utilised sentence transformation could yield good results, but might suffer from being able to distinguish important information.

The aforementioned issue can be resolved by using a labelling strategy (NER), which can be implemented in a way that allows for discovery of important information from the text. This data can then be inserted into a similarity comparison algorithm or pairwise comparison [51]. The drawback is a possibility to discard too much data for similarity comparison algorithms that use sentence context to perform accuracy calculations.

2.3.2 How NER works

NER [71] is a sub-task of information extraction. It locates named entities in unstructured texts. NER is usually performed on a corpus (a collection of texts) to both tag and train on. The texts are usually preprocessed to remove noise, such as special characters and broken whitespaces. Sometimes punctuations and stopwords are removed, but not always as we will see later, as it might be

counter intuitive for the task. Each word is then broken into a "token" and has a corresponding "ner" tag. These tags might also sometimes be called "IOB" tags.

The IOB (formerly known as "BIO") format is short for inside-outside-beginning format [55]. The purpose of this format is to not only to give each word within a corpus a "tag" but to explain to the learning algorithm what is important and what is not. Take for example the sentence "World Trade Center is in New York, said Alice". The ground truth tagging of this sentence would be as follows:

World	Trade	Center	is	in	New	York	,	said	Alice
B-BU	I-BU	I-BU	O	O	B-LOC	I-LOC	O-PUNCT	O	B-P

In this case the algorithm would be looking for three labels: building (BU), location (LOC) and person (P). However, the punctuations (PUNCT) have also been labelled, even though they are not within the labels being searched for. This is because the algorithm can be taught to find punctuations so it would not confuse it for another label and then given "O" to know that it should discard it. "B-" means that an entity is beginning and "I-" means that the same entity is continuing. If we had two "B-" next to each other, we would know that even though they can have the same entity names, they are two separate names, while having "B-" and "I-" next to each other means that both tokens belong to the same entity. This creates a very powerful method to teach the algorithm how to recognise patterns and distribute weights.

The obvious drawback with this system is that it cannot do nesting. For instance "The Department of Computer Science in Surrey" could be mostly labelled as "Education". However, "Computer Science" itself could also be labelled as "Experience". Thus, if it is surrounded by education, we would want computer science to be recognised as such as well, but otherwise be recognised as experience. However, labelling it two different things in separate places, can cause inconsistencies for the weights and thus end with improper results, especially if the dataset is imbalanced towards one of the two tags (experience or education). Apart from named entity recognition tagging, a NER algorithm can also utilise POS tagging, word embedding and chunking to identify the named entities. The POS tagging allows for the algorithm to recognise specific parts of speech, such as adjectives, verbs and nouns. Word Embeddings can be numeric or vector representations of words, essentially encoding the words to a numerical vector of fixed size, making it plausible for a machine learning model to process. Chuncking (or sometimes also called "windowing") is taking a piece of text, for instance a sentence and separating it into "chunks". These chunks can be labelled with new values or it can be used to generate more data for an algorithm to learn [3]. When the model is trained and applied to a text, it is capable of finding entities within a given text.

Algorithms often have two ways to find labels within texts. One way is to use index matching as shown above in the IOB example to match a token at an index to a tag at the same index. However, when given a text, the entities can also be represented as start position to end position with a label. For instance a sentence could be represented as

"Sarah K Senior Software Engineer London"

$$person(0|6), experience(8|31), location(33|39)$$

From the example above, we can see that:

1. "Sarah K" has been labelled as a "person", indicating that the entity is representing a person's name.
2. "Senior Software Engineer" has been labelled as "experience" indicating that this is a job experience this individual has had.
3. "London" is labelled as a "location", indicating that this person has worked there.

With this not all of the words have to be parsed into tokens and entities can be specifically defined within ranges. However, the drawback of this method is that it can produce broken words if the indices are positioned incorrectly. For instance, if experience would start at position 6, it would also include "K" from "Sarah K" as well as an empty whitespace, which would at best make the algorithm produce terrible results and at worst be untrainable. As such, great care and error detection needs to be implemented when choosing this method.

The benefit of NER is creating structure to unstructured data. It identifies and categorises key elements such as names, locations, dates and other specific terms within the text. By doing this, NER transforms significant quantities of data into organised datasets, ready for further analysis, effectively creating a summary. More sophisticated NER algorithms perform post-processing steps that refine the output of the model, such as merging adjacent tokens or resolving overlapping entities, handling special cases and ambiguities. A system that allows for fine tuning on this is also beneficial.

2.3.3 Choosing a NER Method

As mentioned in the introduction, the aim of this report is to utilise named entity recognition in order to create a system that is capable of extracting the most relevant information from a CV and building a ranking system around that. While an entirely new algorithm architecture could be built using either neural networks or linear regression, there are already many capable models out there that tackle the labelling issue. Such LLMs (Large language models)

include BERT (DistilBERT, RoBERTa, BERT Large, etc), NLTK and spaCy [61].

The main reason why pre-trained models often outperform custom built models is because they are assembled by large corporations with money and resources to invest into spending time improving the model. This translates to better "weights" within the model when starting out and as such, significantly improves results when fine-tuning a model to a new dataset [69].

From the models mentioned in this section, BERT [8] based architecture is considered to be the most accurate. It is, however, also considered the most computationally expensive [61]. While BERT might be expensive to train on, it has managed to outperform many popular models and is significantly better from architectures that are not pre-trained [4]. It should be also noted that BERT can perform many tasks (text classification, sequence classification, fill mask, etc)[9], not just named entity recognition, also known as token classification. Thus, without fine-tuning, BERT might not be able to perform as well as other purpose built pre-trained models.

As BERT is not a model, but an architecture that many transformer based models are based on, the spaCy [13] pipeline has also integrated BERT into its workflow [62] using Hugging Face's [20] interface. It is capable of performing entity recognition for non-overlapping tokens using "Entity Recogniser". SpaCy uses transition-based algorithms to find important information close to the entities. It is mentioned however, that if the entities tend to be long and characterised by middle tokens, it might perform poorly [15]. However, as the training data is planned to be in short sentences and is labelled in IOB format, the expectation is for the pipeline to perform adequately on this problem.

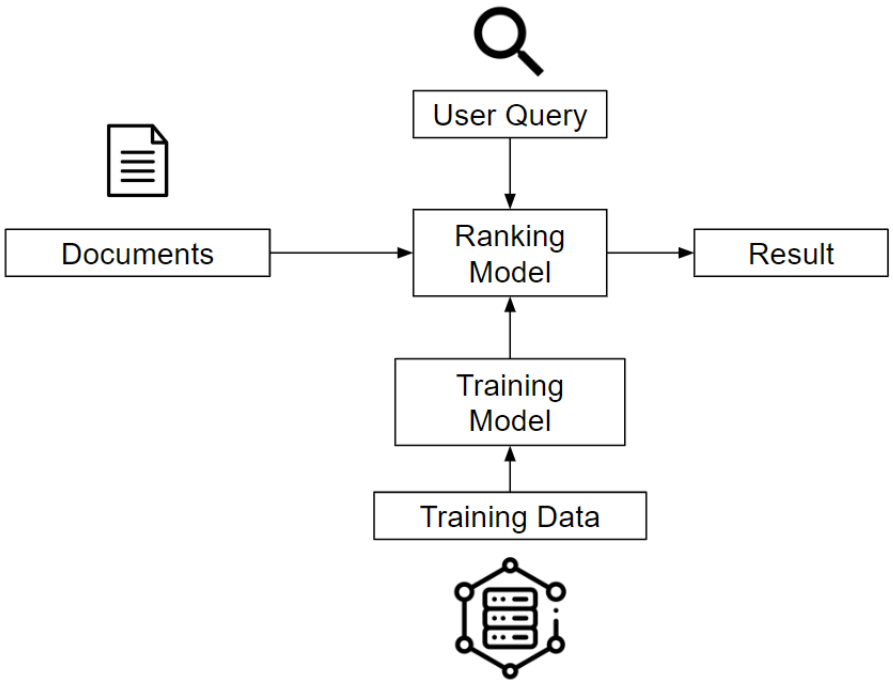
2.4 Ranking Strategies

2.4.1 User Interaction

While the NER model is capable of recognising entities, which on their own perform a similar function to a summary, these labels are often disjointed from one another, removing meaning, as well as context, and thus would take significant time for a human to sort though. With the end goal of the system being capable of ranking candidates to a query and presenting the end user with the best possible applicant, a ranking model is required alongside labelling. In a paper released in 2023 for "Resume Analysis using Machine Learning and NLP" by International Research Journal of Modernization in Engineering Technology and Science [16], methods such as entity recognition, semantic analysis and sentiment analysis are outlined as potential techniques.

Pretrained models like BERT already support sequence classification [63] and transformation [60], while rule based methods can be applied to scoring labels

via match queries. Regardless of the method, the query input by the user must be able to be linked with the potential resumes given. Thus, the interaction with the user interface should be as follows:

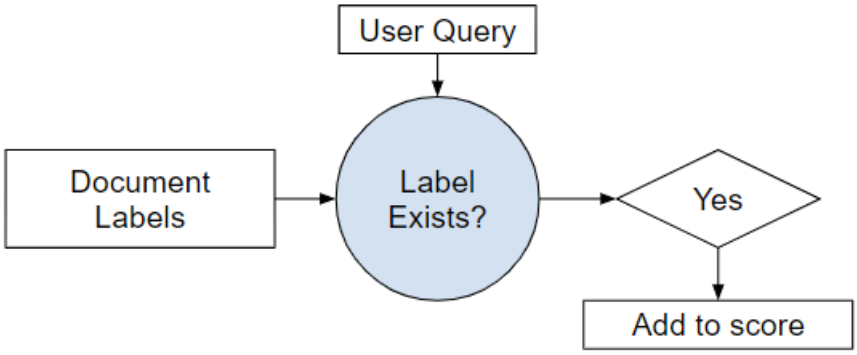


GUI for Ranking Diagram [40]

2.4.2 Rule Based Ranking

The most basic form of ranking is rule based ranking, often combined with data or text parsing. This method relies on defining a ranking criteria. The rule based system will require the labelling model to have extracted key labels, such as skills and years of experience. This would then be used to develop a set of rules that look for similarities within the user query and add scores from the relevant matches.

Each label will be weighted depending on the relevance to the job, especially if it is a critical skill. With the scoring applied, the resumes can be ranked and this can be iterated quickly for selection of the candidate. While the rule based system does lack the flexibility of a more advanced model, they can still provide a solid foundation to compare against when researching rankings.



Rule based scoring diagram [46]

To test the accuracy of a ranking system with a rule based approach, exact label ranking can be tested trivially by checking if any of the user query values exist within the document labels. The problem with this approach is recognising "context" and "intent" from the user. For instance, if the user were to try to enter a requirement for the applicant to have "a bachelor's degree" for education and the NER model has picked up education, but it is written as "BSc", this would not be recognised.

An approach to improve this would be to stem words, as well as use pattern matching[58] options, though this can lead to false positives where a wrong answer is picked up as correct and score is added. For instance, the root for the word "genre" and "gender" is "gen" and if no further processing is done on these words, this would be labelled as a match, causing the score to increase, despite having completely incompatible meanings. Because of this, in real world scenarios, we need something more intricate to deal with ranking.

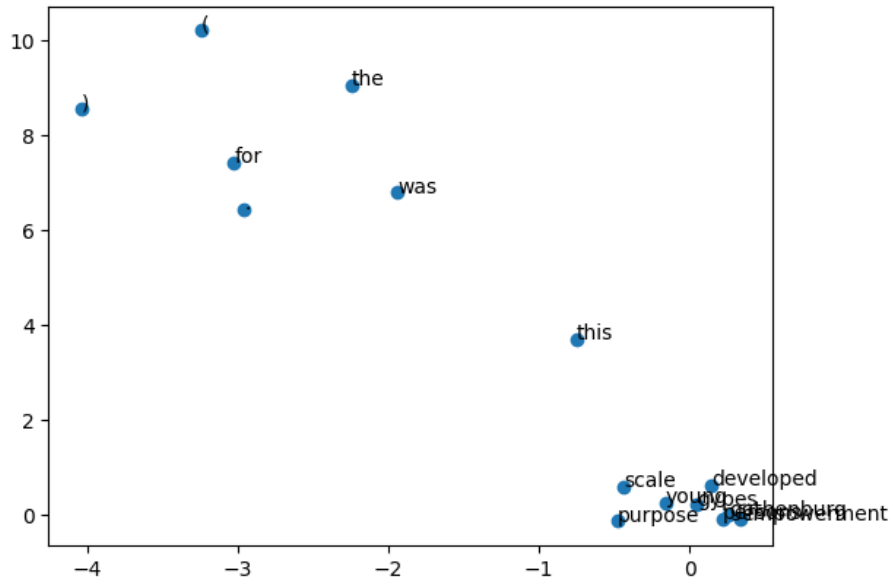
2.4.3 Word Similarity Based Ranking

Another way to handle ranking is to use "word similarity". Word vector libraries like Word2Vec [74] , FastText [73] and Glove [18] can be used to train a completely new corpus of words, or alternatively, a pre-trained model with vectors can already be loaded to start comparing the words.

In order to perform word similarity calculations, words must first be turned into mathematical representations of themselves. The simplest form of this comes in terms of numerical encoding - each word is assigned a unique index. However, taking such an approach with a for loop, would not infer any meaning to the words, as regardless of context or word "closeness", randomly assigning unique ids would not give interpretation.

Because of this, another method for assigning unique values to the words can be used - vectors. The purpose of each vector is to show how close a word is to

another word. For instance, the words “king” and “man” should be quite close to each other in vector representations, however, the word “woman” would be further away from them. However, both “man” and “woman” would be closer to the word “person”. Thus we could perform a calculation where we could get “king” - “man” = “queen”. If the size of the vectors was 2 dimensional, this could be plotted on a graph:



Vectorised Words Plotted on a 2D Graph [49]

From the above it becomes visually clear that stop words and punctuations are much further away from words with meaning, separating them in both context and meaning.

This means that each word in a corpus can be represented by any sized vector. To calculate these vectors, different methods can be applied. For instance, Word2Vec has two methods it can use to calculate word similarity - CBOW (Continuous Bag of Words) and skip-gram [23].

In CBOW the vector sum of every word’s neighbours should be close to the vector of the word itself. In skip-gram the vector of a word should be close to the vector of each of the neighbours. The main difference here being that CBOW will calculate the word vectors based on the sum of the neighbours, while in skip-gram, the vectors of a word are calculated from the neighbours values themselves [74].

In terms of CBOW, it tries to maximise the quantity using the following formula:

$$\sum_{i \in C, j \in N+i} (v_{w_i} * v_{w_j} - ln \sum_{w \in V} e^{v_w * v_{w_j}})$$

Where

1. "V" stands for "vocabulary"
2. "v_w" stands for "one vector for word".
3. "w" stands for "word"
4. "N" stands for "neighbour"
5. and "C" stands for "corpus of words"

While in skip-gram it is the following:

$$\sum_{i \in C, j \in N+i} (v_{w_i} * v_{w_j} - ln \sum_{w \in V} e^{v_w * v_{w_i}})$$

The only difference being the last index.

While Word2Vec has functionality already built into it to calculate similarity between two words, the same can be achieved using cosine similarity. It is calculated between two same dimensional vectors to find the angle between them. This is also pointed out in the 2019 Word2Vec model analysis [75]. The equation for cosine similarity is as follows:

$$similarity = cos\theta = \frac{\overline{x} * \overline{y}}{||\overline{x}||||\overline{y}||}$$

Similarly to Word2Vec, FastText also uses CBOW and skip-gram approach. However, FastText works on a character n-gram level, while Word2Vec works on the word level. Consider the example "wilds" and n-gram = 3, the representative n-grams would be:

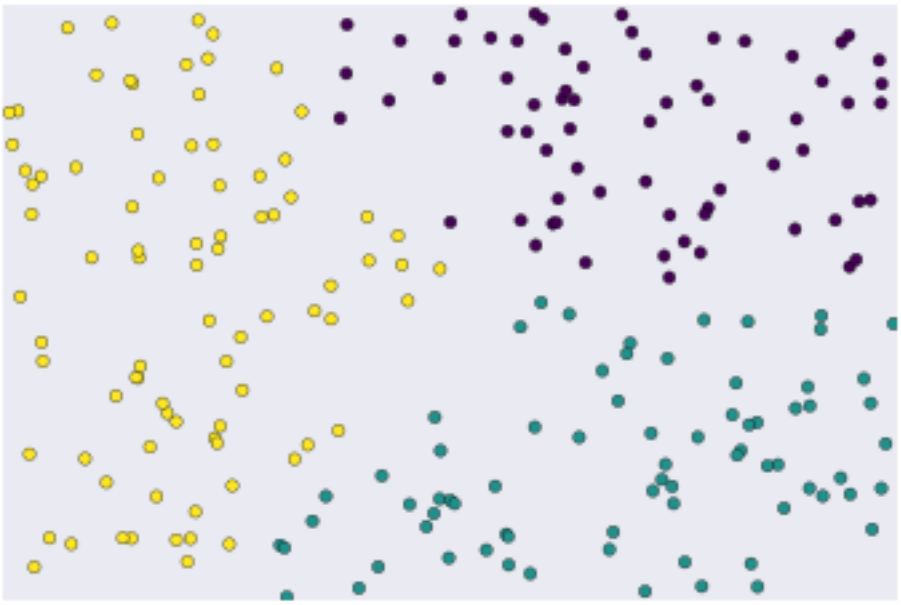
wi, wil, ild, lds, ds and wilds

The embedding of the word wilds can be given as the sum of all vector representations of character n-grams and the word itself [23]. With FastText also using hierarchical classifiers, it is faster than Word2Vec.

Using similarity ranking can be a very powerful approach to rank unstructured data that has close meaning to the original query. However, the issue with this approach is with specifics. Consider two resumes for an information technology specialist and a database engineer. While on first glance these resumes might have a lot of similar words within them, they are for two completely different positions. Thus an information technology specialist might be recommended for a position as a database engineer or the other way around. Of course, this can be mitigated by adding a rule based approach over the top of it, combining the two techniques.

2.4.4 Clustering

On top of the vector space technique, clustering can be applied. The method of clustering is to take a group of words and combine them into "clusters" or "collections" of any size [70]. The reason why this remains a powerful technique is that multiple words that are close to each other can be represented by the same vector, allowing for quicker calculations, as well as meaning between multiple word groups to be inferred.



Example of clustering into 3 groups, represented as a 2D graph [70]

Different clustering techniques can also take different meanings. Take for example a resume where at first we use the word similarity calculated from vectors to find their positions on a graph and cluster them by closeness to a search word "engineer". However, what if the search would not matter about closeness to a skill, but rather a closeness to experience spent as an engineer. The cluster presented would be different and thus be more relevant to the query that we are performing.

Methods like sentence transformation already have this included in their pipeline when using it on pure texts [66].

2.4.5 Ranking with BERT

Being a state-of-the-art language representation model, BERT is used widely in various NLP tasks. As mentioned in the labelling as a ranking strategy section,

BERT is capable of ranking texts. This is done by using language semantics to rank documents or items based on relevance to a given user query. BERT's capabilities allow it to be used on texts without having to label resumes. For the purpose of this report, we want to test both with and without labels and compare the results [10].

BERT is based on transformer architecture [68] which allows it to capture context of the given sentence. This context is particularly useful for understanding semantic meaning of the user queries and corpus text. While BERT is already pre-trained on a large dataset to understand the aforementioned semantics, it can be fine tuned to perform ranking specific tasks by using a specific layer. Labels can also be used to indicate the relevance for documents.

Two ways BERT can compute ranking are pointwise ranking and pairwise ranking [56]. The pointwise ranking has each document to query pair treated as an individual data point and assigned a score. In pairwise ranking, document pairs are compared to a query and based higher or lower depending on their relevance to a given user query.

BERT is also capable of leveraging sentence transformers and sequence classification. In sentence transformation, easy methods are provided to compute dense vector representations of a word, enabling semantic analysis, clustering and word retrieval [66]. In sequence classification, the entire context of a sentence is taken into account, helping to better understand the similarity and meaning [63].

By using BERT, we can leverage pre-trained models, meaning that even if the dataset that the ranking is performed on is relatively small, it can provide relatively reasonable results.

2.5 Flaws And Solutions

2.5.1 Labelling and NER

While the aforementioned techniques provide a solid foundation to start from, there are weaknesses to these approaches. In order to label data with NER, we require a substantial amount of data to first train the model. This requires the end user to either find datasets that have already been set up with the IOB (or similar) format or add to their own labels. The former is reliant on these datasets pre-existing, while the latter would take up a significant time.

This problem would arise when the system has just been integrated by an end user. However, over time, the more data from different resumes the end user has, the better and quicker the result for training becomes. With each recruitment cycle data is added and thus gives the model more to learn from. If the

model is trained with enough variety, it will no longer require training data, just as pre-trained models are capable of generic tasks.

Another issue with training is label accuracy. This means that either the datasets used need to be organised in an effective manner or the end user will need to have a basic understanding of what constitutes a "good" label. Having bad/malformed labels means that the trained model will be significantly worse at recognising labels and thus not provide the expected result.

The benefit of using pre-trained models is that the end user does not have to think about labelling data if the model has already been trained with the labels they are looking for. If not, starting from a checkpoint that has been pre-trained for sentence semantics will take much less time than training from nothing. Having a large enough dataset to train on means that language semantics will be able to predict any job, regardless of if it has been seen before or not. This will allow for similarity ranking without having to create predefined resume topics.

2.5.2 Resume Ambiguity

What remains outside the end user's control, is how the resumes themselves are written. Because of the nature of unstructured data, there is a possibility of valuable candidates being missed due to not entering the right keywords that will be recognised by the NER labelling system. This will lead to the candidate scoring lower. A rule-based system that only performs word comparisons during the ranking is especially susceptible to such an error.

To overcome this, a number of techniques can be applied on top of regular comparison that will help the system understand the context of words and sentences. Approaches such as stemming and lemmatisation, vectorisation, term frequency of word in documents and cosine similarity all allow for additional meaning to be applied on terms found by the NER model, ensuring a more accurate scoring. The methods brought out here are supported by another research paper released in 2020 for "An Automated Resume Screening System Using Natural Language Processing and Similarity" [7]. Of course, by also improving entity recognition within the model, the score will also improve.

3 Design

3.1 Requirement Analysis

3.1.1 Software Requirements

1. Python 3.11.x or greater
2. Node.js 18.17.x or greater
3. Windows OS or Linux OS
4. PyTorch 2.2.1
5. CUDA 12.1
6. Transformers 4.36.2

3.1.2 Hardware Requirements

1. Nvidia GPU for training with CUDA

3.2 GUI System Architecture

3.2.1 Approach

To make the project understandable, the frontend GUI prioritises usability and ease of use for the end user, to quickly and simplistically make iterations on the datasets, as well as receive critical feedback. To avoid complications, the mechanics of the architecture attempt to remain straight forward - a python websocket server that connects the main python runner and the frontend renderer with Electron. All of the data manipulation, model training, data parsing and request/response handling is done on the backend python server. The frontend will only communicate user requests and visualise responses.

Creation of this system allows for increased reliability in data parsing and running commands without having to rely on a singular file structure where each function acts independently without knowledge of other systems or capability to link with them. Having a singular hub also increases robustness and makes sure that any new code written will ensure that previous systems function as intended. This creates a straightforward approach for adding features and visualising existing data to the end users.
.

3.2.2 Front End Design

The frontend is created using Electron, Vue.js and Vite.js to handle running real time update servers and hot module replacement to instantly receive feedback when code changes are made. With initial setup, TypeScript [65], CSS and HTML templating is supported as a default. The layer above Electron is Vue.js that handles data-binding and allows dynamic updates without having to make ajax [5] requests or create Axios [17] integration. This allows for quick template creation for any of the pages required to showcase the app features.

To keep track of the app state, a file with selected global options is saved to the python server. This state file is kept in sync with a local copy on the front end that is saved to a Pinia [22] store using Vue.js. This ensures global state sharing within the frontend and backend. Due to many of the modules requiring knowledge of which dataset are currently being used for labelling or training, both systems need to be aware of the user selections. This also ensures that when the app is restarted, user settings are not lost and do not have to be reconfigured.

The system is divided into 5 main parts - uploading and parsing data into key JSON datasets, visualisation of labelling and errors within each individual dataset, training new models from a given dataset, visualising the model accuracy and labels found from trained models (NER) and ranking the resumes and visualising their rank order in a dataset. The frontend also ensures the user selects a dataset and does not expose functionality that would not be possible without selecting it. This also includes not visualising model accuracy without selecting a model for either NER or ranking.

3.2.3 Backend End Design

The backend separates into 6 distinct parts - data, parsers, server, state, training, and utilities.

The data section of the server holds the datasets in a JSON format as arrays of resume string and entities. Each entity has a start and end index, noting its position within the text, coupled with a label name.

The parsers handle changing resume data into distinct objects that will hold labelled data, entities and error entities. The entity itself is also broken into pieces, with the start, end and label having their own values. The parser entity allows for quick operations on a number of resumes, such as getting the entities or the resumes in a dictionary or formatting the entity order. The parser also handles finding errors in resume strings, such as overlapping indices or cleaning non space starting and ending labels.

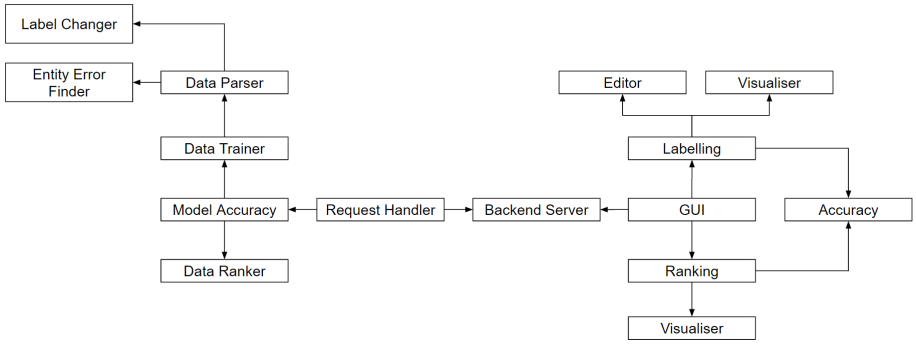
The server is responsible for all the communications between the frontend and backend and serves as a middle point for handling breaking requests into responses and forwarding to their respective task handlers. The task handlers then return responses back to the frontend. The structure of the responses are down to the task handlers.

The state holds a JSON file that saves the global values that should be known by both apps for the ease of querying. While the frontend holds a copy of the state in the Pinia store, any changes to the state will first be made on the server and propagated to the frontend.

The training part of the backend handles model training and getting pre-trained models to run accuracy tests or ranking. This includes training to recognise labels and use the trained model to label raw texts. The system is also able to use the label data taken from the raw texts and to use it for ranking. The ranking system is built on top of sentence classification.

The utilities folder performs reading and writing, as well as alerting the frontend with errors or successful task completions. The frontend is capable of showcasing these alerts regardless of the current interface the user is interacting with.

3.2.4 Full Architecture



Visual Diagram of GUI System Architecture [50]

The collective user interaction system is separated into the following parts:

1. Data Parsing
2. Graphical User Interface
3. Data Label Editor
4. Data Trainer
5. Model Accuracy Viewer
6. Data Ranker

Data-Parser - Uses Python to read and parse PDF files into JSON files or read premade JSON files with entities already created.

Graphical User Interface - Uses Python Websocket server to pass messages from frontend to backend. Uses Electron as means to render a windows application, as well as gain access to web based rendering (HTML, CSS, JavaScript) via chromium. As a reactive framework, it uses Vite.js and Vue.js to render and Typescript to provide typing information.

Data Label Editor - Is able to read the entities from JSON, check them for errors, including overlapping entities and non space ending or starting entities. Also provides capabilities to fix said labels, add new labels or delete problematic

labels.

Data Label Editor - Is able to read the entities from JSON, check them for errors, including overlapping entities and non space ending or starting entities. Also provides capabilities to fix incorrect / errored labels and add new labels. Can also delete problematic labels.

Data Trainer - Allows users to take a pre-existing dataset and use a method to train their own model with it or to use a predefined model to find labels. These labels are parsed using NLP techniques and include skills, education and years of experience from the applicant's resume. This creates a structure to unstructured data and makes it easier for the ranker to analyse.

Model Accuracy Viewer - Ability to view how well a model finds labels comparatively to existing ground truth labels and calculate the accuracy of the model. This will allow for comparison of between model precision and to visualise labelling or ranking errors.

Data Ranker - Taking either full resumes or labels and using them to rank resumes to user input or a job description. The result will provide a ranking of resumes from best to worst.

3.3 NLP Model Design

3.3.1 Datasets

For both training and ranking, finding the correct datasets is crucial to ensure that the data is relevant, comprehensive and of high quality. This will provide the models with the necessary context and examples to perform training effectively. To perform NER tasks on a selection of texts, first a group of resumes is required along with ground truths for tags. For a selection of resumes, the "LiveCareers" dataset [57] posted on Kaggle has over 2400 resumes which could be used for ranking, offering a range of topics and resumes as strings. However, these resumes do not hold any entities. In order to perform labelling, the Laxmimerit dataset [52] has resume data with the required entities to be able to train for entity recognition.

In order to test ranking accuracy a dataset has to be used where a user query matches a number of resumes for a given query. As there are no publicly available datasets for this issue as of May 2024, an alternative approach had to be taken. Creating such a dataset by hand would not be optimal, especially if the chosen method does not perform well and needs to be rapidly improved in the pipeline.

The binoydutt dataset [59] has a list of job descriptions with topics, while "InferencePrince555" has created AI generated resumes on Hugging Face [21] with

each resume also holding a topic. The job descriptions can be used as a "search query" and resumes with relevant topics can be linked to the job description by IDs. As the topics will not match exactly, the matching might also require stemming and data parsing (for instance removing generation query from the topic in the resumes).

3.3.2 Ranking Search Queries

Using the linking of the topics for job descriptions and resumes, a similarity match can be performed on texts to ascertain their relevance to a given job description. This allows to perform different similarity checks and queries on the same dataset. The weakness in this approach is "subject relevancy". As currently the search queries will only be performed on the similarity of topic, they will not take into account required degrees, years of experience, etc. These can be improved with sufficient regex and data parsing methods that will assist in their discovery and ranking.

Five methods will be used to perform ranking - sentence classification, sentence transformation, labelled sentence classification, labelled sentence transformation and rule based ranking. This will give a good subset of differences and showcase if using labelled data will outperform using the entire collection of the text.

3.3.3 Training and Performing NER

As described within the literature review, entities can be formed in two separate ways - using text indices or using word indices. The Laxmimerit dataset has the former approach and as such, this will be the first approach taken to entity recognition. When outlining labels to train for, a general list of labels is outlined in the 2023 hybrid approach paper mentioned in the literature review [2] and is as follows: Name, Contact, Education, Languages, Skills, Projects, Experience, Certifications, Publications and Volunteering. For the purposes of this report we will be focusing on three of the above labels, education, skills and experience, while focusing on a fourth - years of experience. These labels will yield the greatest change within a resume as someone with a set of relevant skills, but without a degree or enough years of experience might not be suitable for a job.

While experience and skills are similar in terms of ranking strategy as both will attempt to find and match to relevancy within the query, both labels pertain different meanings and are thus important for the model to distinguish between. By making the model understand specifics or a label, it will not confuse it for another similar word. For example, years of experience can be represented in a numeric form, however, every number is not a year of experience. These numbers can be quite similar to each other as given to an example below:

01/03/2013 - 01/03/2014 - *Year of Experience for first of March, 2013 to first of March 2014*
01-03-1024 - *A bank sort code.*

By explaining via IOB tagging to the model that "01-03-1024" is a "bank sort code" and tag it at the start with "O-", the model will learn to find sort codes and exclude them from the labelling.

As both spaCy and BERT have models that are already pre-trained for entity recognition, both of them will be fine tuned to the datasets and labels given.

4 Implementation

4.1 Assumptions

4.1.1 GUI

The implementation of the system assumes that the user is capable of interacting with the GUI in order to upload data and select datasets for the purpose of labelling, training and ranking. The system also assumes that the user should be capable of recognising label errors given by the system and fix them using the inbuilt tools or their own methods.

The labels have to be in a format as defined in the full architecture section of this report. The system assumes usage of pre-trained models for users that do not want to train their own models. The datasets are assumed to be in JSON or PDF format. The PDF format will only upload resumes and not the entities. For entities to be included, the data must already be labelled and uploaded as JSON.

The usage also assumes that should the users want to improve the NER pattern model, they are able to add their own patterns within the patterns folder. However, this is not a requirement of using the system.

The current version of the system also assumes the user to be able to install the required packages for the project. Vite comes with a builder version that could be implemented to build the project into an executable. Python code could be improved with CPython to build into an executable for ease of use. Currently, the GUI is run through a "start.bat" file.

4.1.2 NLP Models

The current implementation of the system does not include all the versions of the models. These are instead separated into their subfolders with ".ipynb" or ".py" files. This decision was made due to time limits, as well as to be able to more quickly process iterative changes without having to run the graphical user interface at the same time. The model research is also separate enough from the graphical user interface and visualisation, that it can be run as its own sub-project.

The current ".ipynb" notebook files allow for data parsing to combine datasets into one for job descriptions and resumes, cleaning datasets of whitespaces and non special characters within resumes, training IOB datasets for named entity recognition and using multiple ranking methods to test method accuracy on the dataset.

Assumption on the user is that they are capable of installing required packages via pip install for each of the notebooks. The notebooks also assume that the users are capable of running jupyter notebooks and are able to understand the data shown to them.

4.2 Dependencies

4.2.1 GUI

Required Backend Packages:

1. Pandas (2.1.1)
2. Numpy (1.26.1)
3. spaCy (3.7.2)
4. WebSockets (12.0)

Required Frontend Dependencies:

1. electron-forge/cli: 7.2.0
2. electron-forge/maker-deb: 7.2.0
3. electron-forge/maker-rpm: 7.2.0
4. electron-forge/maker-squirrel: 7.2.0
5. electron-forge/maker-zip: 7.2.0
6. electron-forge/plugin-auto-unpack-natives: 7.2.0
7. electron-forge/plugin-vite: 7.2.0
8. typescript-eslint/eslint-plugin: 5.0.0
9. typescript-eslint/parser: 5.0.0
10. vitejs/plugin-vue: 5.0.3
11. electron: 28.2.0
12. eslint: 8.0.1
13. eslint-plugin-import: 2.25.0
14. sass: 1.70.0
15. ts-node: 10.0.0
16. typescript: 4.5.4
17. electron-squirrel-startup: 1.0.0
18. pinia: 2.1.7
19. vue: 3.4.15

4.2.2 NLP Models

Required Model Dependencies:

1. spaCy 3.7.4
2. Transformers 4.36.2
3. PyTorch 2.2.1
4. CUDA(nvcc), release 12.1, V12.1.66
5. Torch+CUDA 2.2.1+cu121
6. NLTK version 3.8.1
7. Scikit-learn version 1.3.2
8. Matplotlib 3.8.0
9. Pandas 2.1.1

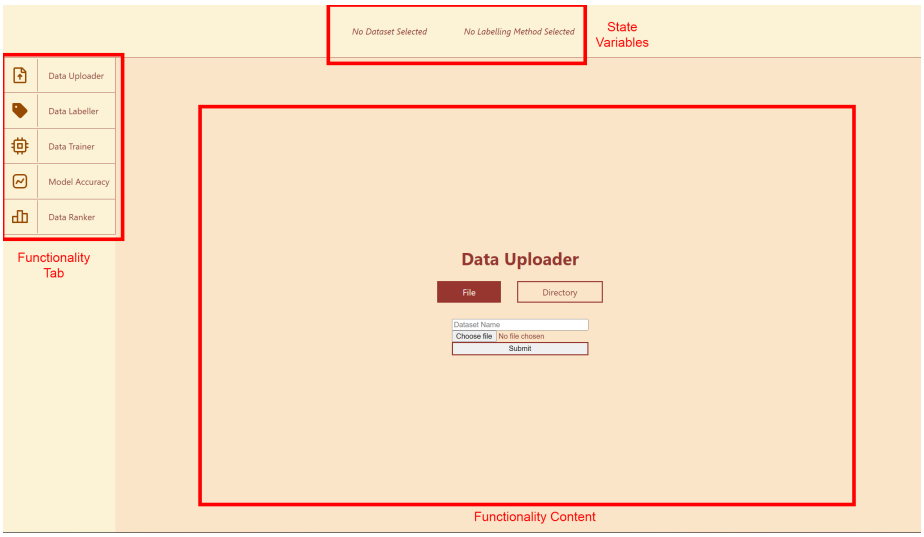
10. Numpy 1.26.1
11. Datasets(HuggingFace) 2.18.0
12. Gensim 4.3.2

4.3 Implementation Front-End

The implementation of the project starts by running "start.bat" that launches the main.py python file. This file is responsible for starting both the backend and frontend systems that run as one whole - should the backend be shut down, the front end shuts down as well. The front end can be shut down without destroying the backend. This is so that should a non GUI be used, a simple UDP server could still communicate with the backend system.

4.3.1 Main UI

The frontend runs a visualiser in electron using yarn commands that will display a GUI to the user:



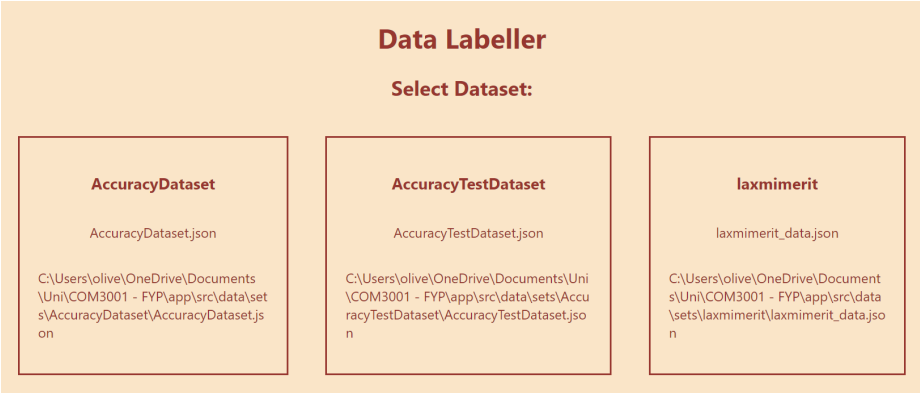
Front Page Displaying Content [37]

As seen from the diagram above, the left hand side controls the different functionality tabs for the user. The state variables header allows the user to visibly see which dataset and model they have currently selected, as well as any other consistent variables. The centre of the page remains for the content of the application itself. This layout provides the user with an easy understanding of what dataset they are currently using and what are possible functionalities of the system.

The first tab, data uploader, deals with selecting and uploading a JSON file or a directory of PDFs. If the end user already has a dataset constructed with entities in a JSON format, they can upload it as a file. However, if they have a set of resumes with no entities, they can upload them as their own dataset, with them written into a JSON file with empty entities. This is managed from "DataUploader.vue".

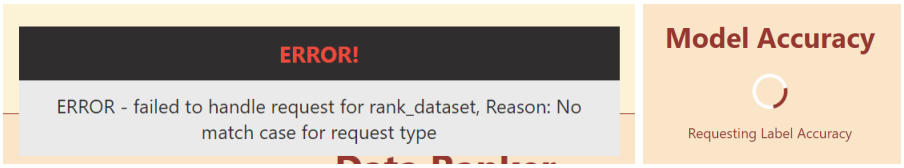
All of the functionality for the ui is handled by "MainUIManager.vue", that controls tabbing functionality and loading other utility components that are supposed to persist throughout the app (Alert.vue and AppState.vue). The MainUIManager is loaded by "App.vue" that is the base vue component.

4.3.2 Reusable Components



Data Labeller before a dataset is selected [26]

There are few reusable components that are utilised along the project. An example of this can be seen when the dataset tab is selected for data labelling. None of the data labelling functionalities are performed until the user has selected a dataset. To be able to select a dataset, the component "QueryDatasets.vue" presents the users with the ability to select datasets. The QueryDatasets.vue is a vue template that can be included into any other template to force the user to select a dataset before they can access the main functionality of the tab. Currently, all components other than the data uploader rely on a dataset being selected.

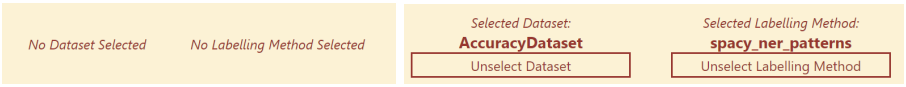


Example of error alert [29] and *the loader component* [34]

To alert the user if something has gone wrong or if some task has been successfully completed that does not have an immediate visual feedback, for example uploading resumes as a dataset, an alert system has been implemented using the "AlertManager.vue" component. This component is added to the "MainUIManager.vue" component and is placed within a slot that overlays the rest of the UI. Handling alert capturing is done in "alert_handler.ts".

The loader component is a reusable component that is added to most request components. This is to make sure that the user is notified if something is expected to process for a while on the server.

4.3.3 Global Values



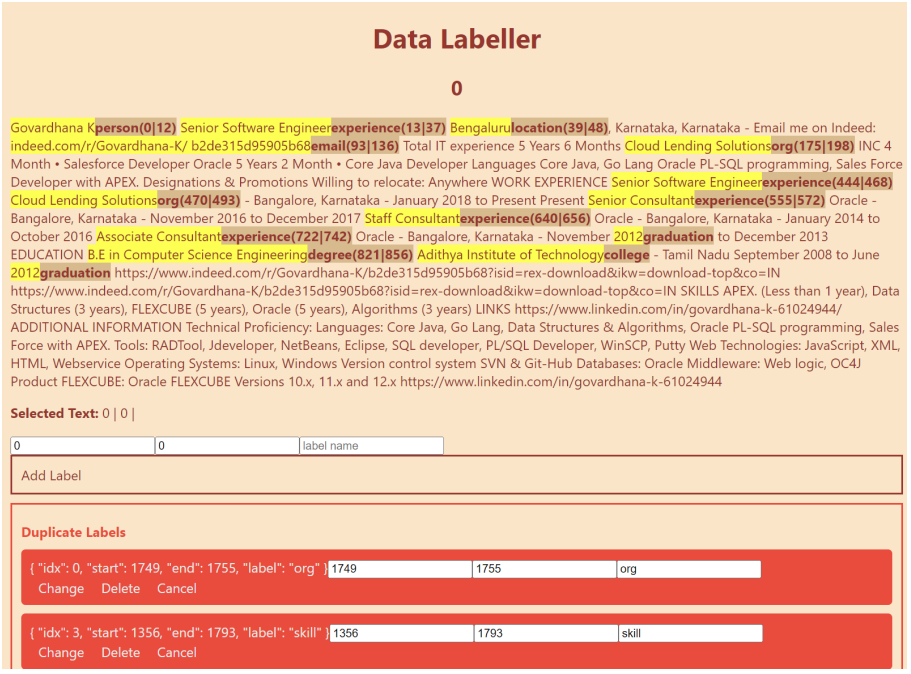
Example of no-state state with Dataset and Model selected [31]

The header of the app visualises the current selected Dataset and Models used to do either labelling or ranking. These can also be unselected from the same controls. If a dataset or model is unselected, the system automatically detects it by subscribing to notifications from "app_state_handler.ts" using "Pinia" and forces the user to reselect the dataset and/or model where required.

4.3.4 Request/Response

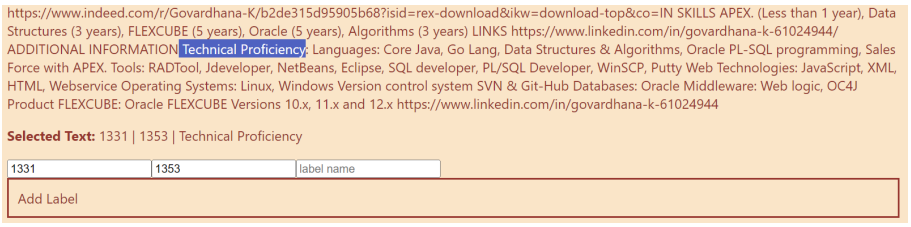
All of the communication between the backend server and the frontend GUI is done by "request_handler.ts" and imported into each of the components that requires communication with the server. This allows for the creation of a robust sequential framework that passes requests to the server and waits for them to be resolved before continuing with the next task, rendering the response.

4.3.5 Data Labelling



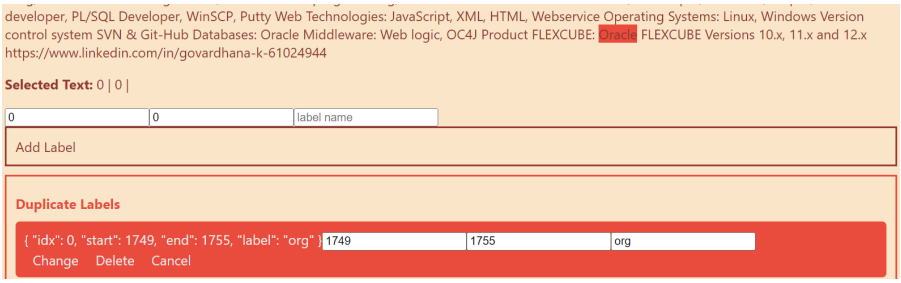
Example of Data Labeller [27]

When a dataset is selected, each resume within the dataset is visualised with any entities that are present within the dataset format. The text of the entities are highlighted in yellow. The label corresponding to the entity is added after the text of the entity, along with the text indices of start and end of the label, as shown on the image above.



Example of adding a label [24]

Text within the resume can be selected and this will show on "Selected Text" output with the selected text indices and the selected text. A name to a label is required to be added and can then be submitted to add to the entities of the resume.



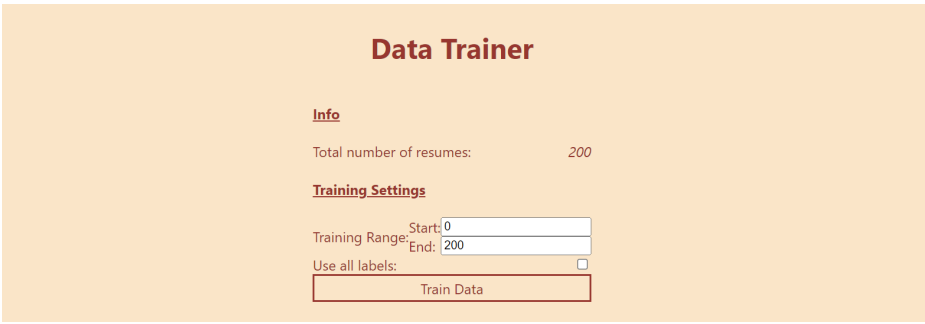
Example of error labels [30]

Any entities deemed to be in a "non trainable state" are highlighted in red when the error entity text is selected. It is down to the user to find the best solution to resolving the conflict between the labels, either by changing the start and end indices of the label or to completely delete the label and add new labels instead. During the training process any label that is considered as an "error label" can be excluded. This is done to ignore the labels that have not yet been fixed by the end user, while still being able to run the training sequence. The currently recognised labels for errors are:

1. Overlapping duplicate labels - entities that overlap in indices with another entity, for instance an entity that spans from 125 - 150 and an entity spanning from 135 - 160 would be considered "overlapping".

2. Non-full strings - entities that do not start or end with a "space" character. Most training models do not handle labels in the centre of text strings. For instance labelling "linked-in" as org (organisation) in "https://www.linked-in.com" would be considered an error.

4.3.6 Data Training

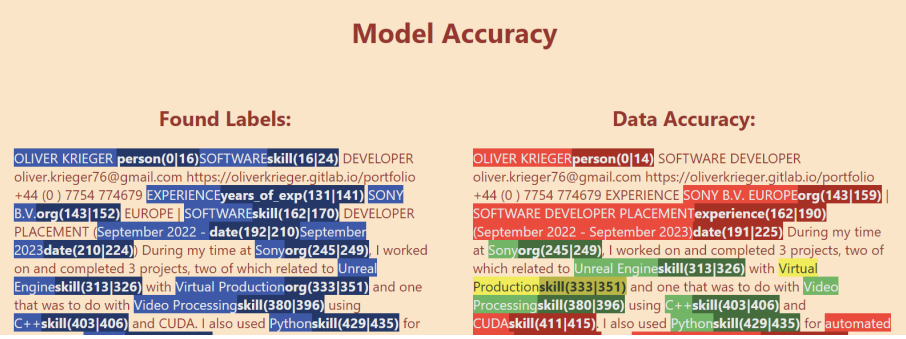


Example of the Data Training options [33]

The data training tab currently requires only a dataset to be selected and will train it with spaCy NER model. The training options say how many resumes

are in the dataset and allow for a range of CVs to be selected for training if the user wants to only train on certain resumes. There is an option to use broken labels if the user wants, if all labels are fixed but the system believes some might still be erroring. This is to add to user control should unstructured data result in unexpected errors. By default it is recommended to leave the options unmarked. An improvement to the existing system would be adding which model the dataset should be trained with, so that the datasets could also be trained with other NER models.

4.3.7 Model Accuracy



Example of the Model Accuracy tab for NER [35]

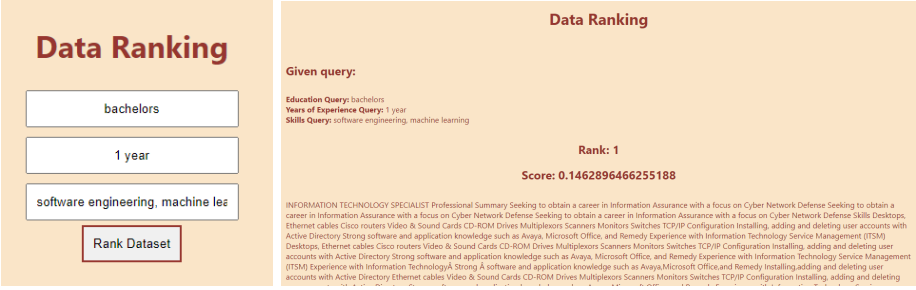
The model accuracy tab allows for testing dataset accuracy for a given model to find named entities. The left hand side displays the found entities by the model in blue and the right hand tests the ground truth entities against the resume to see how many labels were correctly predicted. Labels in red are labels that were not found, green labels are correctly found labels and yellow labels are ones where the indices of the label were correct, but the model incorrectly labelled the entity. For instance, the image above shows that virtual production is labelled as an organisation, but should actually be labelled as a skill.

{ "all_labels": 52, "correct_labels": 13, "incorrect_labels": 5, "missed_labels": 34, "accuracy": 0.25, "precision": 0.7222222222222222, "tpr": 0.2765957446808511, "fnr": 0.723404255319149 }

Example of numerical accuracy [32]

A summary of the numerical accuracy calculated by accuracy, precision, true positive rate and false negative rate can be found at the bottom of each resume. The numbers are calculated using a confusion matrix. It also lists the total number of labels in the resume, how many labels were correctly found, how many were incorrect and how many were missed.

4.3.8 Ranking



Example of data ranking results [28]

The data ranking tab allows for the user to enter their own query for education, years of experience and skills to rank the resumes found within the dataset. The current GUI ranking uses sequence classification to compare similarity between the search query and the documents. When the ranking is finalised, the server returns the rank position for a resume, the score, and the resume in text format. Future improvements would see the text be a separate load in option once ranking is finished, as not to overcrowd the user with information.

4.4 Implementation - Back End

4.4.1 Server Implementation

When "main.py" runs from "start.bat", the python backend starts a websocket server that will wait to hear requests from the front end GUI or a UDP server to handle tasks. All possible requests are added to "request_handler.py" that is called when the server gets another request. This request is broken apart on the "request_type" and an appropriate sub handler is called to complete the task or respond with an error.

4.4.2 Data

The data is separated into two parts - datasets and patterns. The uploaded datasets end up in the dataset folder in the appropriate JSON format. NER patterns that are used by spaCy are also added to this folder. For future improvements, the patterns should also be uploaded along with the files or separately, so that they may be selected for the pattern model.

4.4.3 Parsers

Throughout the server, "data_parser.py" is used to read resumes and mutate the data within the parser. For new requests the parser is reconstructed. While this is less efficient than having it loaded into memory at all times, it ensures data consistency and avoids desynced states. This approach also follows the

best practice of distributed systems in which the system reconstructs objects from the beginning each time, sacrificing efficiency, but gaining robustness.

The data parser takes all the resume data from the JSON files and makes them into objects that hold data on the resume string, labelled resume (where the labels are added to the text for visualisation), entities (parsed into their own objects) and error entities (found by the "error_finder.py"). The parser allows all resumes to be made back into a dictionary string that can then be responded with to the front end to visualise. The resumes also hold a topic that can be used during the ranking process.

4.4.4 State

The state of the backend keeps the state in a JSON format and reads and writes updates to it as required. Any changes that are made to the state must first propagate to the server and respond to the frontend once the updates are complete on the backend. The reading and writing of files are managed by utility libraries in the "utils" folder.

4.4.5 Training

The training commands go to the "train_model.py" that then handles sending the dataset in parser format to the model itself. The NER model used will then be trained with a spaCy model that first splits the entities into an understandable format for spacy and creates a pipeline that allows for training.

In the case of calculating model accuracy for resumes, a pre-trained model is loaded first. This means using spaCy default, using spaCy with patterns that have been loaded from the patterns folder or using a different, pre-trained model. The resumes are passed through the model, which finds and appends labels to trained_entities. Accuracy is then calculated by using resume entities and comparing the index locations and labels for each entity.

While the trained model is as good as the amount of training done on it and relies on the model weights to be adjusted appropriately, for the pattern approach adjusting patterns with new data can increase the accuracy without having to change much in terms of the model. Labelling something as a specific label (C++ is a skill) or defining more complex patterns (years of experience can be a number followed by a month, a year, a day, etc) can add to the understanding of the pattern matching algorithm.

4.4.6 Ranking

Similarly to training for NER, the ranking algorithm will decide on the request which method to use for the resume ranking. The most basic approach to this is to take all the named entities found by the NER model and compare to the skills and years of experience given by the user input, adding +1 every time we encounter a label or enough years of experience in the list.

However, a more sophisticated approach is to use the BERT pre-trained model by taking a user query and scoring from it to adjust document rankings. To test accuracy for this, we can add a list of test queries and expected resume rankings for each of the query, then compare the model rankings for each of the queries given. This helps us create a ground truth to train the documents against. This method requires a lot of user input to create a dataset.

A resolution to this is to use different datasets - one dataset for resumes to topics and one dataset for job description to topic. From these two datasets a third can be created where the job description is linked to every resume of the same topic and thus, these resumes should be ranked the highest, while other resumes should rank lower. This will give a metric to measure how well the model is performing.

4.5 Implementation - Labelling

4.5.1 Data Analysis

As the main focus of the report, the first step to being able to rank resumes was to build a NER model, capable of finding four labels - work experience, years of experience, education and skills. As this project started with spaCy, the default named entities did not include any of the aforementioned labels. Because of this, another dataset with pre-labelled data was required to adjust spaCy's entity recognition.

The original livecareers dataset found on kaggle had resumes with topics. However, as it had no label data, it was deemed unusable for training entities early on in the project timeline. The next dataset to explored was Laxmimerit. Laxmimerit is a dataset consisting of 200 resumes and 3206 labelled entities from the "CV-Parsing-using-Spacy-3" repository [52]. It has 11 unique entities:

1. Years of Experience
2. Name
3. Skills
4. Email Address
5. Designation
6. Companies worked at
7. College Name
8. Graduation Year
9. Degree
10. Location

From the 11 entities, all 4 entities exist that are required for ranking.

4.5.2 Data Processing

When first attempting to train the laxmimerit dataset it became obvious that it had incompatibilities with the spacy system that did not allow it to be trained. Mainly, many of the entities were misaligned within the text, with the index indicating that a word should start in the middle of another word. In some cases the entities overlapped with each other, which was also not possible for spaCy to train. Finally, the resume texts had special characters and unremoved whitespaces within the indices, prompting the model to give an error.

To be able to handle all of the cases mentioned above, a data cleaner was required that would be able to handle each of these cases. After cleaning the data, spaCy was able to train on the data, but performed poorly. As the next step, tokenisation was performed on the dataset using the "data_tokeniser", turning long labels into tokens and applying the same label from the long token to each individual token. For example, in resume index three, the following change would be made during tokenisation:

Bachelor of Technology in Computer Science - ***Degree***
to

Bachelor	of	Technology	in	Computer	Science
B-Degree	I-Degree	I-Degree	I-Degree	I-Degree	I-Degree

4.5.3 SpaCy

The spaCy model was setup with the best practices given on the "Entity Recogniser" [15] documentation. This involved first taking all of the IOB data and converting them to spaCy's format of word index to label. This data was then loaded into a spaCy vocabulary using Doc and DocBin ready to be trained on. The training arguments were spaCy predefined base configurations with NER pipeline added, using GPU.

When training on Laxmimerit dataset the first time however, it produced poor results - less than 1 percent. While originally surprising, as the model was set up correctly by using and training on CoNNL 2003 dataset to recognise entities, it became apparent that due to the linguistic features of spaCy [53], the dataset was not optimal for training. It should be mentioned here that the pattern recognition using js_skill_patterns from Microsoft's "SkillsExtractorCognitiveSearch" project [54], also produced results that were far more capable than that of the model.

In the labelling strategies section it was explored that labels can also be presented in text index format. All of the entities in the laxmimerit dataset are not

split into IOB format. In fact, many of the labels have the text representations include multiple entities in a single label. For example in the second resume, there is already a skill represented by the following:

Python (2 years), SQL. (1 year), NOSQL (1 year), R (2 years), Machine Learning (2 years)

This confuses the model making it expect the skill in a format where in order to mark it as a "skill" all of the above have to follow each other. An approach to fix this was taken and the data was tokenised as mentioned in the data processing part above, splitting each of the long labels into token labels of IOB format.

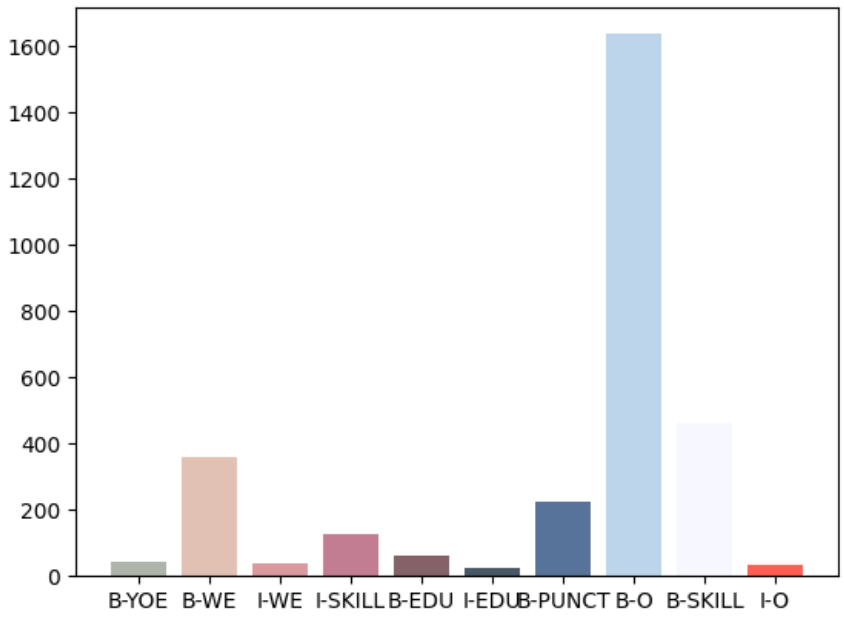
However, when put through the model again, another issue became apparent - due many of the labels being defined differently within the same context, the spaCy model was unable to yield any meaningful results on training. That, combined with potential overlaps in labels made it difficult to get meaningful information from the Laxmimerit dataset.

4.5.4 ChatGPT Text Generation

With the Laxmimerit dataset performing poorly for the task, it became apparent that a better dataset with a correct IOB structure was required. Having explored NER with CoNNL dataset before hand, an attempt was made to have ChatGPT generate short sentences that would include the work experience, education, skills and years of experience labels in an IOB structure.

This approach proved to be efficient. Originally created with GPT 3.5 for 210 sentences with 3009 entities, split into a train/test/validation sets of 60% /20% /20%, the generated dataset contained the following 10 labels: 'B-YOE', 'B-WE', 'I-WE', 'I-SKILL', 'B-EDU', 'I-EDU', 'B-PUNCT', 'B-O', 'B-SKILL', 'I-O'

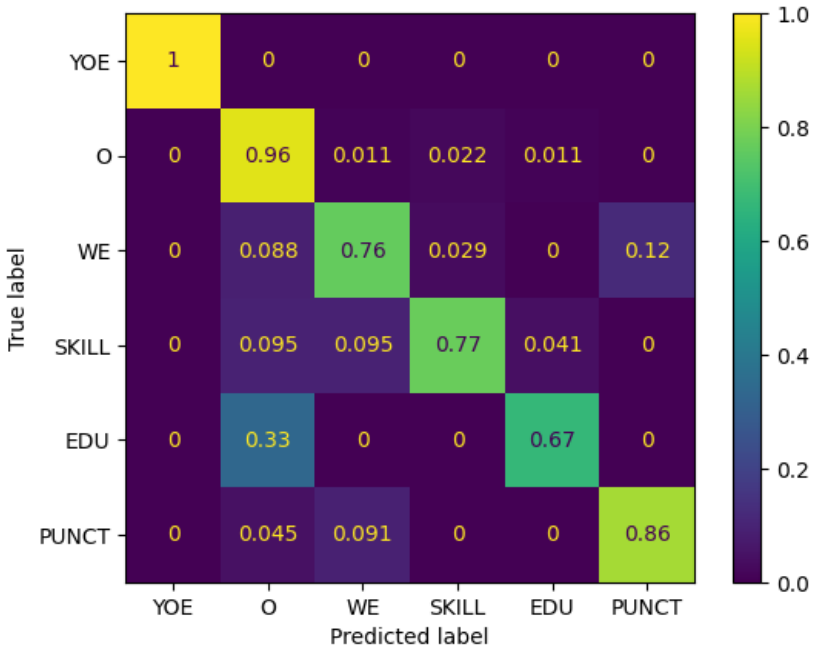
The split of the labels across the data was as following:



GPT 3.5 label balance [38]

While it can be seen that the "B-O" label forms the greatest number of entities, this is to be expected as most tokens will remain outside the searched labels. However, more entities could be defined that could also be marked with the "O" tag to balance the dataset, as well as improve recognition for other words.

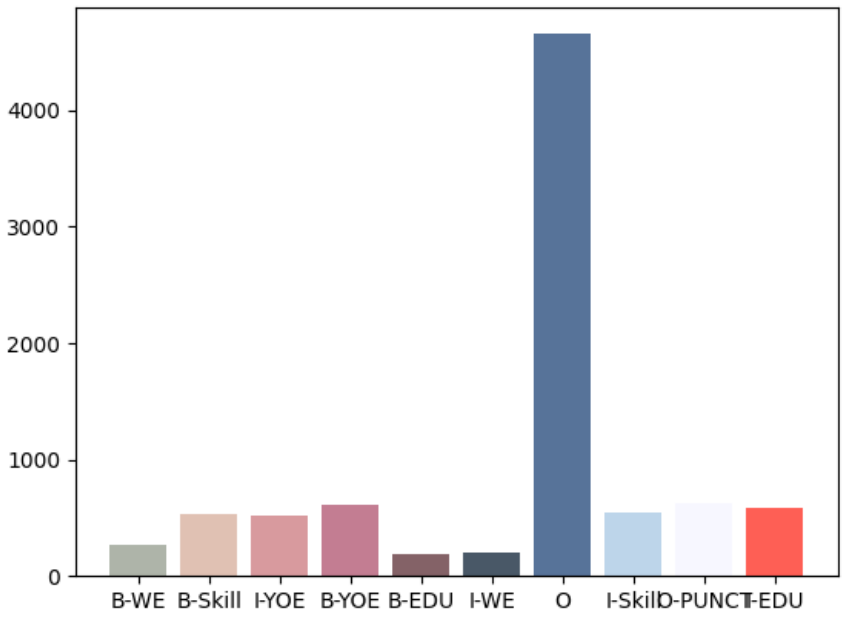
The resulting accuracy can be seen as follows for each label:



GPT 3.5 spaCy NER results [47]

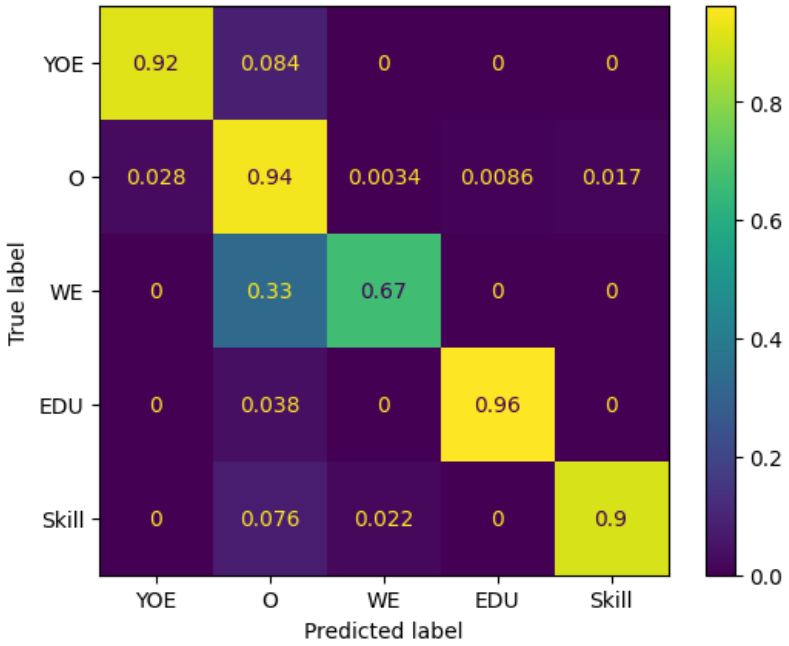
The resulting normalised values indicate that the model can predict each label with a generally good accuracy, with a perfect score for years of experience and lowest score for predicting education. However, when reviewing labels within the dataset, there were some label mismatches generated from the data, as well as not having enough variation within the sentences.

Because of this, a second dataset was built using ChatGPT4, this time for 454 sentences with 8722 entities. This time, the punctuation was also marked as "O" to make sure that it would be an exclusive learning outcome for the model. Otherwise, the labels stayed the same, resulting in the following data:



GPT 4 label balance [39]

While the "O" tag is still the greatest among all the labels, the dataset is generally much more balanced for all the other labels present. This can be also seen within the learning results of the model:



GPT 4 spaCy NER results [48]

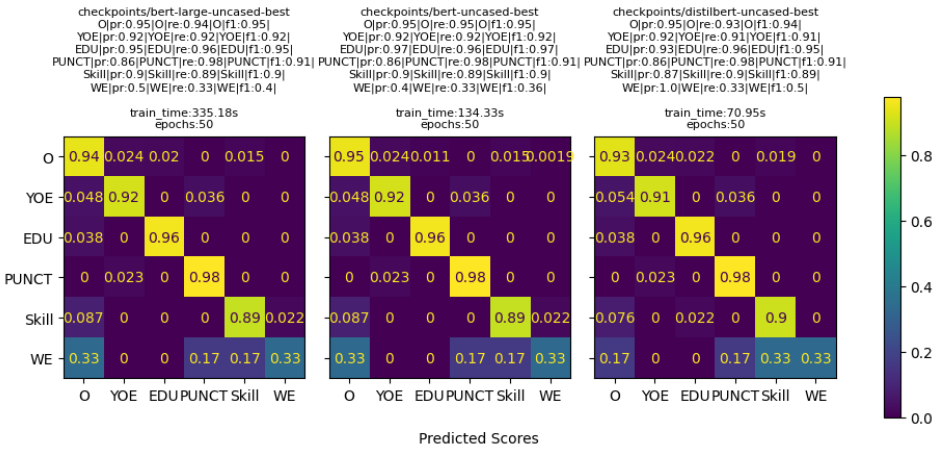
While there was some loss within the work experience tag, all of the other labels were recognised over 90% of the time, making a massive improvement on the original GPT 3.5 dataset and its ability to recognise the labels. The spaCy model for GPT4 data ran for 466.55s or approximately just over 7 minutes. The recall, precision and f1 scores were as follows:

	YOE	O	WE	EDU	Skill
precision	0.904762	0.946459	0.500000	0.975610	0.892473
recall	0.915663	0.943201	0.666667	0.961538	0.902174
f1_score	0.910180	0.944828	0.571429	0.968523	0.897297

GPT 4 spaCy NER precision, recall and f1 scores [36]

4.5.5 BERT based models

In order to compare spaCy's performance to another model, three BERT based models were trained on the ChatGPT 4 dataset to see the differences. The models chosen were BERT-large-uncased, BERT-uncased and DistilBERT-uncased. These models used the hugging faces pretrained models and auto tokenisers to encode the data and then train on it. The results were as following:



GPT 4 BERT Large, BERT and DistilBERT NER results [25]

In most cases, all of the BERT models recognised labels in a similar way after being trained for 50 epochs. The largest differences come in training time and work experience label accuracy. While BERT-large and BERT-uncased trained for longer, they actually achieved poorer results in f1 scores for work experience. The training time for DistilBERT also surpasses the other two models, taking nearly 5 times less time to train than BERT-large.

The results may of course vary with a larger dataset. It is important to note here that in terms of accuracy both spaCy and BERT perform similarly, however, spaCy is more accurate for the work experience label, it also takes much longer to train than the BERT models. As the work experience label is the worst recognised tag among all of the models, it indicates an issue within the dataset which could be fixed by adding more cases of work experience in the future and reviewing how ChatGPT 4 generated them (if there are any issues with mislabelling as there were with ChatGPT 3.5).

4.6 Implementation - Ranking

4.6.1 Data Analysis

With the NER predictions working successfully, a ranking layer could be built on top of it. For this another dataset consisting of job descriptions / user queries and resumes was required. While such a dataset could not be found on public forums, two datasets could be combined for this purpose. The combined dataset forms 853 job descriptions and 32481 resumes with 52 unique topics for the resumes and 725 unique topics for the job descriptions. Out of the 725 job description topics, 425 have matches that can be used to test for ranking accuracy. Out of the 32481 resumes, 1403 were matched.

During the similarity testing part of the ranking, it became clear that a lot of the context within the generated resume dataset, as well as the job descriptions, were not relevant to the topics. This was most likely caused by the stemming approach. Thus, another dataset was generated using ChatGPT 4 to see if the problematic part was within the data or the method. The generated dataset was 100 resumes across 10 topics: arts, automated testing, banking, consultant, electrical engineering, fitness, healthcare, human resources, information technology and sales. Each topic had 10 resumes and 1 job description generated, bringing the dataset to 110 texts.

4.6.2 Data Processing

In order to match the topics together in the aforementioned dataset, a combination of tokenisation and stemming was used. This assisted in finding similar root words within the topics of the resumes. However, when testing the accuracy of the ranking datasets, it became clear that such a large dataset was not required. Within the "data parsing ranking dataset" project, a limit was implemented for both job descriptions and resumes for a specific topic that could be added into the dataset. The testing limits were set to a maximum of 2 job descriptions and a maximum of 10 resumes per selected topic. Out of the 52 topics, 10 were chosen and made sure that they have both at least 1 job description and 1 resume.

4.6.3 Ranking - Sentence Transformers

The sentence transformers use the hugging face sentence transformer library to perform the similarity check. The sentence transformer loaded is distilbert-base-nli-mean-tokens. To perform the similarity check, the transformer first encodes the text and then a cosine similarity equation is performed on the encodings to find distance between them. To rank the scores, they have to be sorted. In case of spaCy labelling, the texts are first labelled and then the transformation is performed.


```
    "selected_pipeline": "sentence_transformer",
    "test_time": "18.91s",
    "accuracy_precent": 17.77777777777778,
    "accuracy_str": "32/180",
```

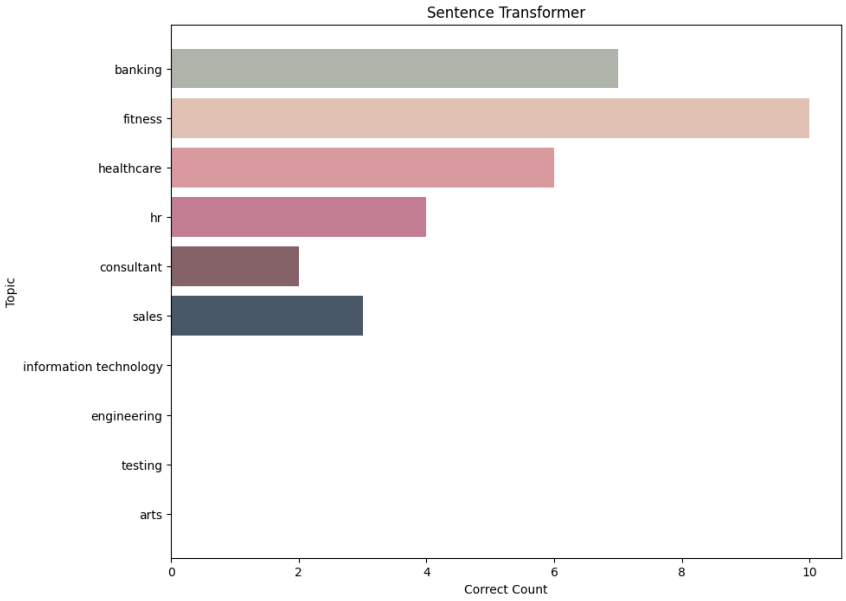

Listing 1: Sentence Transformer ranking result


```
    "selected_pipeline": "spaCy_transformer",
    "test_time": "761.82s",
    "accuracy_precent": 17.22222222222222,
    "accuracy_str": "31/180",
```


Listing 2: SpaCy Sentence Transformer ranking result

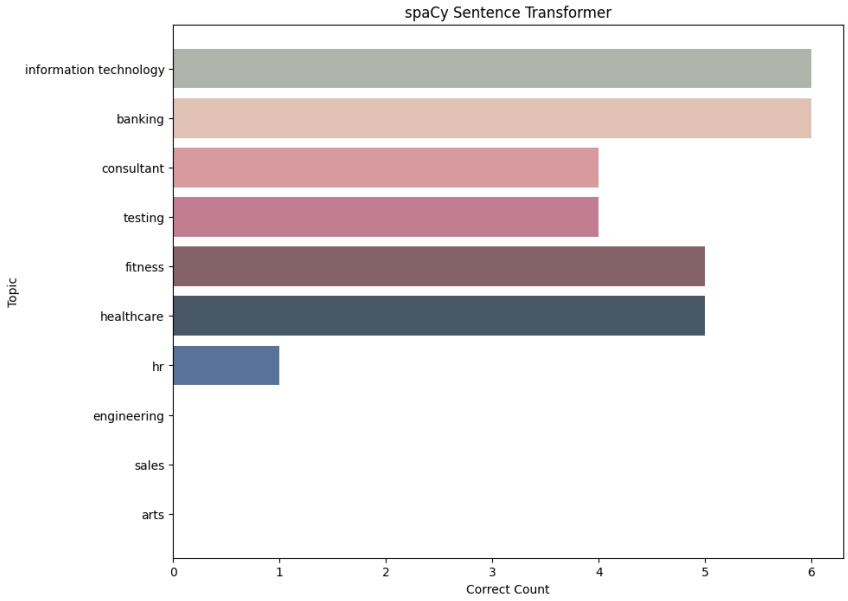
Using the combined dataset of 20 job descriptions and 100 resumes, the sentence transformer managed to predict for both non-labelled and for labelled data with an accuracy of about 17%. The main difference is within time taken - non-labelled data took only 18.91s to process, while labelled data took 761.82s to process. This is down to each text having to be labelled first before it can be similarity checked. In the future this time could be cut down by first pre-labelling the text before running the similarity tests.

For the sentence transformer non-labelled, the most recognised topic is fitness, while least recognised topics are information technology, engineering, testing and arts.



Sentence Transformer ranking results per topic [42]

Using labelled data with spacy alongside sentence transformation, the most recognised topics are information technology and banking, while the least recognised topics are engineering, arts and sales.



SpaCy Sentence Transformer ranking results per topic [45]

However, the scores drastically improved after performing evaluation on the new dataset. In order to test out the speed of labelled data, the dataset was first prelablled and saved as a text. The sentence transformation was then performed on top of it.

```
    "selected_pipeline": "sentence_transformer",
    "test_time": "11.78s",
    "accuracy_precent": 37.0,
    "accuracy_str": "37/100",
```

Listing 3: Sentence Transformer ranking result with GPT4 dataset

```
    "selected_pipeline": "spaCy_transformer",
    "test_time": "14.12s",
    "accuracy_precent": 37.0,
    "accuracy_str": "37/100",
```

Listing 4: SpaCy Sentence Transformer ranking result with GPT4 dataset

A 20% percent increase in accuracy was a great improvement on the previous dataset. It also indicated that the method for ranking text similarity was sound.

4.6.4 Ranking - Sequence Classification

In the sequence classification part of the ranking, the query and the corresponding documents are first ran through the BERT Tokeniser to get the embeddings. The model then predicts outputs and uses softmax to generate scores. As with

sentence transformers, In order to rank the documents, the outputs have to be sorted.

```
    "selected_pipeline": "sequence_classifier",
    "test_time": "92.61s",
    "accuracy_precent": 21.666666666666668,
    "accuracy_str": "39/180",
```

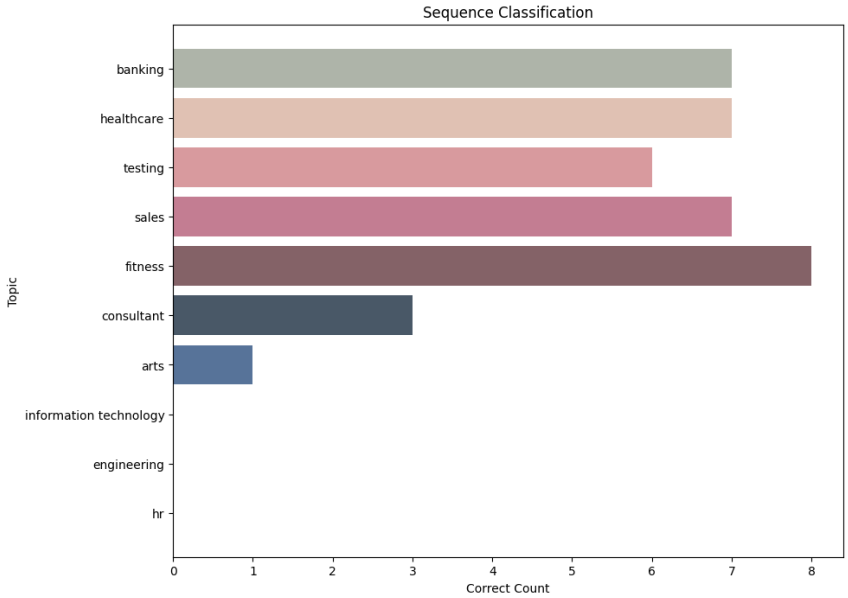
Listing 5: Sequence Classifier ranking result

```
    "selected_pipeline": "spaCy_classifier",
    "test_time": "1328.25s",
    "accuracy_precent": 20.555555555555557,
    "accuracy_str": "37/180",
```

Listing 6: SpaCy Sequence Classifier ranking result

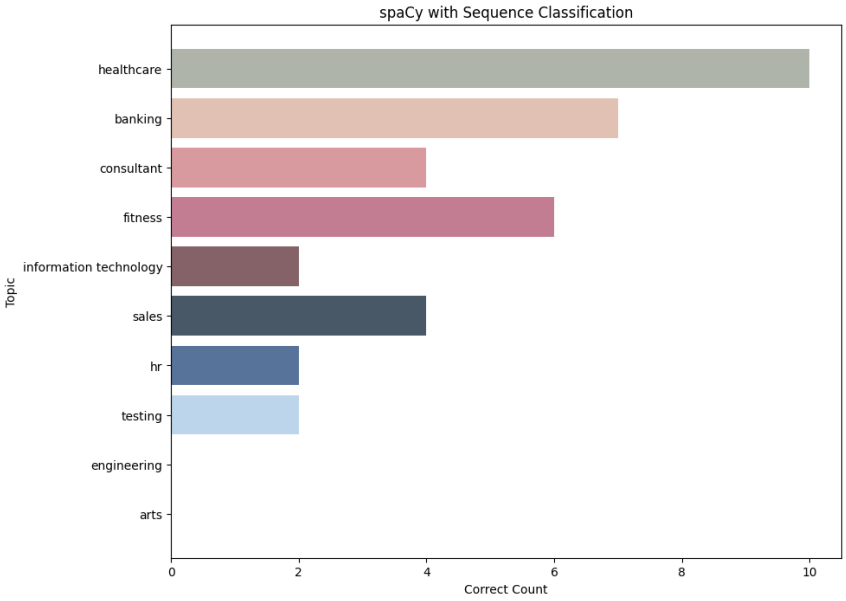
The sequence classification managed to predict for both non-labelled and for labelled data with an accuracy of about 20%. Just as with the sentence transformer, the main difference is once again with the time taken - non-labelled data took only 92.61s to process, while labelled data took 1328.25s to process.

For the sequence classification non-labelled, the most recognised topic is fitness, while least recognised topics are information technology, engineering, testing and arts.



Sequence Classification ranking results per topic [43]

Using labelled data with spacy alongside sequence classification, the most recognised topic is healthcare, while the least recognised topics are engineering and arts.



Spacy Sequence Classification ranking results per topic [44]

Changing the datasets and pre-labelling them first, once again yielded interesting results:

```
    "selected_pipeline": "sequence_classifier",
    "test_time": "61.20s",
    "accuracy_precent": 65.0,
    "accuracy_str": "65/100",
```

Listing 7: Sequence Classifier ranking result with GPT4 dataset

```
    "selected_pipeline": "spaCy_classifier",
    "test_time": "79.53s",
    "accuracy_precent": 66.0,
    "accuracy_str": "66/100",
```

Listing 8: SpaCy Sequence Classifier ranking result with GPT4 dataset

While the non-labelled data still ranked faster, the accuracy had moved up by a near 45% margin. Observing the two methods of classification and transformation, this seemed as a good indication that by changing the datasets to be more relevant to the topic using ChatGPT4 generated data was the correct choice.

4.6.5 Rule Based Approach

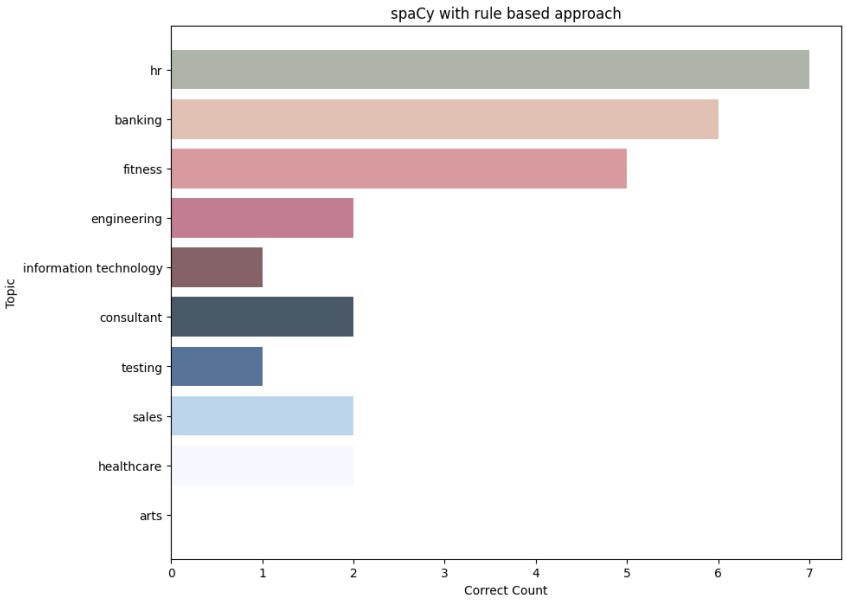
The rule based approach uses spacy to label entities for all of the texts and then stems all of them. It then uses word comparisons to find any potential word

matches and adds score.

```
    "selected_pipeline": "spaCy_rulebased",
    "test_time": "1921.86s",
    "accuracy_precent": 15.555555555555555,
    "accuracy_str": "28/180",
```

Listing 9: SpaCy Rule Based ranking result

The rule based approach took the longest to run, taking up 1921.86s and produced only a 15.5% accuracy. The most recognised topic for the rule based approach was hr, while the least recognised topic was arts. While the rule based approach might have recognised the least amount of resumes correctly, what it has done is recognised at least one resume from most topics, giving it the widest coverage.



Spacy Rule Based ranking results per topic [41]

This result is likely because how the rule based approach is handled - the spaCy labeller finds the correct labels and then each label is compared against another label with their stems. Any stopwords that are within labels are removed, if any are found. If a match is found, a point is added. While rudimentary in its approach, by reducing the amount of important data and finding the matches, the rule based approach still manages to hold close to the datasets.

The most impressive change from changing the dataset however, comes from the rule-based approach:

```
    "selected_pipeline": "spaCy_rulebased",
    "test_time": "315.64s",
    "accuracy_precent": 68.0,
    "accuracy_str": "68/100",
```

Listing 10: SpaCy Rule Based ranking result with GPT4 dataset

Having been the worst at recognising resumes to queries, the rule based approach moved up to 68% accuracy.

4.6.6 Word Vector Approach

The final method for ranking was to implement a word vector similarity check. This would download pre-trained vectors from FastText [14] and use them to perform analysis on the data given. The word vector approach takes the topic of the job description and compares every single word within the resume for similarity. All the similarity scores are added together and then an average similarity is calculated.

```
    'selected_pipeline': 'word_vector',
    'test_time': '314.81s',
    'accuracy_precent': 61.0,
    'accuracy_str': '61/100'
```

Listing 11: Word Vector Based ranking result with GPT4 dataset

The result of the similarity check on the full text was already quite good, reaching a 61% accuracy. However, when using pre-lablled text with the spacy model, the accuracy moved up to 75%. Within this approach, the labelling technique makes a great difference, causing the more important data to increase the models ability to recognise similarity.

```
    'selected_pipeline': 'labelled_word_vector',
    'test_time': '97.51s',
    'accuracy_precent': 75.0,
    'accuracy_str': '75/100'
```

Listing 12: Labelled Word Vector Based ranking result with GPT4 dataset

Not only does the labelled text improve the accuracy, it also decreases the amount of time taken to reach a conclusion

5 Evaluation

5.1 Results

From the GUI, the tasks set out to were reached for both user interactions and visualisations. The use of the GUIs has simplified the interaction between users and the data manipulation, making understanding the project more accessible and intuitive. As shown in the implementation, it offers a visually-oriented way to control data labelling, viewing model accuracy and ranking, eliminating the need for command-line instructions. This visual approach is particularly advantageous when including new models and allows to easily comprehend large volumes of data. By visually breaking down datasets, GUIs allow for quicker pattern recognition and trend analysis, enhancing decision-making processes. Overall, GUIs and data visualisation empowers users to manipulate, analyse, and understand data through a more comprehensible.

For labelling, the most accurate results came from using spaCy over spaCy patterns and BERT. The patterns reached only an average accuracy of 25% on unstructured data, while BERT large reached an average accuracy of 83.6%, BERT uncased an average accuracy of 83.8% and DistilBERT an average accuracy of 83.5%. SpaCy reached an average label accuracy of 87.8%, reaching at least a 4% improvement on the next best model in label recognition.

For ranking, the best method was originally using sequence classifiers on pure texts, reaching a 21.6% accuracy. However, by replacing the dataset and pre-labelling the data, the rule based approach became the most accurate, reaching 68% accuracy on the new dataset. Finally applying the word vector approach reaches the maximum accuracy of 75% with the labelling approach.
While the improvement on the side of rule based approach is impressive, the sentence transformation and classification between full texts and labelled texts does not have a massive difference. However, the accuracy does also not drop between the two methods, indicating that the labelling is still a viable approach. This is further proved by the word vector approach, where adding pre-lablled spacy text as the data increases the accuracy by 14%.

6 Conclusion

6.1 Project Evaluation

With the finalisation of this report, the project has reached a successful conclusion. The following objectives will show how each completion was performed:

Uploading data in a PDF or JSON file format and extracting data into relevant JSON format.
The app is capable of adding new data and constructing it into datasets within the GUI with minimal interactions.

Providing a visual GUI for the users to interact with as to reduce complexity of running commands, as well as providing a reliable system to run tasks in.
The GUI creates the ability to view and edit data, reducing complexity of understanding data, as well as being able to visualise training data to the end user.

The capability to add and remove labels from resumes for training ground truths. The system should also be capable of recognising faulty labels (labels that overlap with other labels or do not start or end with a space).
The data labelling section of the GUI can add and remove labels from the training data, allowing for the user to be able to see errors, as well as fix issues within the dataset.

Find and label entities within texts that will be used for ranking.
The data labelling section is also capable of removing and adding new entities during the label editing process.

Rank resumes from user input or job description.
The GUI is capable of taking user input and ranking the resumes to the search query, providing a visual representation at the end.

Provide accuracy metric for finding labels from ground truths to see how well a model is performing.
The labelling accuracy can be viewed within the GUI to visualise how well ground truth labels are recognised.

Given a train/test/validation split of data, a model capable of training named entities and producing metrics of how well the model has learned.
Within the models section, both the spaCy and the BERT models are capable of recognising named entities from a set of training data. Further to this, there are multiple data parsing functionalities to help with processing and adjusting data for the training.

Given a text in string format, the model is capable of recognising entities and displaying them.

Using either the GUI or spaCy's displacy, the entities trained from the model can be visualised for the end user to understand how the data is labelled.

Given a job description or a search query, the model is capable of finding best resulting resumes.
There are various methods implemented within the models section of the project that allow for ranking of texts based on similarity and evaluating through topic, including sequence classification, sentence transformation, labelling approaches and rule based ranking.

6.2 Future Work

Given more time with the project and extending the scope, there are a multitude of additional features and improvements to be explored. As it currently stands, while the GUI is capable of running each of the outlined tasks assigned to it, not all of the NER models (BERT) and ranking models (labelled texts) are included in the GUI. It would be beneficial for an end user to have all of these be implemented in the GUI, rather than keeping them separate. A simple UDP client to interact with the GUI without having to run frontend would also be beneficial, as it would speed up development time.

For ranking, many improvements could still be explored. This project only looked at ranking the texts within their similarity, but for aligning the evaluation datasets for ranking, used topic matching. Because it is proven that the resumes can be matched on the topic, it would be beneficial to also explore if topics can be robustly recognised from texts and establish if this could be used as a ranking metric.

Similarity comparison could also be improved by using more relevant data. An optimal way to construct texts for similarity comparisons would be to test datasets first with word vector similarities to make sure that the texts score highly for the topic that they are being tested for. As this project used rigorous topic matching, perhaps more leniency should be introduced. For instance, while topics "electrical engineering" and "information technology" might fall into different topics, they have similarity within both texts and might thus have to be included as a potential match. Because the recruiting process is complicated even for a person, sometimes having a person from a different area may work better due to skill and similarity matching to the job.

Another improvement would be to introduce a human verified IOB dataset for training. As currently most of the data is generated by ChatGPT, this does not have oversight if the data is actually correct in each instance of the tags. A human verified dataset could ensure robustness within the datasets, as well as mitigate any learning issues.

6.3 Final Statement

In conclusion, this project successfully implemented a graphical user interface (GUI) that enhances data visualisation, particularly for error handling and for key entities such as work experience, education, years of experience, and skills within resumes. The GUI not only enriched the user experience but also enabled more efficient data interaction.

Additionally, the project successfully developed a named entity recognition model that proficiently extracts essential information from various unstructured resumes. Moreover, the project explored advanced methodologies for resume ranking, utilising sequence classification and sentence transformation techniques on both full and labelled texts.

Despite these methods, the accuracy of the resume ranking models remained modest at first, averaging only at approximately 20%, with labelled texts providing no significant enhancement in accuracy. However, by switching the datasets, as well as pre-labelling the texts, the results improved in somecases by 40%, with maximum accuracy reached being 75%.

These results underscore the complexities of automating intricate text analysis tasks and signify the necessity for continued research and refinement of models to elevate their effectiveness in practical applications. The project thus establishes a robust foundation for future exploration and potential enhancements in automated resume analysis and ranking systems.

6.4 Statement of Ethics

6.4.1 Data Privacy and Confidentiality

Given the nature of this report, resume data will often hold sensitive personal information such as names, phone numbers and email addresses. It is the duty of this project to ensure this data is anonymised to protect the individuals privacy with personal identifiers removed. Early on, the project did use the Laxmimerit dataset which had names within the dataset. However, as the project progressed, this dataset was removed due to training issues and replaced with AI generated data. The resumes used for ranking were also AI generated and the job description dataset did not contain individuals personal information.

As this project does not hold nor distributes any of the data trained with, there is no need for implementing security on the data. However, were the circumstances to change and sensitive data were to be held, a strong data security measure must be implemented to prevent unauthorised access.

6.4.2 Bias and Fairness

Machine learning models often include biases that can perpetuate and amplify existing biases. The aim of this project was to be as unbiased as possible and only perform ranking solely on the similarity of words and skills. The data was AI generated and meant to be generic, not specifying race, gender or affiliations.

6.4.3 Consent and Informed Participation

As mentioned before, this project utilises AI generated data in both training and evaluating the accuracy of the methods. The data was generated using ChatGPT for IOB tagged sentence to train named entity recognition and to generate resumes for fake applicants.

The open source data used from public repositories has been verified not to have any licences for use nor individual conformations of use were mentioned.

6.4.4 Compliance with Legal and Ethical Standards

This project is in compliance with any relevant laws and regulations, such as GDPR and does not seek to expose or use any personal information. The ethics within the project is verified by the Self-Assessment for Governance and Ethics (SAGE) form to ensure compliance.

6.4.5 British Computer Society Code of Conduct

Further to that, the project lines with the British Computer Society (BCS) code of conduct, as is expected from all of their members. The research performed within this project was done so without discrimination and took steps to remain

unbiased.

The work performed was done so within professional competence of the project member's abilities and it was ensured that the member continued to develop their skills in understanding machine learning algorithms in text similarity ranking and named entity recognition.

The project remains within the legislation and upholds the BCS standards.

6.4.6 Social Responsibility

The emphasis of this project was to point out that the current model for performing recruiting with the number of applicants each year is becoming unsustainable. While this project does understand the dangers of automating a role that is currently managed by humans, it is the belief of this report that the models built here should only be used to assist and unburden the human individuals, not to replace them.

It is also important to understand that this project does not seek to gain any commercial or monetary gain, but only to explore and understand the concepts of how to choose a best possible candidate for a relevant position.

References

[1] *4 Best frameworks for cross-platform desktop app development - Scythe Studio Blog*. https://scythe-studio.com/en/blog/4-best-frameworks-for-cross-platform-desktop-app-development. (Accessed on 05/10/2024).

[2] *5974d1c986a52df62d99aa79530e3575.pdf*. https://d197for5662m48.cloudfront.net/documents/publicationstatus/133262/preprint_pdf/5974d1c986a52df62d99aa79530e3575.pdf. (Accessed on 05/09/2024).

[3] *7. Extracting Information from Text*. https://www.nltk.org/book/ch07.html. (Accessed on 05/11/2024).

[4] *A Comparison between Named Entity Recognition Models in the Biomedical Domain*. https://aclanthology.org/2021.triton-1.9.pdf. (Accessed on 05/11/2024).

[5] *AJAX Introduction*. https://www.w3schools.com/xml/ajax_intro.asp. (Accessed on 05/12/2024).

[6] Wael Albassam. *The Power of Artificial Intelligence in Recruitment: An Analytical Review of Current AI-Based Recruitment Strategies*. June 2023. DOI: 10.26668/businessreview/2023.v8i6.2089.

[7] *AN AUTOMATED RESUME SCREENING SYSTEM USING NATURAL LANGUAGE PROCESSING AND SIMILARITY*. https://www.intelcomp-design.com/paper/2etit2020/2etit2020-99-103.pdf. (Accessed on 05/12/2024).

[8] *Application of Entity-BERT model based on neuroscience and brain-like cognition in electronic medical record entity recognition - PMC*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10547885/. (Accessed on 05/11/2024).

[9] *BERT*. https://huggingface.co/docs/transformers/en/model_doc/bert. (Accessed on 05/11/2024).

[10] *BERT-ER — Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. https://dl.acm.org/doi/abs/10.1145/3477495.3531944. (Accessed on 05/12/2024).

[11] *Build cross-platform desktop apps with JavaScript, HTML, and CSS — Electron*. https://www.electronjs.org/. (Accessed on 05/10/2024).

[12] *Collaboration among recruiters and artificial intelligence: removing human prejudices in employment - PMC*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9516509/. (Accessed on 05/09/2024).

[13] *Design and implementation of an open source Greek POS Tagger and Entity Recognizer using spaCy*. https://dl.acm.org/doi/pdf/10.1145/3350546.3352543. (Accessed on 05/11/2024).

[14] *English word vectors · fastText*. https://fasttext.cc/docs/en/english-vectors.html. (Accessed on 05/15/2024).

[15] *EntityRecognizer · spaCy API Documentation*. https://spacy.io/api/entityrecognizer. (Accessed on 05/11/2024).

[16] *fin_irjmets1684836041.pdf*. https://www.irjmets.com/uploadedfiles/paper/issue_5_may_2023/39611/final/fin_irjmets1684836041.pdf. (Accessed on 05/12/2024).

[17] *Getting Started — Axios Docs*. https://axios-http.com/docs/intro. (Accessed on 05/12/2024).

[18] *GloVe: Global Vectors for Word Representation*. https://nlp.stanford.edu/projects/glove/. (Accessed on 05/12/2024).

[19] *How many applications to get a job? [2024 Data & analysis]*. https://standout-cv.com/how-many-applications-to-get-a-job. (Accessed on 05/10/2024).

[20] *Hugging Face Hub documentation*. https://huggingface.co/docs/hub/index. (Accessed on 05/11/2024).

[21] *InferencePrince555/Resume-Dataset · Datasets at Hugging Face*. https://huggingface.co/datasets/InferencePrince555/Resume-Dataset. (Accessed on 05/13/2024).

[22] *Introduction — Pinia*. https://pinia.vuejs.org/introduction.html. (Accessed on 05/12/2024).

[23] *Introduction to FastText Embeddings and its Implication -*. https://www.analyticsvidhya.com/blog/2023/01/introduction-to-fasttext-embeddings-and-its-implication/. (Accessed on 05/12/2024).

[24] Oliver Krieger. Example of adding a label. (Accessed on 05/12/2024).

[25] Oliver Krieger. *BERT results on label training for GPT 4 data*. (Accessed on 05/12/2024).

[26] Oliver Krieger. *Data Labeller before a dataset is selected*. (Accessed on 05/12/2024).

[27] Oliver Krieger. *Example of Data Labeller*. (Accessed on 05/12/2024).

[28] Oliver Krieger. *Example of data ranking results*. (Accessed on 05/12/2024).

[29] Oliver Krieger. *Example of error alert*. (Accessed on 05/12/2024).

[30] Oliver Krieger. *Example of error labels resolved*. (Accessed on 05/12/2024).

[31] Oliver Krieger. *Example of no-state and state with Dataset and Model selected*. (Accessed on 05/12/2024).

[32] Oliver Krieger. *Example of numerical accuracy*. (Accessed on 05/12/2024).

[33] Oliver Krieger. *Example of the Data Training options*. (Accessed on 05/12/2024).

[34] Oliver Krieger. *Example of the loader component*. (Accessed on 05/12/2024).

[35] Oliver Krieger. *Example of the Model Accuracy tab for NER*. (Accessed on 05/12/2024).

[36] Oliver Krieger. *F1 Scores for GPT 4 data training for labelling using SpaCy*. (Accessed on 05/12/2024).

[37] Oliver Krieger. *Front Page Displaying Content*. (Accessed on 05/12/2024).

[38] Oliver Krieger. *GPT 3.5 label data balance visualised*. (Accessed on 05/12/2024).

[39] Oliver Krieger. *GPT 4 label data balance visualised*. (Accessed on 05/12/2024).

[40] Oliver Krieger. *GUI Ranking Diagram*. (Accessed on 05/12/2024).

[41] Oliver Krieger. *Ranking results for labelling with rule based approach*. (Accessed on 05/12/2024).

[42] Oliver Krieger. *Ranking results for sentence transformer*. (Accessed on 05/12/2024).

[43] Oliver Krieger. *Ranking results for sequence classifier*. (Accessed on 05/12/2024).

[44] Oliver Krieger. *Ranking results for spaCy sequence classifier*. (Accessed on 05/12/2024).

[45] Oliver Krieger. *Ranking results for spaCy setence transformer*. (Accessed on 05/12/2024).

[46] Oliver Krieger. *Rule Based Scoring Diagram*. (Accessed on 05/12/2024).

[47] Oliver Krieger. *SpaCy results on label training for GPT 3.5 data*. (Accessed on 05/12/2024).

[48] Oliver Krieger. *SpaCy results on label training for GPT 4 data*. (Accessed on 05/12/2024).

[49] Oliver Krieger. *Vectorised Words Plot*. (Accessed on 05/12/2024).

[50] Oliver Krieger. *Visual Diagram of GUI System Architecture*. (Accessed on 05/12/2024).

[51] *Label Ranking: A New Rule-Based Label Ranking Method — SpringerLink*. `https://link.springer.com/chapter/10.1007/978-3-642-31709-5_62`. (Accessed on 05/11/2024).

[52] *laxmimerit/CV-Parsing-using-Spacy-3: Resume and CV Summarization and Paring with Spacy in Python*. `https://github.com/laxmimerit/CV-Parsing-using-Spacy-3/tree/master`. (Accessed on 05/13/2024).

[53] *Linguistic Features · spaCy Usage Documentation*. `https://spacy.io/usage/linguistic-features`. (Accessed on 05/14/2024).

[54] *microsoft/SkillsExtractorCognitiveSearch*. `https://github.com/microsoft/SkillsExtractorCognitiveSearch`. (Accessed on 05/14/2024).

[55] *NLP — IOB tags - GeeksforGeeks*. `https://www.geeksforgeeks.org/nlp-iob-tags/`. (Accessed on 05/11/2024).

[56] *Pointwise vs. Pairwise vs. Listwise Learning to Rank — by Nikhil Dandekar — Medium*. `https://medium.com/@nikhilbd/pointwise-vs-pairwise-vs-listwise-learning-to-rank-80a8fe8fadfd`. (Accessed on 05/12/2024).

[57] *Resume Dataset*. https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset. (Accessed on 05/13/2024).

[58] *Resume Parsing using spaCy. What is Resume Parsing ? — by Vikrant Patil — Medium*. https://medium.com/@vikrantptl06/resume-parsing-using-spacy-af24376ec008. (Accessed on 05/12/2024).

[59] *Resume-Job-Description-Matching/data.csv at master · binoydutt/Resume-Job-Description-Matching*. https://github.com/binoydutt/Resume-Job-Description-Matching/blob/master/data.csv. (Accessed on 05/13/2024).

[60] *SentenceTransformers Documentation — Sentence-Transformers documentation*. https://www.sbert.net/. (Accessed on 05/12/2024).

[61] *Sentiment Analysis of App Reviews: A Comparison of BERT, spaCy, TextBlob, and NLTK — by Francis Gichere — Medium*. https://francisgichere.medium.com/sentiment-analysis-of-app-reviews-a-comparison-of-bert-spacy-textblob-and-nltk-9016054d54dc. (Accessed on 05/11/2024).

[62] *spaCy meets Transformers: Fine-tune BERT, XLNet and GPT-2 · Explosion*. https://explosion.ai/blog/spacy-transformers. (Accessed on 05/11/2024).

[63] *Text Classification with BERT. In this tutorial, we will use BERT to... — by Khang Pham — Medium*. https://medium.com/@khang.pham.exxact/text-classification-with-bert-7afaacc5e49b. (Accessed on 05/12/2024).

[64] *Topic Analysis: A Complete Guide*. https://monkeylearn.com/topic-analysis/. (Accessed on 05/09/2024).

[65] *TypeScript: Documentation - Generics*. https://www.typescriptlang.org/docs/handbook/2/generics.html. (Accessed on 05/12/2024).

[66] *Using Sentence Transformers at Hugging Face*. https://huggingface.co/docs/hub/en/sentence-transformers. (Accessed on 05/12/2024).

[67] *Vue.js - The Progressive JavaScript Framework — Vue.js*. https://vuejs.org/. (Accessed on 05/10/2024).

[68] *What are Transformers? - Transformers in Artificial Intelligence Explained - AWS*. https://aws.amazon.com/what-is/transformers-in-artificial-intelligence/. (Accessed on 05/12/2024).

[69] *What Is a Pretrained AI Model? — NVIDIA Blog*. https://blogs.nvidia.com/blog/what-is-a-pretrained-ai-model/. (Accessed on 05/11/2024).

[70] *What is Clustering? — Machine Learning — Google for Developers*. https://developers.google.com/machine-learning/clustering/overview. (Accessed on 05/12/2024).

[71] *What Is Named Entity Recognition (NER) and How It Works?* https://www.altexsoft.com/blog/named-entity-recognition/. (Accessed on 05/11/2024).

[72] *What is Windows Presentation Foundation - WPF .NET — Microsoft Learn*. https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0. (Accessed on 05/10/2024).

[73] *Word representations · fastText*. https://fasttext.cc/docs/en/unsupervised-tutorial.html. (Accessed on 05/12/2024).

[74] *Word2vec - Wikipedia*. https://en.wikipedia.org/wiki/Word2vec. (Accessed on 05/12/2024).

[75] *Word2Vec Model Analysis for Semantic Similarities in English Words - ScienceDirect*. https://www.sciencedirect.com/science/article/pii/S1877050919310713. (Accessed on 05/12/2024).

[76] *World Population by Year - Worldometer*. https://www.worldometers.info/world-population/world-population-by-year/. (Accessed on 05/09/2024).

Appendix

Project Timeline

MONTH	TASKS
Ocrober	1. Research into different resume ranking methods (Literature Review) 2. Keyword, grammar and statistical parsing, topic classification and modelling research 3. Information retrieval from resumes 4. Topic modelling achieved using word frequency
November	1. Understanding NER methods for text labelling and learning about text similarity for ranking 2. Datasets found 3. Research on spaCy, how it works and text analysis, as well as first pass on using spacy to find labels
December	1. Building first pass on system for front end GUI and backend server (websockets) 2. Text parsing into string and resumes 3. Error handling and visualisation for labels, as spacy model would not train otherwise
January	1. Better label visualisations and ability to add and remove labels 2. Re-work frontend app to work better / be more consistent 3. Model training tabs and options

Feburary	1. BERT model for document ranking research 2. Accuracy visualisation for label finding using NER 3. NER with patterns 4. Draft report
March	1. NER with SpaCy trained 2. Dataset Parsing 3. Ranking Strategies 4. Creating dataset from job descriptions and resumes using topics
April	1. Data Analysis 2. Change from open source datasets to GPT generated data for labelling due to poor label performance 3. IOB structure implemented on data
May	1. Improve on similarity checks 2. Ranking data generated via GPT 3. NER with DistilBERT 4. Main report