



Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science in Engineering - 30.0 credits
Software Engineering
Course Code: DVA501

A COMPARATIVE ANALYSIS OF NLP ALGORITHMS FOR IMPLEMENTING AI CONVERSATIONAL ASSISTANTS

Aanchal Upreti
aui22001@student.mdu.se

Examiner: Antonio Cicchetti
Mälardalen University, Västerås

Supervisor(s): Wasif Afzal
Mälardalen University, Västerås

Company Supervisor(s): Mayank Darbari
Nokia Corporation, Helsinki

10th October 2023

Abstract

The rapid adoption of low-code/no-code software systems has reshaped the landscape of software development, but it also brings challenges in usability and accessibility, particularly for those unfamiliar with the specific components and templates of these platforms. This thesis targets improving the developer experience in Nokia Corporation’s low-code/no-code software system for network management through the incorporation of Natural Language Interfaces (NLIs) using Natural Language Processing (NLP) algorithms.

Focused on key NLP tasks like entity extraction and intent classification, we analyzed a variety of algorithms, including MaxEnt Classifier with NLTK, Spacy, Conditional Random Fields with Stanford NER for entity recognition, and SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET for intent classification. Each algorithm’s performance was rigorously evaluated using a dataset generated from network-related utterances. The evaluation metrics included not only performance metrics but also system metrics.

Our research uncovers significant trade-offs in algorithmic selection, elucidating the balance between computational cost and predictive accuracy. It reveals that while some models, like RASA DIET, excel in accuracy, they require extensive computational resources, making them less suitable for lightweight systems. In contrast, simpler models like Spacy and StanfordNER provide a balanced performance but require careful consideration for specific entity types.

While the study is limited by dataset size and focuses on simpler algorithms, it offers an empirically-grounded framework for practitioners and decision-makers at Nokia and similar corporations. The findings point towards future research directions, including the exploration of ensemble methods, the fine-tuning of existing models, and the real-world implementation and scalability of these algorithms in low-code/no-code platforms.

Contents

1. Introduction:	1
2. Problem Formulation	1
3. Background	3
3.1. Natural Language Processing (NLP):	3
3.2. Entity extraction:	4
3.3. Need for custom Entity Extraction:	6
3.4. Intent classification:	6
3.5. Word Embeddings:	8
3.6. Tools and Libraries for Entity Extraction and Intent Classification:	9
4. Related Works	10
5. Research Methodology	11
6. Approach:	13
6.1. Data:	13
6.1.1 Data Composition:	14
6.1.2 Data Synthesis:	16
6.1.3 Data Processing:	17
6.1.3.1 Conversion of data into IOB format:	17
6.1.3.2 Conversion of data into Spacy format:	19
6.1.3.3 Conversion of raw data into format suitable for Intent Classification:	21
6.1.4 Data Annotations:	21
6.1.4.1 Custom Entity Labeling:	21
6.1.4.2 Custom Intent Labeling:	22
6.2. Model Architecture:	23
6.2.1 NLTK with MaxEnt Classifier	23
6.2.1.1 Natural Language Toolkit (NLTK):	23
6.2.1.2 Maximum Entropy (MaxEnt) classifier:	24
6.2.2 Stanford NER with Conditional Random Fields for Entity Extraction:	25
6.2.2.1 Stanford’s Named Entity Recognition (NER):	25
6.2.2.2 CRF Model	26
6.2.3 spaCy for Named Entity Recognition:	26
6.2.4 Rasa DIET for Entity Recognition and Intent Classification:	27
6.2.5 Support Vector Machine (SVM) for Intent Classification:	29
6.2.6 Naive Bayes for Intent Classification:	30
6.2.7 Logistic Regression for Intent Classification:	31
6.2.8 Decision Tree for Intent Classification:	32
6.2.9 Random Forest for Intent Classification:	32
6.3. Experimental setup:	33
6.3.1 Experiment 1: Entity Recognition:	33
6.3.2 Experiment 2: Intent Classification:	35
6.4. Evaluation:	36
6.4.1 Performance Metrics:	36
6.4.2 System Metrics:	37
6.5. Results:	37
6.5.1 Overall Comparison of Performance Metrics:	37
6.5.2 Entity-wise Comparison of Entity Recognition Models:	38
6.5.3 Intent-wise Comparison of Intent Classification Models:	40
6.5.4 Comparison of models based on system metrics:	42
7. Discussions:	43

8. Limitations:	45
9. Conclusion and Future Work	45
10. Acknowledgement:	47
References	48

1. Introduction:

The use of low code/no code software systems is growing rapidly, as they allow developers to create software applications more quickly and with less technical expertise[1]. However, these systems can still be challenging to use, particularly for users who may not be familiar with the specific components and templates provided by the system[2]. One promising approach to addressing this challenge is the use of natural language interfaces (NLIs)[3], which allow users to interact with software tools and systems using written language, similar to conversational assistants or chatbots.

NLIs can be particularly useful for tasks that may be difficult to express using traditional interfaces, or for developers who may not be familiar with the underlying technologies being used[4]. By providing a more intuitive and efficient way for users to interact with the system, NLIs offers a number of potential benefits, including improved usability and accessibility of software development tools and resources, reduced learning curves for new developers, and the ability to provide real-time assistance as developers work on coding tasks[4].

Natural language interfaces, which enable the use of natural language (e.g., English) to interact with computer systems are often implemented using natural language processing (NLP)[5] techniques, which allow computers to understand, interpret, and generate human language. While numerous NLP approaches exist, including rule-based[6], statistical[6], machine learning[7], deep learning[7], and hybrid methods[6], challenges remain in developing NLP algorithms for natural language interfaces in low-code/no-code software systems. One challenge is the need to handle a wide range of user input, including variations in language, dialect, and syntax. This requires NLP algorithms with a high degree of flexibility and adaptability. Another challenge is the need to accurately interpret and respond to user intent, particularly in the context of complex or ambiguous input. This requires NLP algorithms with a high level of semantic understanding and the ability to disambiguate meaning. A third challenge is the need to integrate the natural language interface with the underlying software system in a seamless and intuitive manner. This requires careful design and testing to ensure that the interface is intuitive and easy to use for developers. Since, most of the NLP algorithms can be complex and may require a large dataset of annotated examples for machine training, it is very important to identify the algorithm that is able to effectively and efficiently handle the complexity of the domain, the ambiguity of natural language, and the integration with the existing visual interface, while also addressing the challenges of limited training data.

2. Problem Formulation

The focus of this thesis is inspired by Nokia's specific use case. Nokia has developed a low-code/no-code software system for network management. Its primary users are developers with programming experience and network operation knowledge. Recognizing the potential of making their low-code/no-code system even more user-centric, Nokia Corporation has shown interest in enhancing its platform with natural language interfaces (NLIs). This motivation stems from the challenges faced by users in interacting with the current system and the belief that NLIs can significantly enhance the user experience. The nuances of human language, combined with the adaptability and power of NLIs, could potentially revolutionize how developers interact with Nokia's system. By incorporating NLI, users gain the ability to interact with the system using natural language text. For instance, instead of navigating through a plethora of menus and options, a user wanting to limit bandwidth to 500 Mbps in a specific area can directly input the command in natural language, such as "limit the bandwidth to 500 Mbps in the campus." The system then processes the user input and outputs the confirmation that the bandwidth limit has been set for the campus.

In order to process such user input, NLI is backed with natural language processing techniques[8], encompassing various subtasks. These algorithms are responsible for interpreting and processing user input and generating appropriate responses. In our research, we focus on subtasks such as entity extraction[9] and intent classification[10]. Entity extraction entails identifying and categorizing entities like people, organizations, locations, and specific objects in text. Intent classification involves discerning the intended meaning or desired actions.

Nokia has experimented with the incorporation of NLI into their current platform, utilizing

RASA's DIET[11] model for NLP sub-tasks of entity extraction and intent classification. This model, being deep learning-based, demands substantial datasets for effective training. However, the developed low-code/no-code system is very simple and lightweight, meaning that implementing NLP algorithms, which are mostly complex and rely on large datasets, poses integration and future maintenance challenges.

Hence, Nokia's interest lies in incorporating NLI on their low-code/no-code platform, while simultaneously ensuring that the system remains lightweight, agile, and maintainable. To integrate NLI effectively, it's imperative to delve into the underlying NLP algorithms that power them. Thus, through this work, Nokia is keen on investigating alternative NLP algorithms that not only interpret queries accurately but are also optimized for training data, computational efficiency, system integration, and long-term maintainability. Their strategy is to assess a diverse set of NLP algorithms based on key performance metrics, including precision, recall, and f1 score, as well as system metrics like memory usage, processing time, and resource consumption. Thus, in our work, we begin by identifying potential NLP algorithms that align with the general need for effective natural language processing. Generally, there's a universally acknowledged requisite in the field of NLP: algorithms must excel in processing and interpreting natural language with accuracy. Independent of Nokia's particular scenario, any NLP solution's worth is gauged by its efficacy. Hence, performance metrics like precision, recall, and the f1 score stand out as universal indicators, assessing an algorithm's proficiency in its core function. Following this, we narrow down choices based on the specific needs of Nokia's platform, gauged through system metrics evaluation. Specific to Nokia, the nature of their system inherently demands freedom from the computational intricacies often tied to deep learning models. Being a low-code/no-code platform, which has limited capability to manage extensive datasets or high resource demands, it's imperative for it to stay lightweight and agile. This characteristic drives the need for algorithms that are not only computationally economical but also independent from massive datasets. Thus, the focus direction of this work leans towards investigating simple algorithms and omitting the emphasis on resource-intensive deep learning models. System metrics, encompassing memory usage, processing time, and resource consumption, are pivotal in this context as they are the tangible indicators that mirror Nokia's desire to uphold the platform's agility and simplicity. Similarly, the evaluation based on such metrics becomes particularly crucial when model selection decisions cannot be drawn based purely on conventional performance metrics.

Thus, the objective of this research is to explore and perform a comparative analysis to identify the effective and efficient NLP algorithms and techniques that can be implemented on natural language interfaces that can potentially enhance the developer experience when integrated on low-code/no-code software systems developed by Nokia. It's important to note that this work does not encompass the integration of these algorithms into Nokia's system but instead offers a blueprint for the types of models available and their evaluation based on performance and system metrics.

Aligned with this goal, the thesis addresses the following questions:

- RQ1: In the broader landscape of AI conversational assistants, which NLP algorithms, especially concerning entity extraction and intent classification, are prominently discussed?
- RQ2: Upon evaluation, how effective and efficient are these NLP models selected from the literature review in terms of system metrics and performance metrics??

To reiterate, this study's core purpose isn't to single out the absolute best model, but to shed light on various algorithms' capabilities and limitations, providing Nokia with comprehensive insights. This research provides Nokia with insights into the available models for specific tasks and how these models perform in terms of efficiency and effectiveness. The study will offer an in-depth analysis of the current state-of-the-art NLP algorithms and their suitability for this application. The focus of this thesis, while inspired by Nokia's specific use case, has broader implications. While Nokia's interest in enhancing the developer experience by integrating NLI is the starting point, the findings of this research aim to be generalizable. Given that the majority of low-code/no-code platforms lean towards being lightweight, the findings from this research will be instrumental in guiding the future inclusion of NLI in similar systems. By pinpointing NLP algorithms that respect the data and resource constraints of such platforms, the research not only addresses Nokia's immediate challenges but also lays a robust foundation for future studies in this domain.

3. Background

3.1. Natural Language Processing (NLP):

Languages that have naturally developed over time and are employed by humans, such as Finnish, English, Nepali, and French, are known as natural languages. The scientific examination of these languages from a computational standpoint is referred to as Natural Language Processing (NLP), a discipline primarily focused on the interaction between computers and human languages. Additionally, NLP serves as an interdisciplinary domain that integrates linguistics, cognitive science, computer science, and Artificial Intelligence (AI). Natural Language Processing (NLP) algorithms[8] are a key component of natural language interfaces, as they are responsible for interpreting and processing user input and generating appropriate responses. As depicted in Figure 1, domain experts can craft their own natural language interfaces using NLP techniques, tailored to their respective fields, leveraging specific knowledge to effectively utilize retrieved information.

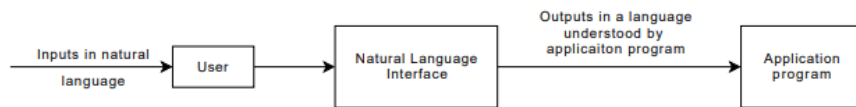


Figure 1: Example of NLP system

Natural Language Processing (NLP) techniques are widely used in many applications and encompasses several key tasks, including:

1. Automatic summarization: Creating concise and coherent summaries from source documents, presenting information in an easily comprehensible manner.
2. Information extraction: Deriving information from unstructured text sources, such as identifying topics within a document.
3. Information retrieval: Locating relevant text documents from large collections to retrieve specific information.
4. Named entity recognition: Identifying named entities like individuals, locations, and time references within text.
5. Natural language generation: A software process involved in machine reading that generates human-like textual output, as seen in chatbots generating responses.
6. Optical character recognition: Automatically extracting data from printed or handwritten text, commonly in scanned documents or image files.
7. Machine translation: The automated translation of text from one language to another without human intervention.
8. Question answering: Automatically responding to questions based on contextual information. For instance, a system answers queries based on provided text.
9. Speech recognition: Identifying spoken words and converting them into text format.
10. Text classification: Accurately categorizing text into predefined classes, such as determining the sentiment of a movie review.

In our research, we focus on subtasks such as entity extraction and intent classification or text classification.

In summary, Some of the commonly used techniques for the task of entity extraction and intent classification include:

- **Rule-based extraction:** This method involves using predefined rules, such as regular expressions, to identify entities and intents in the text. This method is simple and fast, but not very flexible or scalable[6].
- **Dictionary-based extraction:** Dictionaries or directories are used to identify entities. This method is more flexible than rule-based extraction, but it still has limitations when it comes to working with named entities that are not in the dictionary[6].
- **Statistical models:** This method involves training machine learning models, such as Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs), on annotated data to identify entities in the text. This method is more flexible and scalable than rule-based and dictionary-based methods, but it requires annotated training data and can be computationally expensive[7].
- **Machine learning and Deep Learning Models:** This method involves the use of machine learning models such as Random Forest, Decision Tree, and neural network-based models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to identify entities and intents in the text. This method has shown great success in entity extraction and other NLP tasks, but it requires a large amount of training data and computational resources[6].

3.2. Entity extraction:

Entity extraction[9] is the process of identifying and extracting important entities from unstructured text data. An entity can be any real-world object or concept, such as a person, organization, location, date, product, or event. It plays an important role across diverse applications, including information extraction, question answering, automatic summarization, and machine translation. Named Entities (NEs) refer to words or phrases that are grouped into specific categories related to a particular topic. These entities typically contain crucial information within a sentence, serving as significant attributes for the majority of language processing systems.

For instance, in the following sentence: “I am looking for a Mexican restaurant in the center of the town.”, entity extraction would identify the following entities:

Cuisine: Mexican

Location: center of the town

The process of Named Entity Recognition (NER) generally comprises three primary stages:

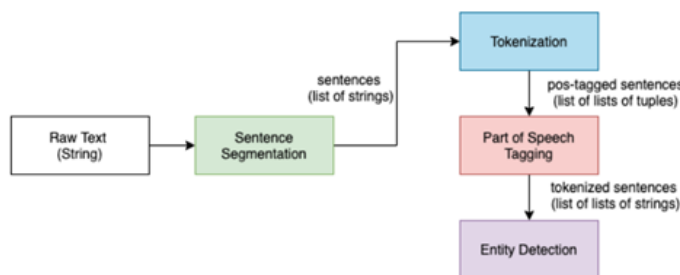


Figure 2: Working of Entity Extraction Model

1. **Tokenization:** During tokenization, the text is divided into separate words or tokens.
2. **Part-of-speech tagging:** In this step, each token is assigned a label corresponding to its grammatical part of speech, such as noun, verb, adjective, and so on.
3. **Entity classification:** The last step involves identifying and categorizing the named entities into predefined groups.

Figure 2, shows an example, highlighting the words in the sentence that represent Named Entities (NEs). During the model training process, a Named Entity Recognition (NER) system can be exposed to diverse entities, which may vary based on the specific project. Typically, pre-trained NER models encompass common entities like names, locations, organizations, dates, and more. The most commonly used NEs can be found in Table 1.

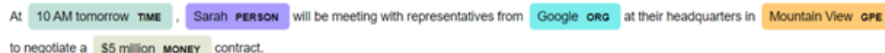


Figure 3: Named Entities present in text being highlighted

Table 1: Named Entity Types and Examples

NE Type	Examples
ORGANIZATION	Georgia-Pacific Corp., WHO
PERSON	Eddy Bonte, President Obama
LOCATION	Murray River, Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 million Canadian Dollars, GBP 10.40
PERCENT	twenty pct, 18.75%
FACILITY	Washington Monument, Stonehenge
GPE	South East Asia, Midlothian

Entity Extraction is usually performed using machine learning or deep learning algorithms that are trained on annotated text data. Annotated text data is text data that has been labeled with named entity categories. Machine learning algorithms use these annotated datasets to learn patterns and identify named entities in raw text data.

Some of the commonly used NLP algorithms to extract entities in NLP, include:

- Named Entity Recognition (NER): This is a common task in NLP that involves identifying and classifying named entities, such as names of people, organizations, places, and dates, in text. NER models can be rule-based or machine-learning-based and typically use techniques such as regular expressions, decision trees, and neural networks to identify and classify named entities[12].
- Regular Expression: This is a pattern-matching method that uses a set of rules to identify entities in text. Regular expressions are often used to extract entities in simple cases where a set of predefined patterns can be used to identify entities[13].
- Conditional Random Fields (CRF): This is a statistical learning technique that models the sequence of words in a text and predicts entity labels for each word. CRF models are often used for NER because they can effectively capture the relationships between words and entity labels in a sentence[7].
- Hidden Markov Models (HMM): This is a statistical model that can be used to predict the most likely sequence of entity labels for a given text[14]. HMMs are often used for NER because they can effectively capture the dependencies between entity labels in a sentence.
- Support Vector Machines (SVMs): This is a machine learning algorithm that can be used to classify text into different entity categories. SVMs are effective with high-dimensional data and can be used for NER when the number of entity categories is large[15].

- **Neural Networks:** This is a machine learning technique that can be used to identify and classify named entities in text[16]. Neural networks, such as recurrent neural networks (RNNs) and transformer models, are widely used for NER because of their ability to process complex language models and to capture dependencies between words and entity labels.
 - **Rasa DIET (Deep Information Extraction Transformations):** The Rasa DIET[11] model is a multilayer neural network that uses a combination of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to explore the connections between words, phrases, and concepts in natural language texts and can handle both intent classification and entity recognition. It provides the flexibility to plug and play various pre-trained embeddings like BERT, GloVe, ConveRT etc. To extract entities, Rasa DIET encodes the text into a sequence of vector representations and uses these representations to predict entity labels for each token in the text.

3.3. Need for custom Entity Extraction:

As previously discussed, it's well-known that entities are typically categorized into commonly used entity types when training NLP models with pre-trained resources. For instance, as depicted in Figure 4, the model identified entities like "quantity" and "time." However, our dataset primarily comprises network-related utterances, and generalizing these into standard entity categories isn't conducive to our specific requirements. In our case, the conventional approach to entity extraction doesn't align with our objectives. Instead, we opted for a customized approach to tailor the entity extraction task to our problem domain. This enables the development of models that accurately identify the desired entities within the context.

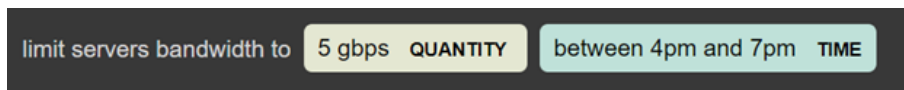


Figure 4: Incorrect Entity label classifications



Figure 5: Correct Entity label classifications

In doing so, we engaged in annotating the data with the precise entities we aimed to recognize—highlighted text as shown in Figure 5. To achieve this, we employed a combination of rule-based methods and manual annotation techniques to create labeled data for training purposes. Various annotation tools, such as Brat[17], Prodigy[18], and NER Annotator[19], can be employed for this purpose. The details pertaining to the data annotation process are elaborated upon in Section 4 of this document.

In summary, Entity extraction plays an important role in NLI by allowing systems to understand the meaning of the text and process text data efficiently. By identifying relevant entities and relationships, NLI systems can provide more accurate and relevant information retrieval, answer questions, and perform actions, resulting in a more natural and user-friendly experience.

3.4. Intent classification:

Intent classification in NLP (Natural Language Processing) [10] refers to the task of determining the underlying intent or purpose behind a piece of text. It is a type of text classification task where the goal is to predict the class label that best describes the intent of the text.

For example, in a customer service chatbot scenario, intent classification can help determine whether a customer is asking for a product, asking for support, or asking for a refund. The

model analyzes text received from a customer, such as a suggestion or question, and then predicts the most likely intention based on training data. For instance, in the following sentence: "I am looking for a Mexican restaurant in the center of the town.", intent extraction would identify the following intent:

Intent: RestaurantSearch

Intent classification models are typically trained on a large set of text sample data along with corresponding intention labels. The model learns to identify patterns in the text that indicate certain intentions and uses this information to predict new text data. The process of intent classification generally comprises following stages:

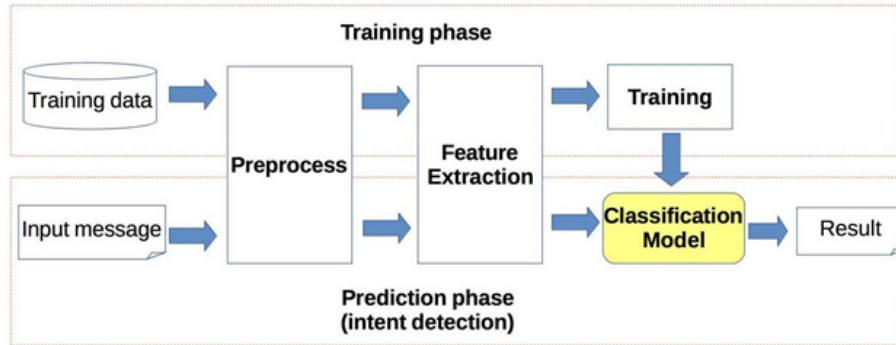


Figure 6: Steps in Intent Classification

- **Data Collection:** Gathering a diverse and representative dataset that covers a wide range of user queries and intents is crucial for training a robust Intent Classification model.
- **Preprocessing and Tokenization:** The collected text data undergoes preprocessing to remove noise and standardize the format. Tokenization breaks down the text into individual words or tokens, forming the basis for subsequent analysis.
- **Feature Extraction:** Relevant features are extracted from the tokenized text. These features encompass linguistic attributes, contextual information, and syntactic structures that aid in understanding the underlying intent.
- **Model Selection and Training:** Choosing an appropriate classification model is pivotal. Various models, including traditional machine learning algorithms and deep learning architectures, can be considered. The selected model is trained on the dataset, learning to map input queries to their corresponding intents.
- **Model Evaluation and Optimization:** The trained model is rigorously evaluated using metrics to measure its accuracy and generalization capabilities. Optimization techniques are applied to fine-tune model parameters and enhance performance.
- **Deployment and Real-time Prediction:** Once the Intent Classification model demonstrates satisfactory performance, it can be deployed for real-time applications, effectively categorizing user queries.

Some of the commonly used NLP algorithms for Intent classification tasks include:

- **Logistic regression:** Logistic regression [20] is a simple but effective algorithm that can be used to classify intentions. It is especially useful when the number of classes is limited and the amount of training data is small.
- **Support Vector Machines (SVM):** SVM [15] is another commonly used algorithm for intention classification. It works well for a limited number of classes and when the data are not linearly separable.

- Naive Bayes: Naive Bayes[21] is a fast and simple algorithm that can be used to classify intentions. It is based on Bayes' theorem and assumes that the features in the data are independent.
- Recurrent Neural Networks (RNN): RNN[22] is a popular deep learning algorithm that is often used for intention classification. It is particularly useful when the data contains a sequence of words or sentences and context is important.
- Convolutional Neural Networks (CNNs): CNN[23] is another deep learning algorithm that can be used for intention classification. It is especially useful when the data is structured and patterns need to be identified.
- Rasa DIET (Deep Information Extraction Transformations): RASA DIET[11] have become very popular for NLP tasks, including intention classification. They work well with large amounts of unstructured textual data and can capture complex relationships between words and phrases. For intent classification, DIET uses a pre-trained model and represents the text as a sequence of tokens or words and encodes these tokens in a multi-layer Transformer network. The resulting vector representations of the tokens are then fed into a classifier, such as a feedforward neural network, to make the final prediction of the user's intent.

In summary, Intent classification is an important component in creating conversational AI systems such as chatbots, virtual assistants, and customer service applications because it helps provide users with more personalized and relevant responses.

3.5. Word Embeddings:

Word embeddings[24] enable the conversion of text data into numerical vectors. Each word is represented by a fixed-length vector, and similar words possess comparable representations, as illustrated in Figure 7[25], where the colored vectors signify the similarity between words. These word embeddings are commonly pretrained and often serve as the initial data processing layer in deep learning models. However, they can also be learned during the training process.

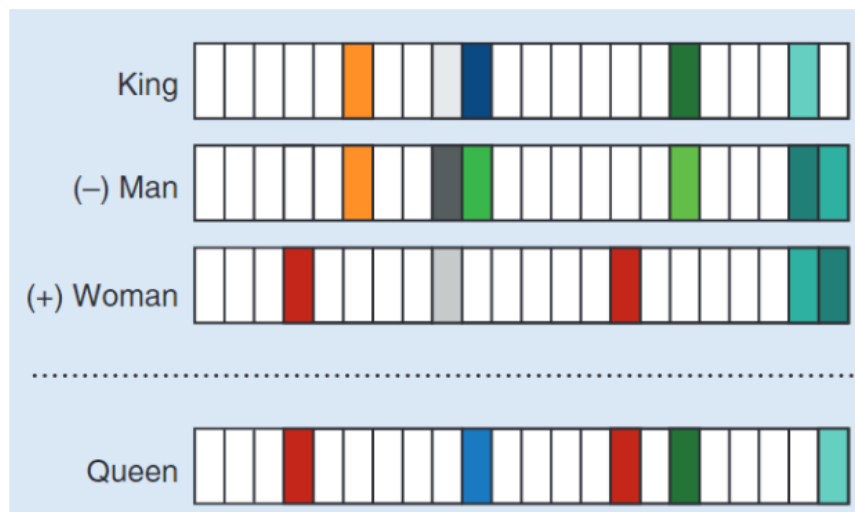


Figure 7: Word embeddings represented by a D-dimensional vector

Different types of word embeddings exist, depending on their training methods. Some of the most common ones include Word2Vec[26], TF-IDF[27], and GloVe[28]. In our study, TF-IDF is employed as a word embedding technique for the intent classification task. On the other hand, for entity classification, we utilize built-in feature extraction methods such as those provided by NLTK and Stanford NER. Additionally, we use tok2vec[26] for SpaCy and a CountVectorsFeaturizer[25] for RASA.

TF-IDF, short for Term Frequency-Inverse Document Frequency transform text documents into numerical feature vectors that can be effectively employed in various machine learning tasks. Term Frequency (TF) is a metric that gauges the significance of a term within a specific document. It is computed by dividing the frequency of a term (word) within a document by the total number of terms present in that document. This concept is rooted in the notion that terms occurring more frequently within a document tend to hold greater importance for that particular document. Inverse Document Frequency (IDF) is a measure of the rarity or distinctiveness of a term across a collection of documents. It is calculated as the logarithm of the total number of documents divided by the number of documents containing the term. Terms appearing in only a few documents are assigned higher IDF values, indicating that they offer more informational or distinctive content compared to terms present in numerous documents.

The calculation of TF-IDF involves multiplying the TF value of a term in a document by its corresponding IDF value. This computation assigns greater weights to terms that are both highly frequent within a specific document (high TF) and relatively scarce across the entire collection of documents (high IDF).

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \quad (1)$$

Where, t is the term (word) in question, d is the specific document and D is the collection of all documents.

The individual components can be defined as follows:

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d} \quad (2)$$

$$\text{IDF}(t, D) = \log \left(\frac{\text{Total number of documents in collection } D}{\text{Number of documents containing term } t} \right) \quad (3)$$

The process of vectorization ensues once TF-IDF values have been determined for all terms in each document. These computed values are utilized to construct numerical feature vectors that represent each document. Each individual term becomes a distinct feature dimension, and its associated TF-IDF value is indicative of the value along that particular dimension.

3.6. Tools and Libraries for Entity Extraction and Intent Classification:

There are several popular Python libraries that can be used for Entity Extraction, Intent Classification and Semantic parsing in NLP. The python libraries used in our work are:

- spaCy: It is an industrial-strength NLP library[29] that provides efficient and easy-to-use intent classification, entity extraction systems and dependency parsing. It supports training of custom intent classifiers and entity recognition models, and includes pre-trained models for various tasks.
- Stanford NLP: It is a suite of Java-based NLP tools developed by Stanford University, including a part-of-speech tagger, named entity recognizer, and parser. It can be used in Python using the PyNLPI library and provides a Python interface for various tools[30].
- Scikit-learn: This is a machine learning library for Python that provides a simple and efficient interface for training and evaluating machine learning models. It includes several classifiers, such as support vector machines (SVMs) and decision trees, which can be used to classify intentions, and entity extraction can be performed using the Named Entity Recognition (NER) module of the library[31].
- NLTK: This is a comprehensive library for natural language processing tasks including tokenization, stemming, tagging, parsing, semantic analysis, intention classification and named entity recognition modules. The intention classification module is based on the maximum entropy model, and the named entity recognition module supports entity extraction for different types of named entities.[32]

4. Related Works

Natural Language Processing (NLP) has undergone a series of developmental stages in its relatively recent emergence. Natural Language Processing (NLP) algorithms [5] are a key component of natural language interfaces, as they are responsible for interpreting and processing user input and generating appropriate responses. In this thesis, we are mainly interested in the sub-tasks of NLP, which includes Entity extraction[9], intent classification[10]. There is a growing body of research on the use of NLP algorithms for such NLP tasks, including rule-based, statistical, and hybrid methods[6] in the NLP interface to improve developer usability. Initially, during the 1980s, the rule-based approach was a significant milestone, involving the manual creation of rules to address language processing tasks. Noteworthy systems of that era included SHRDLU[33] developed by Winograd, which operated in limited block worlds with restricted vocabularies, and ELIZA by Weizenbaum[34], a conversational program that employed pattern matching and substitution for human-machine interactions.

The subsequent phase marked the statistical revolution, where NLP research extensively embraced statistical models, utilizing machine learning (ML) algorithms such as decision trees to make probabilistic decisions based on input data[35]. A more recent paradigm shift emerged with the integration of deep neural architectures, specialized hardware, and the abundance of available data[36]. This evolution has been driven by the vast textual data accessible on the internet, contributing to a surge in NLP research and creating novel applications for natural language interfaces.

There have been a number of studies and successful implementations of projects focusing on the use of natural language interfaces for developer experience. One early example is the "IntelliSense"[37] feature in Microsoft's Visual Studio development environment, which uses NLP techniques to provide code completion suggestions and documentation links as the developer types. Another example is the "Stack Overflow Assistant"[38] project, which uses a combination of NLP and machine learning to provide answers to developers' questions based on data from the Stack Overflow programming Q&A website. More recently, there has been a trend towards the use of chatbots as natural language interfaces for developer experience. One example is the "GPT-3 Developer Assistant,"[39] which uses the GPT-3 language model to provide assistance to developers on a variety of topics, including code debugging and library documentation. In addition to the specific systems and studies mentioned above, there are a number of other notable examples of natural language interfaces for developer experience such as CodeGenie[40], CodeFlow[41], DeepCode[42]. There have also been research studies on the use of natural language interfaces for other aspects of the developer workflow, such as requirements gathering and project management. For instance, the "NLPlanner"[43] system uses NLP to allow developers to create software development plans using natural language input, and the "Requirement Miner"[44] system uses NLP to extract requirements from natural language project descriptions. These systems have demonstrated the potential of natural language interfaces to improve the developer experience and make software development tools more accessible and user-friendly.

Z. Liu, et al.[45] present a comparative study of NLP algorithms for intention classification and entity recognition in the development of NLP interfaces, compares the performance of rule-based systems, statistical methods, and deep learning approaches mainly Logistic regression, conditional random fields (CRFs), Naive Bayes classifiers, hidden Markov models (HMMs), BERT and highlights trade-offs between accuracy, efficiency, and computational resources. Similarly, J. Kim, et al [46] in their research present a rule-based system for classifying intentions and entity recognition in the development of NLP interfaces, demonstrating the effectiveness of handwritten grammars and dictionaries for simple and easily implemented NLP applications.

Further, X. Li, et al.[47] in their research present a deep learning-based approach to intention classification and entity recognition in the development of NLP interfaces, using a combination of CNNs, RNNs, and LSTMs to achieve high accuracy and efficiency in natural language input processing. Moreover, Y. Zhang, et al. [48] present an empirical evaluation of transfer learning approaches for entity recognition and intention classification in the development of NLP interfaces, comparing the performance of BERT and GPT-2 with traditional statistical methods such as HMMs, MEMs, and CRFs.

Moreover, xu, et al. [49] present an ensemble approach to intention classification and entity recognition in the development of an NLP interface, combining the results of several NLP algorithms

to improve the overall accuracy and stability of the NLP system. The paper shows that the ensemble approach outperforms individual algorithms and emphasizes the importance of combining different NLP algorithms when developing an NLP interface.

Pilault, et al.[50] focuses on transfer learning for intention classification and entity recognition in the development of NLP interfaces, exploring the use of pre-trained NLP models such as BERT and GPT-2 to fine-tune and adapt to a target NLP application. The paper shows that transfer learning can significantly improve the accuracy and efficiency of NLP systems, and emphasizes the importance of pre-trained models in the development of NLP interfaces.

T. Nguyen, et al.[51] presents a framework for integrating NLP algorithms to improve developer experience when developing NLP interfaces, demonstrating how different NLP algorithms can be combined and optimized to improve the accuracy and efficiency of NLP applications. The paper demonstrates the importance of integrating NLP algorithms to improve developer experience when developing NLP interfaces, and presents a roadmap for future research in this area.

Similarly, previous research has extensively leveraged a variety of NLP tools for the purpose of Named Entity Recognition (NER). For instance, a study by Ramachandran et al. [52] utilized Spacy for entity extraction in the medical domain, emphasizing its high performance and efficiency. Likewise, Johnson et al.[53] employed NLTK in their research on historical text analysis, citing its flexibility and extensibility as key factors for their choice. Stanford NER has also been a popular choice; it was used by Vychezhnanin et al. [54] in a benchmark study to evaluate the efficacy of different NER tools across multiple languages. The work confirmed Stanford NER’s high accuracy and robustness, particularly in handling complex syntactic structures. More recently, the RASA DIET architecture was adopted by Sharma et al. [55] for a chatbot application in customer service. Their work demonstrated that RASA DIET is particularly well-suited for real-time applications where both intent recognition and entity extraction are required. These examples illustrate the diversity of applications and domains where these NER tools have been successfully employed.

While many preceding investigations regarding entity and intent categorization in the realm of natural language interfaces lean heavily towards deep learning approaches, our exploration veered towards simpler algorithms. Notably, prior studies in the network-related domain zoomed in on a singular model, predominantly the deep learning-based RASA. However, our research took a broader lens, examining multiple models. In our work, we have employed four different Entity Recognition techniques: MaxEnt Classifier with NLTK, Spacy, Conditional Random Fields with Stanford NER, and RASA DIET, and six different algorithms for the task of intent classification: SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, and Random Forest, along with RASA DIET. Unlike previous studies that primarily assessed the performance of NLP algorithms in the network management domain based solely on performance metrics, this research goes a step further by incorporating system metrics, which are paramount in evaluating the feasibility of these models in real-world scenarios.

5. Research Methodology

This section outlines the comprehensive approach adopted to address the thesis’s objectives. The core objective of this research revolves around evaluating the efficiency and effectiveness of various NLP algorithms. A comprehensive and iterative approach was adopted to ensure that the selected algorithms were not only well-suited for the task but also met the system’s requirements.

The research methodology is guided by the quantitative research design principles outlined by Creswell et al. [56], where the performance of each algorithm was compared under the same conditions.

• Problem Formulation

The initial phase started with engaging with the primary stakeholder, Nokia, to gain a deeper understanding of their low-code/no-code system and the challenges associated with it. This involved one-on-one interviews, focus group discussions, and observation sessions. Understanding the stakeholder perspective ensures that the research is grounded in real-world challenges and aids in formulating the precise research questions that the subsequent stages address.

- **Literature Review**

Post problem formulation, a literature review was conducted. This phase aimed to identify, evaluate, and interpret all available research relevant to our first research question, specifically focusing on AI conversational assistants and NLP algorithms related to entity extraction and intent classification. The review process was structured using established guidelines^[57] to ensure a comprehensive and unbiased assessment. We implemented search strategies that extracted data on key topics such as the nature of the algorithm, its performance indicators, and its domain of application. Databases like IEEE Xplore¹, ScienceDirect², and Springer³ were scoured using search terms such as "NLP algorithms", "conversational AI", "entity extraction", and "intent detection". To maintain topical relevance, only articles that underwent peer review within the past decade were included in our review. Exclusions were made for redundant studies, articles not in English, or those not entirely accessible.

This rigorous approach enriched our understanding of the domain, confirming that our selected algorithms for testing align with cutting-edge practices. Recognizing existing approaches, their merits, and their drawbacks allows us to establish a foundation for our experiments and contextualize our findings.

- **Experimental Design**

For this study, we adopted a Comparative Experimental Design, influenced by the principles outlined by Wohlin et al. ^[58]. This approach enables the concurrent evaluation of different treatments (here, NLP algorithms) within controlled conditions, facilitating a comprehensive comparison of the outcomes of the algorithms under study and answering our second research question.

The nature of the experiment in our case is quantitative. Numerical metrics were chosen to evaluate and compare the treatments. The comparison was made on the basis of quantitative metrics including system metrics like precision, recall, f1 score, computation time, and resource consumption. Through the comparison of these metrics, the experiment aims to assess the performances of the different NLP algorithms and determine the suitable ones.

Subsequent sections delve deeper into the foundational structure of our experiments, namely Experiment 1: Entity Extraction and Experiment 2: Intent Classification.

- **Programming Tools:** The experimentation process was carried out using the Python programming language. Essential Python libraries for this experiment encompassed visualization tools like seaborn, machine learning libraries like scikit-learn, and NLP tools such as spaCy, Stanford NLP, NLTK, with NER Annotator facilitating custom annotations.
- **Controlled Variables:**
 - * **Evaluation Environment:** The computational setup remained consistent to ensure results were not affected by external factors.
 - * **Data:** Across all algorithmic experiments, the size and integrity of the data were maintained.
- **Dependent Variables:**
 - * **Performance Metrics:** These include precision, recall, and f1 score.
 - * **System Metrics:** Metrics like memory consumption, processing duration, and resource usage were observed.
- **Independent Variables:** The model features served as the independent variables in our experiment. These attributes, such as different entity and intent types, fed into the models, possibly influencing the dependent variables. Tweaking these features could yield varied outcomes.
- **Assumptions:** Certain assumptions underpinned our research:

¹<https://ieeexplore.ieee.org/>

²<https://www.sciencedirect.com/>

³<https://www.springer.com/>

- * **Data Representativeness:** The selected dataset is believed to aptly mirror the real-world language queries that the Nokia system might encounter.
 - * **Algorithmic Compatibility:** There’s an underlying presumption that the algorithm’s performance in our study would also be reflected in a similar fashion when integrated into the Nokia system.
- **Design Procedure:** The experiment follows these steps:
1. **Algorithm Selection:** Algorithms were selected based on the literature review as explained in previous section.
 2. **Dataset Selection:** Established benchmark datasets were utilized for entity extraction, ensuring that the datasets capture a wide variety of entity types and sentence structures[59].
 3. **Data Pre-processing:** Following the guidelines from Garcia et al.[60], data was sanitized to remove irrelevant or noisy content. Custom annotations were added by tagging more entities and intents, adhering to the standards by Ruppenhofer et al.[61]. The data was then converted into a format suitable for NLP tools, as suggested by Cho et al. [62]. For training and validation, a 70:30 split was adopted, as recommended by Kohavi et al.Kohavi *et al.* [63].
 4. **Algorithm Training:** Each algorithm was trained using the designated training set. Parameter optimization was achieved using cross-validation [63].
 5. **Evaluation:** We gauged each algorithm’s computational requirements. Relying on insights from Han et al. [64], we observed each algorithm’s effect on computation time, memory, and CPU utilization. Performance was analyzed through established metrics by Sokolova et al.[65], focusing on precision, recall, and the F1 score.

- **Comparative Analysis**

After conducting the experiments, a methodical comparison of the obtained results for each NLP algorithm was undertaken. This comparison not only aids in understanding the performance of the algorithms but also offers actionable insights to identify the effective solution for our specific needs. The comparison was done for the following reasons:

- To distinguish the strengths and weaknesses of each technique in relation to entity extraction and intent classification.
- To establish a reference framework for future studies and optimizations.

6. Approach:

In this section, we outline the two experiments used in our work to compare and evaluate the performance of different NLP algorithms for entity extraction and intent classification tasks. Each experiment relies on specific algorithms, which we’ll discuss in detail. Before diving into these methods, we’ll start by explaining the primary steps common to most ML projects: data collection, preprocessing, and Exploratory Data Analysis (EDA). These steps are essential because they prepare the data for further processing and give us an initial understanding of its features and patterns.

6.1. Data:

The data utilized in this research is sourced from the Lumi Nile Chatbot Dataset[66], available at <https://github.com/lumichatbot/chatbot/tree/master/nlu-engine/data>. This dataset was originally developed in conjunction with the 2021 USENIX Annual Technical Conference paper, "Hey, Lumi! Using Natural Language for Intent-Based Network Management,"[67] authored by a research team led by Arthur S. Jacobs. The dataset aims to facilitate the use of natural language in managing intricate networking tasks and consists of utterances specifically related to network operations. These utterances enable network operators to convey their high-level intentions using natural language. The dataset itself has been generated based on real-world intents obtained from

U.S. university network policies, and the utterances are constructed as if network operators are directly interacting with a management system.

The Lumi Nile Chatbot Dataset is pivotal to the present study, serving as the foundational resource for the training and evaluation of natural language understanding models implemented in intent-based network management systems.

6.1.1.1 Data Composition:

In the dataset used for this research, three primary components contribute to the robustness and flexibility of the Natural Language Understanding (NLU) system: utterances, lookup files, and synonym files. *Utterances* are natural language expressions that serve as the raw input for the NLU system. These utterances are processed to identify various *entity types*, which are predefined categories of data that the system needs to recognize and act upon.

To provide a brief understanding of the types of utterances obtained as raw data from the Lumi Nile dataset, we present the following examples; these examples illustrate the diversity of entities belonging to a certain entity type and the entity types that the system can recognize and process.

Table 2: Raw utterances as obtained from Lumi Nile dataset

No.	Example Utterance
1	From [dorms](location) to [internet](location) [add](operation) [nat](middlebox)
2	[unblock](operation) [facebook](service)
3	[Add](operation) [DPI](middlebox) middlebox into [all traffic](traffic) from [Internet](location) to the [Labs](location)
4	[limit](qos_constraint) [servers](location) [bandwidth](qos_metric) to [5](qos_value) [gbps](qos_unit) between [4pm](hour) and [7pm](hour)
5	[block](operation) all [student](group) traffic in the [network](location)
6	[Set](operation) [quota](qos_metric) to [10](qos_value) [GB per week](qos_unit) at [dorms](location)
7	[Limit](qos_constraint) [students](group) in the [dorm](location) to [10](qos_value) [GB per week](qos_unit)
8	Make sure [all traffic](traffic) from the [Internet](location) to the [labs](location) is [inspected](middlebox) by the [DPI](middlebox) middlebox
9	[block](operation) [F2movies traffic](service) for [students](group) in the [labs](location)
10	[Students](group) shall [download](qos_constraint) [no more than](qos_metric) [1000000](qos_value) [MB per week](qos_unit)

The raw data obtained from the Lumi Nile dataset follows a specific annotation format to indicate entities and their corresponding types within the natural language utterances. In this format, square brackets [] are used to encapsulate the entities, and parentheses () immediately following the square brackets are used to specify the type of the entity.

For instance, consider the example utterance "[limit](qos_constraint) [servers](location) [bandwidth](qos_metric) to [5](qos_value) [gbps](qos_unit) between [4pm](hour) and [7pm](hour)". In this utterance, the words inside the square brackets, such as "limit", "servers", "bandwidth", "5", "gbps", "4pm", "7pm", are the entities. The words inside the parentheses, such as "qos_constraint", "location", "qos_metric", "qos_value", "qos_unit", "hour", "hour", indicate the types of these entities, respectively.

Similarly, The *lookup files* provide examples of the entity types present in the dataset, effectively serving as a reference guide for the NLU system to better understand and categorize the entities

present in the utterances. In the scope of the entity recognition task in our research, there are 11 distinct entity types to be recognized. A brief explanation of each of these entity types and their examples is provided in the subsequent sections to offer a comprehensive understanding of their roles and significance in the NLU system.

- **Location:** Refers to the physical or logical place where a network service, device, or user is situated.
- **QoS_Constraint:** Specifies the limitations or conditions under which a network service must operate.
- **Group:** Represents a collection of users or devices that share common attributes or permissions.
- **Hour:** Indicates the time of day, often used to specify when certain network rules or limitations apply.
- **QoS_Metric:** Describes the measurable attributes related to the quality of a network service.
- **Service:** Refers to the specific network services or applications that are being used or managed.
- **Traffic:** Describes the type or category of network data being transmitted or received.
- **QoS_Value:** Specifies the numerical value associated with a QoS metric or constraint.
- **Operation:** Describes the action to be taken on a network service, often in the context of permissions or limitations.
- **Middlebox:** Refers to any intermediary device on a network that performs functions other than just forwarding packets.
- **QoS_Unit:** Indicates the unit of measurement for a QoS metric or constraint.

Table 3: Examples of Entity Lookups

Entity Type	Examples
Location	video server, my network, labs, HR department, University of Illinois
QoS_Constraint	download, upstream, capped at, minimum, incoming,
Group	all professors, teachers, guests
Hour	1am, 2pm, noon
QoS_Metric	bandwidth, download rate, speed
Service	YouTube, Netflix, email
Traffic	news, streaming, music, all traffic
QoS_Value	1, 10, 100
Operation	add, block, allow
Middlebox	NAT, load balancing, intrusion prevention
QoS_Unit	gigabytes, Mbps, Kbps

Complementing the lookup files, the *synonym files* contain alternative expressions or words that map to the same entity. *Synonyms* are employed to accommodate the variability in how

different users express the same idea, including case sensitivity and spelling errors, thereby making the system more flexible and user-friendly. For example, the time "01:00" and "1am" could be synonyms for the same canonical representation of time. The inclusion of lookup and synonym files not only enhances the system's ability to understand varied user inputs but also allows for a more nuanced and comprehensive understanding of natural language.

Table 4: Examples of Synonyms

Synonym	Examples
01:00	01:00, 1am, 13:00
02:00	02:00, 2am, 14:00
Battlenet	blizzard battlenet, Battlenet, battlenet
DMZ	DMZ, demilitarized zone
HR	HR, human resources, Human Resources, HR department
LAN	local area network, LAN
add	add, append, include, increase, must pass through, pass through, forward, forward through, affix
allow	allow, allowed, allowing, grant, let, permit, let on, authorize, pass, admit, tolerate, do not block, must not be blocked, unblock
block	block, blocked, blocking, can not run, cannot run, deny, do not let, may not run, prevent, stop, obstruct, cut off, deter, halt, intercept, impede, stall, thwart, hinder, hold up, close, close off, bar, barred, do not allow, must not be allowed
campus	campus, entire campus
classrooms	classroom, classrooms, class
copyright monitoring	copyright, copyright violation, copyright-monitoring

6.1..2 Data Synthesis:

In the initial phase of our research, we obtained a dataset from Lumi Nile consisting of 141 utterances. While this dataset provided a valuable starting point, its limited size posed a challenge for training robust machine learning models. To address this issue, we decided to augment the dataset by generating synthetic data.

We used Gretel AI[68], a leading platform specializing in data synthesis and privacy-preserving data generation, to extend our dataset. Gretel AI utilizes advanced machine learning algorithms to generate synthetic data that retains the statistical properties and relationships inherent in the original dataset. This approach allows for the creation of data that can serve as a viable substitute for real data, enabling organizations to conduct analyses and develop models without compromising privacy or security.

For the data synthesis process, we configured Gretel AI with the following parameters:

- Model: GPT_X
- Batch Size: 4
- Epochs: 3
- Weight Decay: 0.01
- Learning Rate Scheduler: Linear
- Learning Rate: 0.0002

- Steps: 100
- Rows: 500
- Data Source: Existing 141 sentences from Lumi Nile dataset

The machine learning model used for this task was a variant of the Generative Pre-trained Transformer, denoted as GPT_X. The `batch size` was set to 4, indicating that four data samples were processed in a single batch during the training phase. The model was trained for 3 `epochs`, each epoch being a complete forward and backward pass of all the training examples. To prevent overfitting, a `weight decay` of 0.01 was applied as a regularization technique. The `learning rate scheduler` was set to linear, gradually decreasing the learning rate over time. The initial `learning rate` was set at 0.0002, dictating the step size in the optimization algorithm. The total number of optimization steps was fixed at 100, denoted as `steps`. The model was configured to generate 500 rows of synthetic data, specified by the `rows` parameter. For the data source, we used the existing 141 sentences from the Lumi Nile dataset.

The model took approximately 6 minutes to generate 500 synthetic utterances. Upon inspection of the newly generated data, we found that it contained numerous duplicates and random words that did not contribute meaningfully to the dataset. As a result, we undertook a manual filtering process to remove these inconsistencies. The final dataset, post-filtering, consisted of 199 high-quality synthetic utterances. The data is further split for training and testing purposes in the ratio of 70:30 for both entity and intent classification models.

6.1.3 Data Processing:

In our research, the initial dataset sourced from the Lumi Nile project came pre-labeled, with entities enclosed in square brackets and their corresponding entity types denoted within parentheses. While this format is informative, it is not directly compatible with the natural language processing tools we employed for entity recognition, namely Spacy, StanfordNER and NLTK and for the task of intent classification as well. This conversion was a crucial preprocessing step, enabling us to utilize advanced entity recognition and intent classification algorithms for further analysis.

6.1.3.1 Conversion of data into IOB format:

To bridge this gap, we converted the raw data into the IOB (Inside-Outside-Beginning) format, which is a common tagging format for named entity recognition and is supported by both StanfordNER and NLTK. In the IOB format, each token in a sequence is tagged with one of three labels:

- **B (Beginning)**: Indicates that the token marks the beginning of a named entity.
- **I (Inside)**: Signifies that the token is a part of a named entity but is not the first token in the sequence.
- **O (Outside)**: Denotes that the token does not belong to any named entity.

To illustrate the IOB labeling process, consider the sentence: “John Smith lives in New York and works for United Nations.” First, the sentence is tokenized into individual words. Each token is then labeled according to the IOB format. In this specific example:



Figure 8: Example of IOB label

- The token “John” is tagged as B-PER, indicating the beginning of a person’s name.

- “Smith” is tagged as I-PER, signifying that it is part of a person’s name but not the first token.
- “United” is tagged as B-ORG, marking the beginning of an organization’s name.
- “Nations” is tagged as I-ORG, indicating that it is part of an organization’s name but not the first token.
- All other tokens are tagged as O, denoting that they do not belong to any named entity.

In our study, sentences were converted into IOB format as follows. Consider the two example sentences: “Block F2movies traffic from Internet to student Labs” and “Add DPI middlebox into all traffic from Internet to the Labs.” These sentences are tokenized and labeled as shown below:

Block	B-operation
F2movies	B-service
traffic	I-service
from	O
Internet	B-location
to	O
student	B-location
Labs	I-location
Add	B-operation
DPI	B-middlebox
middlebox	I-middlebox
into	O
all	B-traffic
traffic	I-traffic
from	O
Internet	B-location
to	O
the	O
Labs	B-location

Figure 9: Example of tokens and their corresponding IOB labels

It should be noted that the dataset after conversion consists of two columns: the token and its corresponding entity label. These columns are separated by a tab. Additionally, each sentence in the dataset is separated by a new line.

To reformat the data as depicted in Figure 9, we utilized pre-annotated raw data from the Lumi Nile dataset. The transformation was carried out using the algorithm outlined in the accompanying pseudo-code:

```

1: Initialize:
2:   converted_lines ← empty list
3:   prev_label ← None
4:   label ← None
5: Tokenize:
6:   words ← split input sentence
7: for word in words do
8:   if word starts and ends with square brackets then
9:     label ← content inside square brackets
10:    continue
11:  end if
12:  if label is not None then
13:    prefix ← "B-" if label ≠ prev_label else "I-"
14:    converted_line ← word + \t + prefix + label
15:  else
16:    converted_line ← word + \t + "O"
17:  end if
18:  append converted_line to converted_lines
19:  prev_label ← label
20:  label ← None
21: end for
22: result ← join converted_lines with newline
23: return result

```

Figure 10: Pseudo-code for Sentence to BIO Tag Conversion

The algorithm for converting sentences into the BIO (Beginning, Inside, Outside) tagging scheme uses basic string manipulation and list operations. The algorithm is initiated by tokenizing the input sentence into discrete words. Subsequently, it iterates through each word to check whether it is a label within the square brackets or a regular word. Upon encountering a label, it is temporarily stored for tagging the succeeding word(s). For each standard word, the algorithm verifies the presence of a label. If a label is present, the algorithm decides whether to use the "B-" or "I-" prefix, contingent on whether the label matches the label of the preceding word. Words devoid of labels are tagged with "O". The algorithm maintains a list of these tagged words, which are ultimately concatenated into a string, each tagged word appearing on a new line and separated by a tab character.

6.1..3.2 Conversion of data into Spacy format:

As various methods are available for annotating data in Named Entity Recognition (NER), including IOB, IOB2, BILUO tags, and JSON formats. In versions of spaCy prior to v3, the data had to be annotated in JSON format. However, starting with spaCy v3, this data is required to be converted into the `.spacy` binary format. To streamline this conversion, we developed a specialized function.

In spaCy, the training data for Named Entity Recognition (NER) is structured as a list of tuples. Each tuple contains a sentence and a dictionary. The dictionary, in turn, contains a key called "entities" that maps to a list of tuples. Each tuple in this list represents an entity in the sentence. The tuple contains three elements: the start position of the entity, the end position, and the type of the entity.

For example, consider the following sentence: *"limit servers bandwidth to 5 gbps between 4pm and 7pm"*. The corresponding spaCy format would be:

In this example, the entity *"limit"* starts at position 0 and ends at position 5, and is of type *"qos_constraint"*. Similarly, the entity *"servers"* starts at position 6 and ends at position 13, and is of type *"location"*. This format allows spaCy to understand both the entities present in the sentence and their types, thereby facilitating the training of the NER model.

```

("limit servers bandwidth to 5 gbps between 4pm and 7pm",
 {"entities":[(0,5,"qos_constraint"),(6,13,"location"),(14,23,"qos_metric"),
 (27,28,"qos_value"),(29,33,"qos_unit"),(42,45,"hour"),(50,53,"hour")]})

```

Figure 11: Example of spaCy format

To transform the data into the format outlined in Figure 11, we utilized pre-labeled raw data from the Lumi Nile dataset. We accomplished this transformation by implementing the algorithm described in the provided pseudo-code. This algorithm relies on elementary string manipulation and list operations to produce the specified output. The algorithm starts by splitting the input sentence into individual words. It then iterates through each word, checking if it is enclosed in square brackets, which would indicate that it is a label for the following word. If a label is found, it is stored temporarily, and the algorithm continues to the next word. The actual word is then appended to a list that will eventually form the converted sentence. Simultaneously, the start and end indices of the word in the converted sentence are calculated, and along with the previously stored label, they are added to a list of entities. Finally, the list of words is joined back into a single string to form the converted sentence, and a tuple containing this sentence and the list of entities is returned.

```

1: Initialize:
2:   converted_sentence ← empty list
3:   entities ← empty list
4:   start_idx ← 0
5:   label ← None
6: Split:
7:   words ← split input sentence
8: for word in words do
9:   if word starts and ends with square brackets then
10:    label ← content inside square brackets
11:    continue
12:   end if
13:   end_idx ← start_idx + length of word
14:   append word to converted_sentence
15:   if label is not None then
16:    append (start_idx, end_idx, label) to entities
17:   end if
18:   start_idx ← end_idx + 1
19: end for
20: converted_text ← join converted_sentence
21: return (converted_text, entities)

```

Figure 12: Pseudo-code for Sentence Conversion Algorithm

After transforming the data into the format illustrated in Figure 11, the next step involved converting this data into the `.spacy` binary format, which is compatible with spaCy’s machine learning models. To accomplish this, we utilized the `DocBin` class provided by spaCy for binary serialization of a collection of `Doc` objects. Specifically, we used the `add` method to include each `Doc` object into the `DocBin`. The `to_disk` method was then employed to save the serialized `DocBin` object to disk with a `.spacy` extension. One of the key advantages of using `DocBin` is its efficiency; it not only speeds up the serialization process but also generates smaller file sizes compared to Python’s native pickle module. Additionally, `DocBin` allows for secure deserialization without the need to execute arbitrary Python code.

6.1..3.3 Conversion of raw data into format suitable for Intent Classification:

In the context of intent classification, which is a multi-class classification problem in our study, it is imperative to separate the features from the labels. The model specifically requires the features, which in our case are the utterances, to be devoid of any entity labels. To achieve this, we preprocess the raw data, which is initially annotated with entity labels, to remove these labels and retain only the utterances.

For instance, consider the original sentence annotated with entity labels: `[limit](qos_constraint) [servers](location) [bandwidth](qos_metric) to [5](qos_value) [gbps](qos_unit) between [4pm](hour) and [7pm](hour)`. The Sentence Conversion Algorithm is employed to transform this sentence into a format suitable for intent classification. The algorithm utilizes regular expressions to efficiently strip the sentence of its entity labels as given in the accompanying pseudo-code:

Converted Sentence:

limit servers bandwidth to 5 gbps between 4pm and 7pm

Figure 13: Example of a Converted Sentence for Intent Classification

The converted sentence, as shown in Figure 13, is then used as the feature input for the intent classification model.

Require: Original sentence with text and labels in the format `[text](label)`

Ensure: Converted sentence with labels removed

```

1: converted_sentence ← original_sentence
2: matches ← find all occurrences of [text](label) using regular expression
3: for each match in matches do
4:   text, label ← match
5:   Replace [text](label) in converted_sentence with text
6: end for
7: return converted_sentence

```

Figure 14: Pseudo-code for Sentence Conversion Algorithm

The Sentence Conversion Algorithm employs regular expressions (regex) to transform an original sentence containing both text and labels into a simplified version with the labels removed. The algorithm takes as input a sentence where each word or phrase is annotated in the format `[text](label)`. The objective is to extract just the `text` portions and concatenate them to form a coherent sentence without any labels.

The algorithm initializes a variable, `converted_sentence`, with the original sentence. It then utilizes a regular expression to identify all occurrences of the pattern `[text](label)` within the sentence. Each match is a tuple containing the `text` and its corresponding `label`.

A loop iterates through each of these matches. Within the loop, the algorithm replaces each occurrence of `[text](label)` in `converted_sentence` with the extracted `text`. This is done until all such patterns in the original sentence have been replaced. Finally, the algorithm returns the `converted_sentence`, which is the original sentence stripped of all labels. This approach leverages the power of regular expressions for efficient text manipulation, making it particularly useful for natural language processing tasks where such transformations are common.

6.1..4 Data Annotations:

6.1..4.1 Custom Entity Labeling:

In our research, we focus on custom entity extraction, a specialized form of Named Entity Recognition (NER) that goes beyond the capabilities of general-purpose NER models. While standard NER models are effective at identifying common entities such as names, locations, and dates, they

often fall short in specialized domains. Custom entity extraction allows us to identify and classify domain-specific entities that are not covered by pre-trained models.

A critical step in developing a custom NER model is data annotation. This involves manually tagging or labeling raw data to create a ground truth that the machine learning algorithm can learn from. Various tools are available for this purpose, ranging from simple manual tools to more complex automated or semi-automated platforms. Some popular choices include Brat[17], Prodigy[18], and NER Annotator[19].

Our work involved using the Lumi Nile dataset, which was already annotated with entity labels. While this pre-labeling was advantageous as it provided a ready-to-use dataset, it posed challenges when we tried to adapt it for various NLP libraries such as SpaCy, NLTK, and Stanford NER. The primary issue was that the data was not in a format compatible with these libraries, necessitating a data conversion process.

During this conversion, specifically into Spacy format, we encountered specific inconsistencies related to the start and end indices of the labeled entities. Upon closer inspection, we realized that the algorithm we used for string manipulation was not accurately capturing the entities enclosed within square brackets and their corresponding types within parentheses. This led to incorrectly labeled entities and even resulted in some entities being missed altogether.

To address these challenges, we resorted to manual annotation using a tool called NER Annotator[19]. This tool provides a user-friendly interface for manually tagging entities in a text corpus. It allows the annotator to highlight text spans and assign them custom entity labels. The tool also offers features for reviewing and editing annotations, ensuring a high level of accuracy and consistency.

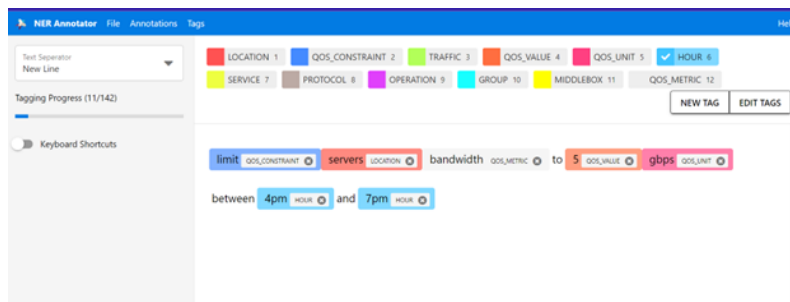


Figure 15: Manual Data Annotation Using NER Annotator

The NER Annotator tool provides a streamlined process for annotating text data for Named Entity Recognition (NER). The workflow is divided into several steps. **Step 1** involves loading a text file that contains the content to be annotated. It is advisable to break the content into paragraphs or passages and maintain a consistent separator between them, such as a newline or an empty line. For large datasets, it’s recommended to divide the text into smaller files. **Step 2** focuses on the actual annotation process. Users can create new tags using the *New Tag* button and remove unwanted tags with the *Edit Tag* button. After annotating an entry, the *Save* button should be clicked to proceed to the next entry. The tool also provides options to adjust the text separator and to export or import tags for team collaboration. **Step 3** allows users to download the annotated data as a JSON file suitable for machine learning training. Finally, the annotations can be converted to DocBin format for training, as per spaCy’s documentation.

By manually annotating a subset of our data, we were able to correct the inaccuracies in the dataset for Spacy and generate a reliable ground truth for training our custom NER model.

6.1.4.2 Custom Intent Labeling:

Our research utilized the Lumi Nile dataset, which came pre-annotated with entity labels and some intent labels. However, the intent labels provided in the dataset did not align with our specific requirements. Consequently, we needed to create our own custom intent labels.

As detailed in Section 4, we first converted the raw data into a format devoid of any labels. Subsequently, we employed rule-based techniques to generate new intent labels. For instance, if an utterance contained the word *"limit"*, it was automatically assigned the label *"limit"*. Following this methodology, we created five distinct labels: *"limit"*, *"add"*, *"block"*, *"allow"*, and *"unblock"*. To enhance the label assignment process, we also utilized a synonym file that listed alternative expressions for each label.

```

1: Input: List of sentences, Synonym files for each label
2: Output: List of labeled sentences
3: Initialize empty list labeled_sentences
4: for each sentence in sentences do
5:   label ← None
6:   for each word in sentence do
7:     if word or its synonyms match with "limit" then
8:       label ← "limit"
9:       break
10:    else if word or its synonyms match with "add" then
11:      label ← "add"
12:      break
13:    else if word or its synonyms match with "allow" then
14:      label ← "allow"
15:      break
16:    else if word or its synonyms match with "block" then
17:      label ← "block"
18:      break
19:    else if word or its synonyms match with "unblock" then
20:      label ← "unblock"
21:      break
22:    end if
23:  end for
24:  Add (sentence, label) to labeled_sentences
25: end for
26: return labeled_sentences

```

Figure 16: Pseudo-code for Sentence Labeling Algorithm

The algorithm takes as input a list of sentences and synonym files for each predefined label, such as "limit," "add," "allow," "block," and "unblock." The algorithm initializes an empty list called `labeled_sentences` to store the labeled sentences. For each sentence in the input list, it iterates through each word and checks if the word or its synonyms match any of the predefined labels. The first match found determines the label for the entire sentence, and the loop breaks to avoid multiple label assignments. The labeled sentence is then added to the `labeled_sentences` list. The algorithm returns this list as the output, providing a set of sentences each associated with a specific intent label.

Upon reviewing the newly labeled data, we found that the majority of the utterances were correctly annotated with the new intent labels. However, there were instances where the labeling was either incorrect or missing. To address these issues, we manually inspected the dataset and made the necessary corrections to ensure accurate intent labeling.

6.2. Model Architecture:

6.2..1 NLTK with MaxEnt Classifier

6.2..1.1 Natural Language Toolkit (NLTK):

The Natural Language Toolkit (NLTK) is a Python library that provides tools to work with human

language data (text). One of its features is the MaxEnt classifier, a machine learning model for classification tasks like Named Entity Recognition (NER). MaxEnt, short for Maximum Entropy, is a statistical model that belongs to the family of exponential models and is equivalent to logistic regression in its formulation.

Similar to other NER systems, NLTK also employs preprocessing steps like tokenization, stemming, and part-of-speech tagging to prepare the text for feature extraction.

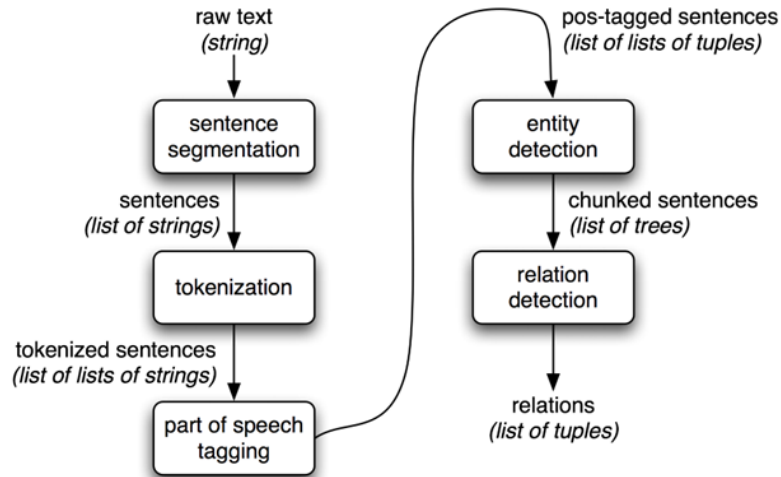


Figure 17: Model Architecture: Natural Language Toolkit (NLTK) for Entity Recognition

NLTK allows for custom feature extraction, enabling the user to define features that capture various lexical, syntactic, and semantic aspects of each word and its context.

6.2.1.2 Maximum Entropy (MaxEnt) classifier:

The MaxEnt classifier[69] in NLTK uses these features to compute probabilities for each entity type given the features of each token. The model consists of an input layer, potentially multiple hidden layers, and an output layer.

In MaxEnt, each token in the text is represented by a feature vector, capturing various attributes like the word itself, its part-of-speech tag, and its surrounding context. Each feature is associated with a weight, learned during the training process, which signifies the importance of that feature in entity classification. The MaxEnt model is based on the Principle of Maximum Entropy.

MaxEnt operates on the Principle of Maximum Entropy, which aims to make the least amount of assumption beyond the known facts (the training data) and select the model with the highest entropy from all models that fit the training data. The model is trained to maximize the likelihood of the observed data, given the features and their weights. Mathematically, the model is formulated as:

$$P(y|x) = \frac{1}{Z(x)} \exp \left(\sum_{i=1}^n \lambda_i f_i(x, y) \right)$$

where $Z(x)$ is a normalization factor, $f_i(x, y)$ are feature functions, and λ_i are the learned feature weights.

MaxEnt computes the conditional probabilities of each entity type given the features of each token. The model is discriminatively trained to adjust these probabilities such that the likelihood of the observed data is maximized. MaxEnt uses a statistical model to calculate the probabilities of each entity type for a given token based on its features. The model employs an optimization algorithm to adjust the weights of these features. Unlike other sequence models like Hidden Markov Models (HMMs), MaxEnt makes fewer independence assumptions. It focuses on the conditional

probability of the output sequence given the input sequence, allowing it to incorporate arbitrary overlapping features.

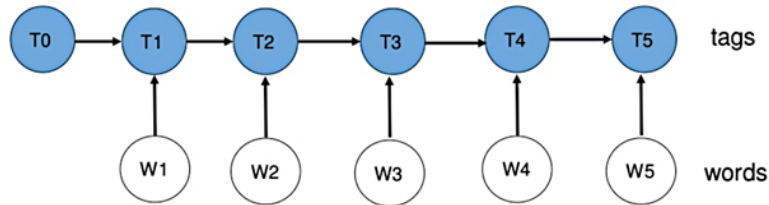


Figure 18: Model Architecture: Maximum Entropy (MaxEnt) classifier for Entity Recognition

The MaxEnt model is trained using methods like Iterative Scaling or Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) to maximize the likelihood of the observed data. During inference, the model calculates the probabilities for each class label and selects the one with the highest probability.

The MaxEnt classifier in NLTK offers a robust and flexible architecture for Named Entity Recognition. By maximizing the likelihood of the observed data and making minimal assumptions, MaxEnt provides a powerful mechanism for entity prediction. Its ability to handle a rich set of overlapping features makes it a versatile tool for various NLP tasks, including domain-specific NER.

6.2..2 Stanford NER with Conditional Random Fields for Entity Extraction:

6.2..2.1 Stanford’s Named Entity Recognition (NER):

Stanford’s Named Entity Recognition (NER) is a Java-based implementation that employs machine learning algorithms for the task of entity recognition. One of the key algorithms it utilizes is the Conditional Random Field (CRF), a form of statistical modeling particularly well-suited for sequence data like text.

Before feeding into the CRF model, the text undergoes several preprocessing steps. These may include tokenization, stemming, and part-of-speech tagging, which prepare the text for feature extraction.

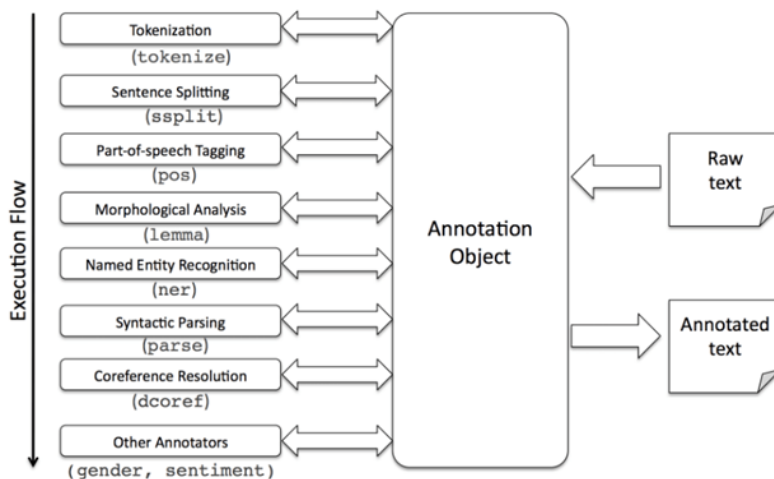


Figure 19: Model Architecture: Stanford NER for Entity Recognition

Stanford NER uses a comprehensive set of features that capture various aspects of each word and its surrounding context. These features can include the word’s stem, its part-of-speech tag,

orthographic features like capitalization, and contextual features such as adjacent words.

6.2..2.2 CRF Model

The core of Stanford NER is a CRF model trained on these features. The model consists of an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next, and the model is trained to minimize a loss function that considers both the predicted and true entity labels.

CRFs are a type of discriminative undirected graphical model. They model the conditional probability $P(Y|X)$, where Y is the output sequence (entity labels in our case) and X is the input sequence (the words in the sentence). The goal is to find the label sequence Y that maximizes this conditional probability, given a particular input sequence X .

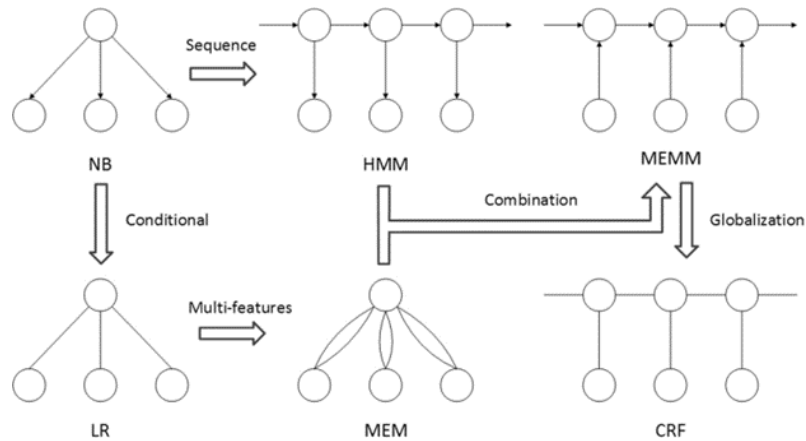


Figure 20: Model Architecture: Conditional Random Fields for Entity Recognition

CRFs use feature functions to capture the state of each word and its context. These feature functions are weighted, and the weights are learned during the training process. Mathematically, the conditional probability $P(Y|X)$ is modeled as:

$$P(Y|X) = \frac{1}{Z(X)} \exp \left(\sum_{i=1}^N \sum_{k=1}^K \lambda_k f_k(y_{i-1}, y_i, X, i) \right)$$

where $Z(X)$ is a normalization factor, f_k are feature functions, and λ_k are the learned weights.

CRFs are typically trained using maximum likelihood estimation. The inference problem, finding the most likely label sequence for a given input sequence, is often solved using dynamic programming algorithms like the Viterbi algorithm.

The Stanford NER, with its use of Conditional Random Fields, offers a robust architecture for the task of Named Entity Recognition. The CRF model allows for the incorporation of a wide array of features, capturing intricate patterns in the text. Its focus on conditional probability, as opposed to joint probability, allows it to make fewer independence assumptions, making it a powerful tool for sequence labeling tasks like NER.

6.2..3 spaCy for Named Entity Recognition:

spaCy is a state-of-the-art Natural Language Processing (NLP) framework that offers a range of functionalities, including Named Entity Recognition (NER). While spaCy provides pre-trained models capable of recognizing standard entities like names, organizations, and countries, it also allows for custom model training to recognize domain-specific entities.

spaCy's standard pipeline includes a tagger for part-of-speech (POS) tagging, a parser for dependency parsing, and an entity recognizer for NER. These components are pre-configured in spaCy's pre-trained models.

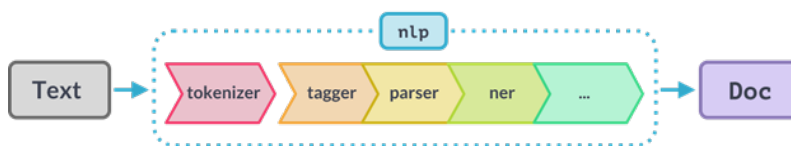


Figure 21: Model Pipeline: spaCy for Named Entity Recognition

The NER component in spaCy employs a transition-based model inspired by techniques used in dependency parsing to capture semantic and syntactic features of the text. This model predicts entities based on the surrounding context, previously predicted entities, and actions taken so far in the parsing process. This strategy is trained on a labeled dataset to learn the optimal sequence of actions for NER.

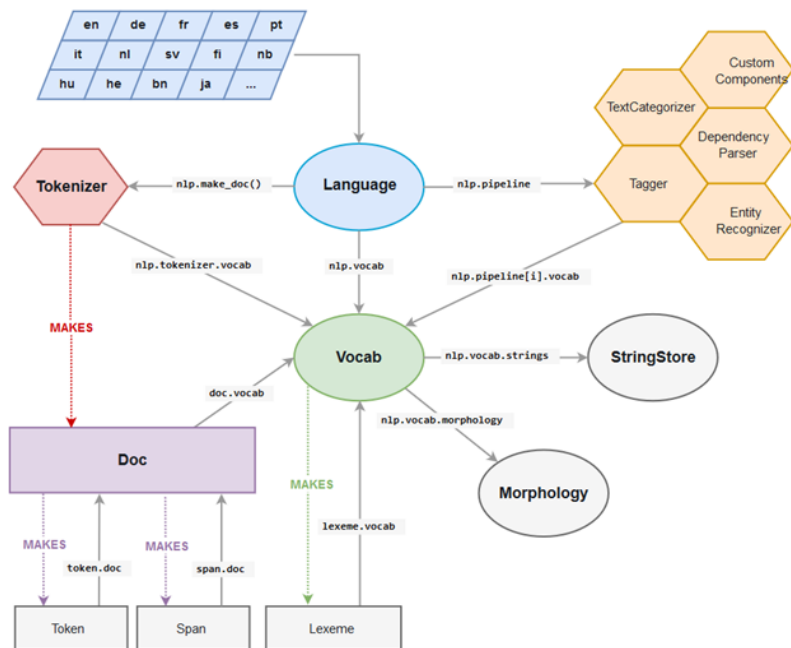


Figure 22: Model Architecture: spaCy for Named Entity Recognition

The model uses a series of actions like SHIFT, REDUCE, and BILUO tags (Begin, In, Last, Unit, Out) to annotate entities in the text. These actions and tags guide the model in identifying the boundaries and types of entities.

A blank model in spaCy serves as a foundational pipeline with only tokenization capabilities. Additional components like NER can be added to this pipeline.

spaCy offers a robust and flexible architecture for Named Entity Recognition, capable of handling both standard and custom entities. Its transition-based model, inspired by dependency parsing techniques, provides a powerful mechanism for entity prediction. The ability to extend a blank model with custom components makes spaCy a versatile tool for various NLP tasks, including domain-specific NER.

6.2..4 Rasa DIET for Entity Recognition and Intent Classification:

Rasa is an open-source machine learning framework designed for building conversational AI systems. The Rasa DIET (Dual Intent and Entity Transformer) model is a sophisticated multilayer neural network that employs a combination of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). It is capable of handling both intent classification and entity recognition tasks. The model also offers the flexibility to incorporate various pre-trained embeddings like BERT, GloVe, and ConveRT.

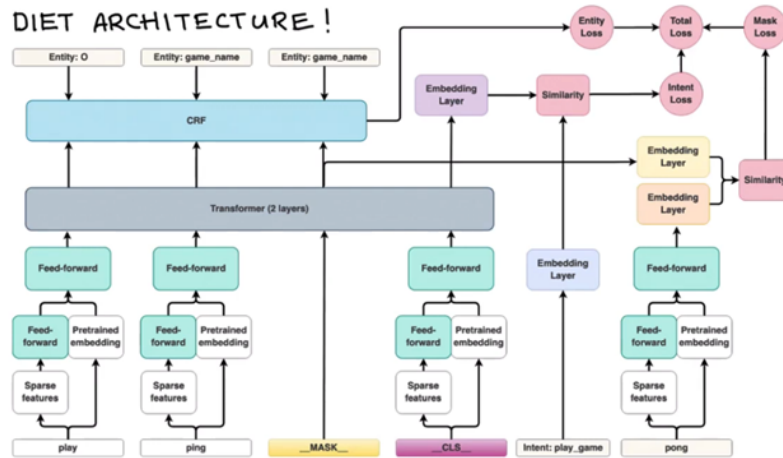


Figure 23: Model Architecture: Rasa DIET for Entity Recognition and Intent Classification

The DIET model is composed of several layers designed to perform specific tasks:

- **Embedding Layer:** This layer is responsible for converting tokens into dense vectors. It can utilize pre-trained embeddings like BERT or GloVe, or learn embeddings from scratch.
- **Convolutional Layer (CNN):** This layer applies convolutional filters to the embeddings to capture local dependencies among words.
- **Recurrent Layer (RNN):** This layer captures the sequential information in the text. It is usually implemented using LSTM or GRU cells.
- **Attention Mechanism:** This is used to weigh different parts of the input text differently, focusing on the most informative parts for the task at hand.
- **Fully Connected Layers:** These layers are used for the final classification tasks, both for intent and entities.
- **Masking:** Used to ignore certain parts of the input, such as padding, during the training process to improve model performance.
- **Entity Loss:** A loss function specifically designed to improve the model's performance on entity recognition tasks.
- **Similarity Measures:** Used in the loss function to measure how close the model's predictions are to the actual labels.

The Rasa DIET pipeline consists of several components, each serving a specific purpose:

- **WhitespaceTokenizer:** Splits the text into individual tokens.
- **RegexFeaturizer:** Generates features based on custom regular expressions.
- **LexicalSyntacticFeaturizer:** Creates features based on the position of the word, its part-of-speech (POS) tag, and other syntactic elements.
- **CountVectorsFeaturizer:** Produces bag-of-words representations for the tokens. The first instance operates at the word level, and the second at the character level (char_wb).
- **DIETClassifier:** Trained on both the intent labels and entities present in the training data. It takes the features produced by the preceding featurizers and generates contextualized word embeddings using a transformer architecture.

In summary, the Rasa DIET model is a robust and flexible architecture for conversational AI applications. It combines the strengths of CNNs and RNNs to effectively handle both intent classification and entity recognition. The pipeline is modular, allowing for easy customization and the integration of pre-trained embeddings for enhanced performance. Its multilayer architecture is designed to capture both local and global contextual information, making it highly effective for the tasks it is designed for.

6.2..5 Support Vector Machine (SVM) for Intent Classification:

Support Vector Machine (SVM)[45] is a supervised learning algorithm widely used for classification tasks, including text classification. In the context of intent classification, SVM works by transforming text data into a numerical representation and finding an optimal hyperplane that separates different intent classes. This section elaborates on the architecture and mathematical foundations of the SVM model used in our research.

The mathematical formulation of the SVM optimization problem for a binary classification can be expressed as:

$$\min_{\vec{w}, b} \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^N \xi_i$$

subject to:

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

where \vec{w} is the weight vector, b is the bias term, ξ_i are the slack variables, C is the regularization parameter, and y_i and \vec{x}_i are the label and feature vector of the i^{th} training example, respectively.

The core idea of SVM is to find a hyperplane that best separates the different classes in the feature space. Mathematically, a hyperplane is defined as:

$$\vec{w} \cdot \vec{x} - b = 0$$

where \vec{w} is the weight vector and b is the bias term. In a binary classification scenario, this hyperplane serves as the decision boundary between two classes. The objective is to find the optimal hyperplane that maximizes the margin, M , defined as:

$$M = \frac{2}{\|\vec{w}\|}$$

The margin is essentially the distance between the hyperplane and the nearest data points (or support vectors) from each class.

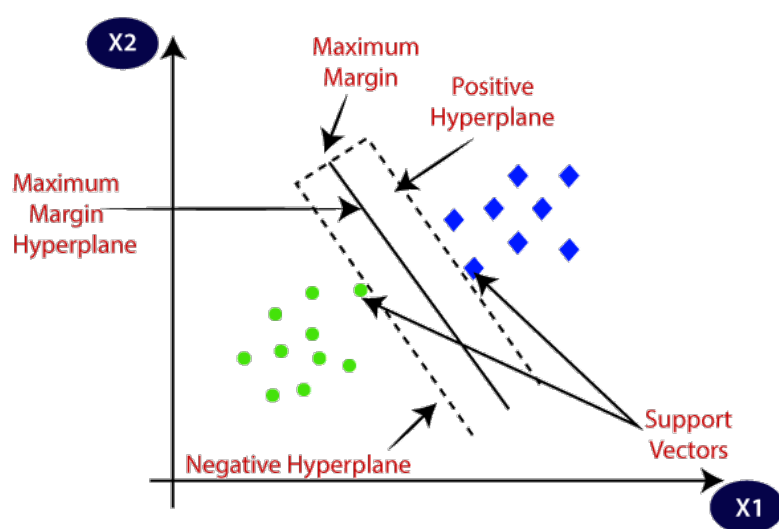


Figure 24: Model Architecture:Support Vector Machine (SVM) for Intent Classification

Support vectors are the data points that lie closest to the decision boundary or on the margin itself. These vectors are critical in defining the optimal hyperplane, as they are the most challenging points to classify. Mathematically, support vectors satisfy the condition $y_i(\vec{w} \cdot \vec{x}_i - b) = 1$ for correctly classified points, or $y_i(\vec{w} \cdot \vec{x}_i - b) < 1$ for misclassified points.

In cases where the data is not linearly separable, SVM utilizes a kernel function to transform the feature space into a higher-dimensional space where a separating hyperplane can be found. Commonly used kernel functions include polynomial kernels, radial basis function (RBF) kernels, and sigmoid kernels.

The model is trained using an optimization process that aims to maximize the margin while minimizing classification error. This is often formulated as a convex optimization problem, which can be solved using techniques like Sequential Minimal Optimization (SMO).

For multi-class problems, the One-vs-Rest (OvR) strategy is employed. For K classes, K binary classifiers are trained. The class corresponding to the classifier with the highest output is selected as the predicted class.

In summary, SVM for intent classification transforms text data into a numerical representation, identifies an optimal hyperplane using support vectors, and employs the kernel trick for handling nonlinear relationships. This makes it a robust and effective model for classifying intents in text data.

6.2..6 Naive Bayes for Intent Classification:

Naive Bayes[45] is a probabilistic classifier based on Bayes' theorem. It is particularly well-suited for text classification tasks, including multi-class problems. The model operates under the assumption that the features (words in the case of text) are conditionally independent given the class label.

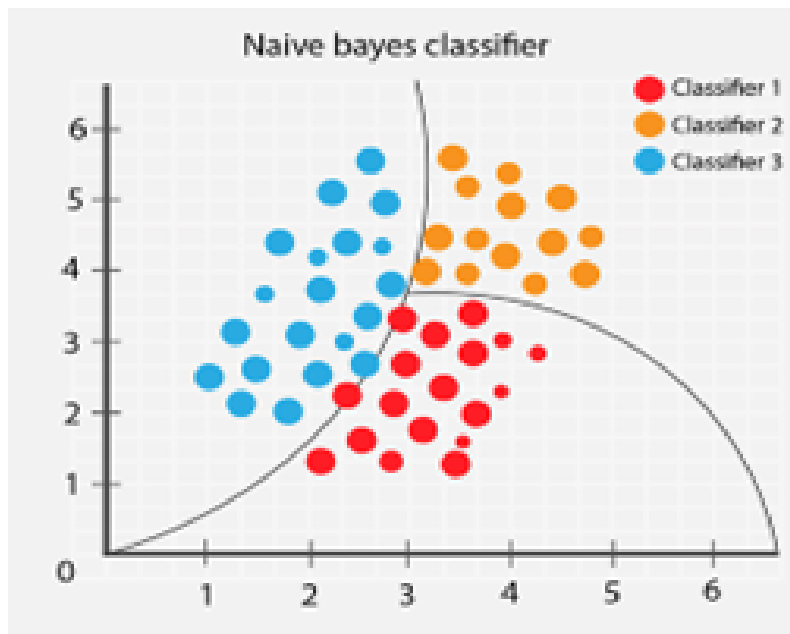


Figure 25: Naive Bayes for Intent Classification

The Naive Bayes classifier is based on Bayes' theorem, which can be expressed as:

$$P(C_k|\vec{x}) = \frac{P(\vec{x}|C_k) \cdot P(C_k)}{P(\vec{x})}$$

where C_k is the class label, and \vec{x} is the feature vector.

One of the key assumptions of Naive Bayes is that the features are conditionally independent

given the class label. Mathematically, this is expressed as:

$$P(\vec{x}|C_k) = \prod_{i=1}^n P(x_i|C_k)$$

where n is the number of features, and x_i is the i^{th} feature.

Naive Bayes estimates two types of probabilities: class prior probabilities $P(C_k)$ and class conditional probabilities $P(x_i|C_k)$. The class prior probabilities are simply the frequencies of each class in the training data. The class conditional probabilities are estimated as:

$$P(x_i|C_k) = \frac{\text{Count}(x_i, C_k)}{\text{Count}(C_k)}$$

where $\text{Count}(x_i, C_k)$ is the number of times feature x_i appears in samples of class C_k , and $\text{Count}(C_k)$ is the total count of all features in class C_k .

For a new document \vec{x} , the posterior probabilities for each class are calculated, and the class with the highest posterior probability is selected as the predicted label:

$$\hat{C} = \arg \max_k P(C_k|\vec{x})$$

In summary, the Naive Bayes model for multi-class text classification is a probabilistic approach based on Bayes' theorem. It assumes conditional independence among features and calculates class prior and conditional probabilities from the training data. The model is simple yet effective, making it a popular choice for text classification tasks.

6.2.7 Logistic Regression for Intent Classification:

Logistic Regression[45] is a supervised learning algorithm commonly used for binary and multi-class classification problems. In the context of intent classification, the model employs the logistic function to model the relationship between the features (words) and the target class labels.

The logistic function, also known as the sigmoid function, is mathematically defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z = \vec{w} \cdot \vec{x} + b$, \vec{w} is the weight vector, \vec{x} is the feature vector, and b is the bias term.

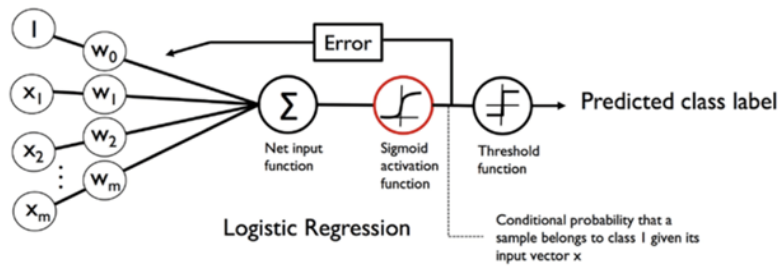


Figure 26: Logistic Regression for Intent Classification

Logistic Regression uses either maximum likelihood estimation or gradient descent to find the optimal weights. The objective function for maximum likelihood estimation is:

$$\mathcal{L}(\vec{w}, b) = \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where $\hat{y}_i = \sigma(\vec{w} \cdot \vec{x}_i + b)$ and y_i is the actual label of the i^{th} sample. The goal is to maximize this likelihood function.

To classify a new text sample, the model calculates the predicted probability \hat{y} using the logistic function. A threshold is then applied to \hat{y} to determine the final class label. For example, if \hat{y} is greater than 0.5, the sample is classified as positive; otherwise, it is classified as negative.

In summary, Logistic Regression is a robust and effective model for text classification. It uses the logistic function to model the probability of a particular class given the features. The model is trained using either maximum likelihood estimation or gradient descent to find the optimal weights. A threshold is applied to the predicted probabilities to make the final class assignment.

6.2..8 Decision Tree for Intent Classification:

Decision Trees[45] are supervised learning algorithms widely used for classification tasks. In the context of text classification, Decision Trees build a tree-like model of decisions based on the features extracted from the text data.

The algorithm starts with the entire training dataset at the root node. It selects the best feature to split the data based on a criterion such as information gain or Gini impurity. This selected feature becomes the decision criterion at that node, and the data is divided into subsets based on the feature values. The process is recursively applied to each subset.

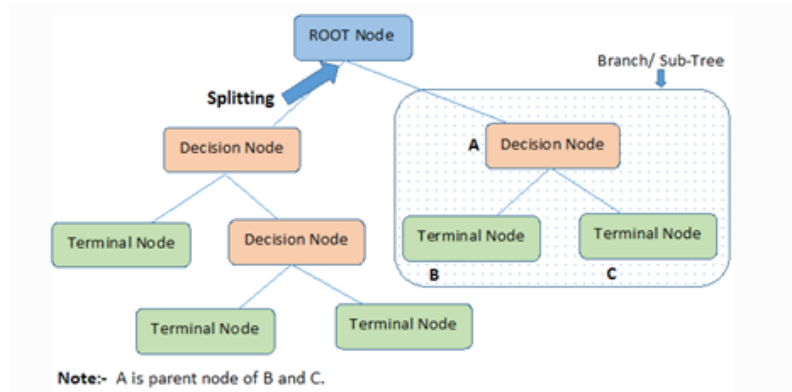


Figure 27: Decision Tree for Intent Classification

The algorithm selects the feature that maximizes the information gain or minimizes the Gini impurity. Mathematically, information gain IG is defined as:

$$IG(D, f) = H(D) - \sum_{v \in \text{Values}(f)} \frac{|D_v|}{|D|} H(D_v)$$

where $H(D)$ is the entropy of dataset D , and D_v is the subset of D for which feature f has value v .

The algorithm continues to split the data at each node until a stopping criterion is met. Common stopping criteria include reaching a maximum depth, reaching a minimum number of samples at a node, or no further improvement in impurity.

Once the Decision Tree is built, it can be used to classify new, unseen text samples. The sample traverses the Decision Tree, following the learned decision rules based on its feature values. At each node, the sample is directed to the appropriate child node until a leaf node is reached. The class label associated with that leaf node is then assigned to the sample.

In summary, Decision Trees are a robust and interpretable model for text classification. They build a hierarchical model of decisions based on feature values, using criteria like information gain or Gini impurity for splitting. The algorithm is straightforward yet effective, making it a popular choice for text classification tasks.

6.2..9 Random Forest for Intent Classification:

Random Forest[45] is an ensemble machine learning technique that builds upon Decision Trees. It is particularly effective for classification tasks, including text classification. The model aims to

improve the performance and robustness of a single Decision Tree by creating an ensemble of trees and aggregating their predictions.

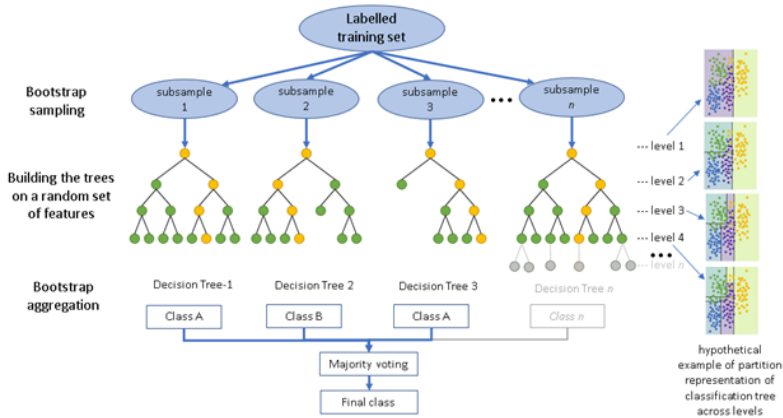


Figure 28: Random Forest for Intent Classification

One of the key techniques used in Random Forest is Bagging, or Bootstrap Aggregation. In this method, multiple data subsets are created from the original dataset by randomly selecting samples with replacement. A Decision Tree is trained on each of these subsets. Mathematically, if D is the original dataset, a subset D_i is created as:

$$D_i = \text{RandomSampleWithReplacement}(D)$$

To introduce randomness and reduce overfitting, Random Forest employs the Random Subspace Method. In this approach, a random subset of features is chosen at each node for making a decision. This ensures that each tree in the ensemble is slightly different, contributing to the model's robustness.

The final prediction of the Random Forest model is based on majority voting from all the Decision Trees in the ensemble. For a given sample, each tree in the ensemble makes a prediction, and the class label that receives the majority of votes is selected as the final prediction.

The final prediction \hat{y} is given by:

$$\hat{y} = \text{MajorityVote}(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n)$$

where \hat{y}_i is the prediction from the i^{th} Decision Tree in the ensemble.

In summary, Random Forest is an ensemble method that builds upon Decision Trees to create a more robust and accurate model. It employs techniques like Bagging and the Random Subspace Method to introduce randomness and reduce overfitting. The final prediction is made through majority voting, aggregating the wisdom of the ensemble for improved performance.

6.3. Experimental setup:

In this section, we describe the training and experimental configurations for the four different Named Entity Recognition algorithms used in this study: NLTK, Spacy, Stanford NER, and RASA DIET.

6.3.1 Experiment 1: Entity Recognition:

NLTK:

The NLTK model employs the MaxEnt classifier, which is based on the Generalized Iterative Scaling (GIS) algorithm. The feature extraction is done using an inbuilt feature extractor. The configuration parameters are as follows:

```
algorithm = 'GIS'
trace = 3
max_iter = 10
```

Here, ‘algorithm = ‘GIS’ specifies the use of the Generalized Iterative Scaling algorithm. The ‘trace=3’ parameter is used for logging verbosity, and ‘max_iter=10’ specifies the maximum number of iterations for the algorithm.

Spacy:

The Spacy model uses a Named Entity Recognition (NER) algorithm with a Transition-Based Parser. The vectorizer used is an inbuilt tok2vec model. The pipeline includes tokenizers and a named entity recognizer. The configuration parameters are:

```
Model = blank("en")
pipeline = tokenizers , 'ner '
iter = 50
drop = 0.5
batch_size = compounding(4.0, 16.0, 1.001)
```

In this configuration, ‘Model = blank("en)” initializes a blank English model. The pipeline includes tokenizers and a named entity recognizer (‘ner’). The ‘iter: 50’ specifies the number of iterations, and ‘drop: 0.5’ is the dropout rate to prevent overfitting. The batch size starts at 4 and increases to a maximum of 16.

Stanford NER:

The Stanford NER model uses a Conditional Random Field (CRF) algorithm with an inbuilt feature extractor. The configuration parameters are extensive and include:

```
useTypeSeqs2 = true
useWordPairs = true
maxNGramLeng = 10
maxLeft = 1
wordShape = chris2useLC
useWordTag = true
useDisjunctive = true
useOccurrencePatterns = true
useClassFeature = true
useNGrams = true
useNext = true
usePrev = true
usePrevSequences = true
map = word=0, answer=1
useWord = true
useShapeConjunctions = True
```

This configuration includes various boolean flags that enable or disable specific features in the CRF model, such as word pairs, n-grams, and disjunctive features. The ‘maxNGramLeng=10’ specifies the maximum length of n-grams to be considered.

Rasa DIET:

The Rasa DIET model employs the DIET Classifier with a pipeline that includes WhitespaceTokenizer, RegexFeaturizer, LexicalSyntacticFeaturizer, and CountVectorsFeaturizer. The configuration parameters are:

```
Language = 'en'
pipeline = analyzer: char_wb, min_ngram: 1, max_ngram: 4
DIETClassifier
epochs = 100
constrain_similarities = true
threshold = 0.3
ambiguity_threshold = 0.1
RegexEntityExtractor
```

Here, ‘Language = ‘en’ specifies that the language used is English. The ‘pipeline’ includes various tokenizers and featurizers. The ‘epochs: 100’ specifies the number of training epochs, and ‘constrain_similarities: true’ ensures that only meaningful similarities are considered.

Each of these models was trained using their respective configurations, and their performance was evaluated based on multiple metrics, as described in the Results section.

6.3..2 Experiment 2: Intent Classification:

In this section, we elaborate on the training configurations for various machine learning algorithms used for intent classification in this study: SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET.

SVM Classifier:

The SVM Classifier model is implemented using the Sklearn library. The configuration parameters are:

```
penalty = 'l2'
loss = 'squared_hinge'
C = 1.0
multi_class = 'ovr'
max_iter = 1000
```

Here, `penalty = 'l2'` specifies the L2 regularization, and `loss = 'squared_hinge'` sets the loss function. The `C = 1.0` parameter controls the margin hardness, and `multi_class = 'ovr'` specifies a one-vs-rest strategy. The `max_iter = 1000` sets the maximum number of iterations for the solver.

Logistic Regression:

The Logistic Regression model is also implemented using the Sklearn library. The configuration parameters are:

```
penalty = 'l1'
C = 1.0
solver = 'liblinear'
max_iter = 100
```

In this configuration, `'penalty = 'l1''` specifies the L1 regularization. The `'C = 1.0'` parameter controls the inverse of regularization strength, and `'solver = 'liblinear''` specifies the algorithm to use for optimization. The `'max_iter = 100'` sets the maximum number of iterations for the solver.

Naïve Bayes:

The Naïve Bayes model is implemented using the Sklearn library. The configuration parameters are:

```
priors = None
var_smoothing = 1e-09
```

Here, `'priors = None'` means that no prior probabilities are provided, and `'var_smoothing = 1e-09'` specifies the portion of the largest variance to be added to variances for calculation stability.

Decision Tree:

The Decision Tree model is implemented using the Sklearn library. The configuration parameters are:

```
ccp_alpha = 0.0
criterion = 'gini'
max_depth = None
max_features = None
max_leaf_nodes = None
min_samples_leaf = 1
min_samples_split = 2
```

This configuration includes various parameters like `'ccp_alpha'` for cost complexity pruning and `'criterion = 'gini''` for the function to measure the quality of a split.

Random Forest:

The Random Forest model is implemented using the Sklearn library. The configuration parameters are:

```
n_estimators = 100
criterion = 'gini'
min_samples_split = 2
```

```

min_samples_leaf = 1
bootstrap = True
ccp_alpha = 0.0
max_samples = None

```

Here, ‘n_estimators = 100’ specifies the number of trees in the forest. The ‘criterion = ‘gini’‘ is used for the quality of a split, and ‘min_samples_split = 2’ specifies the minimum number of samples required to split an internal node.

RASA DIET:

The RASA DIET model employs the DIET Classifier. The configuration parameters are:

```

epochs = 100
constrain_similarities = true
min_ngram = 1
max_ngram = 4

```

Here, ‘epochs = 100’ specifies the number of training epochs. The ‘constrain_similarities = true’ ensures that only meaningful similarities are considered during training.

6.4. Evaluation:

6.4.1 Performance Metrics:

Named Entity Recognition (NER) is assessed by contrasting the model’s predictions with a set of gold standard annotations. Two primary methods for this evaluation are *exactmatch evaluation* and *relaxed match evaluation*. In the former, a span is considered correct only if it perfectly aligns with the gold standard. In the latter, the entity is deemed correct if its type is accurate and it overlaps with the gold standard, even if the boundaries are not exact. In our work, we use the exact-match method for its evaluation metrics.

The evaluation metrics of Precision, Recall, and F-score are calculated using True Positives (TP), False Positives (FP), and False Negatives (FN). TP represents the entities correctly identified by the model and aligned with the gold standard. FP signifies the entities identified by the model but not present in the gold standard. FN denotes the entities present in the gold standard but missed by the model.

- **Precision:** Precision quantifies the model’s capability to correctly identify only the positive samples while avoiding the misclassification of negative samples. Essentially, the focus of precision is to reduce the number of false positives.
- **Recall:** Recall addresses the question: Of the instances that are genuinely positive, how many did the model fail to identify? In essence, recall aims to minimize the occurrence of false negatives.
- **F1-Score:** The F1score serves as the harmonic mean of precision and recall. It provides a balanced measure that considers both the false positives and false negatives, offering a more comprehensive view of the model’s performance.

The mathematical representations of these metrics are as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5)$$

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6)$$

Generally, Entity Recognition and Intent Classification are treated as a multiclass classification problem, and the F-score is commonly used as the evaluation metric. It is often calculated using both micro-averaging and macro-averaging techniques. Thus, in our work, the F1-score, which is the harmonic mean of precision and recall, is used as the evaluation metric to balance between precision and recall.

6.4.2 System Metrics:

In addition to the traditional evaluation metrics like Precision, Recall, and F1-Score, the performance of the model is also assessed using system metrics. These metrics provide insights into the computational efficiency and resource utilization of the model, which are crucial for real-world deployments.

- **Computation Time:** This metric measures the amount of time required for the model to complete its task, from data ingestion to output generation. Lower computation time is generally preferred as it indicates a more efficient model.
- **CPU Usage:** CPU usage quantifies the percentage of the central processing unit’s capacity that is being utilized by the model. High CPU usage may indicate that the model is computationally intensive and could benefit from optimization.
- **Memory Usage:** Memory usage gauges the amount of RAM consumed by the model during its operation. Efficient memory usage is essential for ensuring that the model can run on systems with limited resources.
- **GPU Usage:** For models that are optimized to run on Graphics Processing Units (GPUs), GPU usage measures the percentage of the GPU’s computational power being utilized. This metric is particularly relevant for deep learning models, which often require significant computational resources. In the context of our work, we also conduct an evaluation to determine whether the selected models utilized the capabilities of the Graphics Processing Unit (GPU) for optimal performance. If GPU usage is required, we further assess the extent to which the GPU’s computational resources are utilized.

The results of this evaluation contribute to a comprehensive understanding of the model’s performance and system metrics, encompassing both its predictive capability and its computational efficiency.

6.5. Results:

This section provides insights into the model’s performance, focusing on its predictive ability on the test dataset and the system metrics during the training phase.

6.5.1 Overall Comparison of Performance Metrics:

In this section, we provide a comprehensive overview of the performance metrics attained by four distinct Named Entity Recognition (NER) models: NLTK, StanfordNER, Spacy, and RASA DIET, as well as six Intent Classification algorithms—SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET.

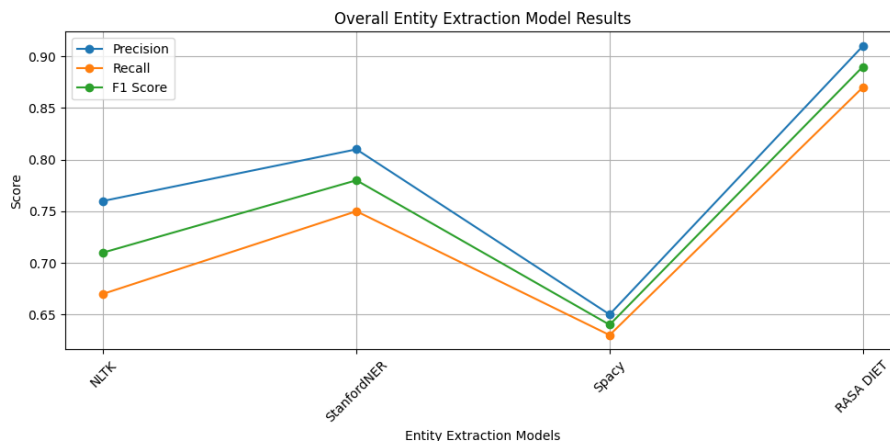


Figure 29: Entity Extraction: Overall Comparison of Performance Metrics

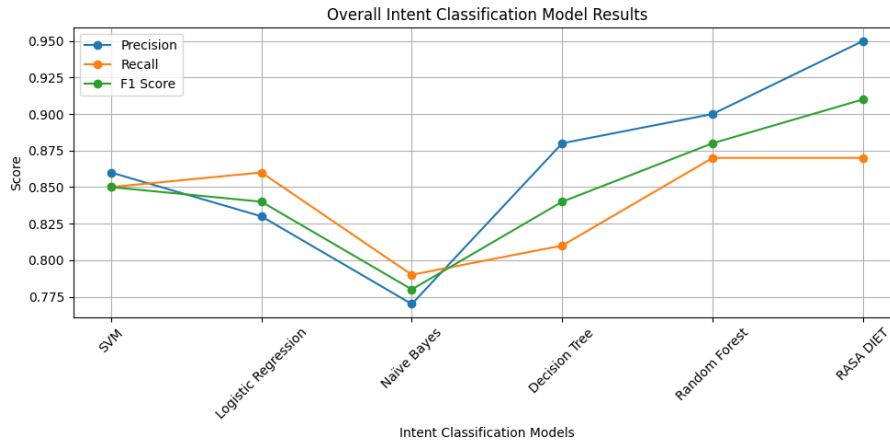


Figure 30: Intent Classification: Overall Comparison of Performance Metrics

It is evident from the results that the precision, recall, and F1 score achieved by the "RASA DIET" model are notably higher compared to the other models and "RASA DIET" model consistently outperforms the other models for both tasks. On the other hand, the "NLTK" model generally exhibits lower scores, indicating comparatively weaker performance. Additionally, it is worth noting that while the performance of the "Spacy" and "StanfordNER" models for entity extraction and "Random Forest" and "Decision Tree" models for intent classification is comparable, they fall short compared to the "RASA DIET" model.

This holistic comparison provided us with a general comprehension of the model's performance. However, it didn't provide detailed insights into the models' individual capabilities concerning entity-wise and intent-wise tasks. Therefore, to achieve a more granular understanding, it became essential to evaluate their performance on specific entity types and intent labels. This focused assessment will allow us to unravel nuances that might not have been apparent in the overall comparison, enabling us to make more informed conclusions about each model's strengths and weaknesses in handling distinct entities and intents.

6.5..2 Entity-wise Comparison of Entity Recognition Models:

We present the performance metrics of four different Named Entity Recognition (NER) models: NLTK, StanfordNER, Spacy, and RASA DIET. The metrics include Precision, Recall, and F1 Score for various entity types. The models were tested on various entity types such as location, quality of service (QoS) constraints, group, hour, QoS metric, service, traffic, QoS value, operation, middlebox, and QoS unit.

Table 5: NLTK Model Results

NLTK			
Entity Type	Precision	Recall	F1 Score
location	0.58	0.83	0.68
qos_constraint	1.00	0.40	0.57
group	1.00	0.45	0.62
hour	1.00	1.00	1.00
qos_metric	0.83	0.62	0.71
service	1.00	0.25	0.40
traffic	0.50	0.11	0.18
qos_value	1.00	0.75	0.86
operation	0.92	0.73	0.81
middlebox	0.60	0.27	0.37
qos_unit	1.00	0.88	0.93

Table 6: StanfordNER Model Results

StanfordNER			
Entity Type	Precision	Recall	F1 Score
location	0.81	0.90	0.85
qos_constraint	0.87	1.00	0.93
group	0.83	0.62	0.71
hour	1.00	1.00	1.00
qos_metric	0.80	0.66	0.72
service	1.00	0.25	0.40
traffic	0.60	0.60	0.60
qos_value	1.00	1.00	1.00
operation	0.76	0.76	0.76
middlebox	1.00	0.75	0.85
qos_unit	0.88	0.88	0.88

Table 7: Spacy Model Results

Spacy			
Entity Type	Precision	Recall	F1 Score
location	0.61	0.48	0.54
qos_constraint	0.66	0.90	0.76
group	0.84	0.64	0.73
hour	0.27	0.60	0.37
qos_metric	0.50	0.31	0.38
service	0.80	0.44	0.57
traffic	0.42	0.71	0.53
qos_value	0.55	0.90	0.69
operation	0.90	0.77	0.83
middlebox	0.62	0.62	0.62
qos_unit	0.72	0.90	0.79

Table 8: RASA DIET Model Results

RASA DIET			
Entity Type	Precision	Recall	F1 Score
location	0.97	0.99	0.98
qos_constraint	0.78	1.0	0.87
group	0.96	1.0	0.98
hour	1.0	0.87	0.93
qos_metric	0.91	0.92	0.92
service	0.96	0.96	0.96
traffic	0.95	0.80	0.87
qos_value	0.88	1.0	0.93
operation	0.90	0.99	0.94
middlebox	1.0	0.96	0.98
qos_unit	1.0	0.97	0.98

The NLTK model showed a high F1 Score for entity types like ‘hour’ (1.00) and ‘qos_unit’ (0.93). However, it performed poorly on ‘traffic’ with an F1 Score of 0.18. The model had a high precision but low recall for entities like ‘qos_constraint’ and ‘service,’ indicating that while the identified entities are accurate, many were missed.

StanfordNER demonstrated robust performance across multiple entities. It achieved an F1 Score of 1.00 for ‘hour’ and ‘qos_value’. However, the model had a relatively low F1 Score for ‘service’ (0.40), despite having a high precision of 1.00, indicating low recall.

The Spacy model had a balanced performance with an F1 Score of 0.79 for ‘qos_unit’ and 0.76 for ‘qos_constraint’. However, it failed to properly identify entities of type ‘hour,’ and ‘qos_metric’ resulting in an F1 Score of 0.37 and 0.38 respectively. The model showed high recall but low precision for ‘traffic’ and ‘qos_value,’ suggesting that it may be generating some false positives.

The RASA DIET model outperformed all other models with consistently high F1 Scores across most entity types. It achieved a perfect F1 Score of 0.98 for ‘location’, ‘group’, ‘middlebox’ and ‘qos_unit’. The model also had high precision and recall, making it the most reliable among the four models evaluated.

6.5..3 Intent-wise Comparison of Intent Classification Models:

The performance of six different Intent Classification algorithms—SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET—was evaluated using three key metrics: Precision, Recall, and F1-Score. The algorithms were tested on various intent labels such as add, allow, block, limit, and unblock.

Table 9: SVM Classifier

SVM Classifier			
Intent Label	Precision	Recall	F1-Score
add	0.76	0.95	0.84
allow	0.75	0.67	0.71
block	1.00	0.83	0.91
limit	0.92	0.73	0.81
unblock	1.00	1.00	1.00

Table 10: Logistic Regression

Logistic Regression			
Intent Label	Precision	Recall	F1-Score
add	0.78	0.95	0.86
allow	0.80	0.44	0.57
block	0.86	1.00	0.92
limit	0.79	0.73	0.76
unblock	1.00	1.00	1.00

Table 11: Naïve Bayes

Naïve Bayes			
Intent Label	Precision	Recall	F1-Score
add	0.86	0.90	0.88
allow	0.57	0.44	0.50
block	0.50	0.67	0.57
limit	0.93	0.87	0.90
unblock	1.00	1.00	1.00

Table 12: Decision Tree

Decision Tree			
Intent Label	Precision	Recall	F1-Score
add	0.90	0.90	0.90
allow	0.57	0.89	0.70
block	1.00	0.83	0.91
limit	0.91	0.67	0.77
unblock	1.00	1.00	1.00

Table 13: Random Forest

Random Forest			
Intent Label	Precision	Recall	F1-Score
add	0.74	1.00	0.85
allow	0.83	0.56	0.67
block	1.00	0.83	0.91
limit	0.92	0.73	0.81
unblock	1.00	1.00	1.00

Table 14: RASA DIET

RASA DIET			
Intent Label	Precision	Recall	F1-Score
add	1.0	1.0	1.0
allow	1.0	0.56	0.67
block	0.92	1.0	0.96
limit	1.0	0.94	0.97
unblock	1.00	1.00	1.00

The SVM Classifier showed a high F1-Score for ‘unblock’ (1.00) and ‘block’ (0.91). However, it had a relatively lower F1-Score for ‘allow’ (0.71), despite having a high precision of 0.75.

Logistic Regression achieved high F1-Scores for ‘add’ (0.86) and ‘block’ (0.92). However, it had a low F1-Score for ‘allow’ (0.57), indicating that the model may have difficulty in accurately classifying this particular intent.

Naïve Bayes had a balanced performance with high F1-Scores for ‘add’ (0.88) and ‘limit’ (0.90). However, it had a low F1-Score for ‘allow’ (0.50), suggesting that the model may be generating some false positives or negatives for this intent.

The Decision Tree model showed robust performance with high F1-Scores for ‘add’ (0.90) and ‘block’ (0.91). It also had a high F1-Score for ‘unblock’ (1.00), making it reliable for these intents.

Random Forest had high F1-Scores for ‘add’ (0.85) and ‘block’ (0.91). It also achieved a perfect F1-Score for ‘unblock’ (1.00), indicating high precision and recall.

The RASA DIET model outperformed all other models with consistently high F1-Scores across

all intent labels. It achieved perfect F1-Scores for ‘add’ (1.0), ‘block’ (0.96), and ‘unblock’ (1.00), making it the most reliable among the six models evaluated.

6.5..4 Comparison of models based on system metrics:

The performance of four different Named Entity Recognition algorithms—NLTK, Spacy, Stanford NER, and RASA DIET and six different Intent Classification algorithms—SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET—was evaluated using three key metrics: Computing Time, CPU Usage, and Memory Usage.

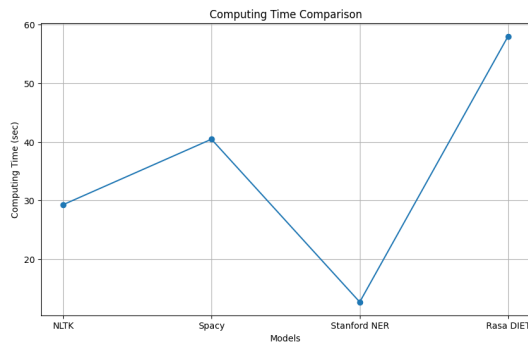


Figure 31: Computing Time Comparison for Entity Extraction Models

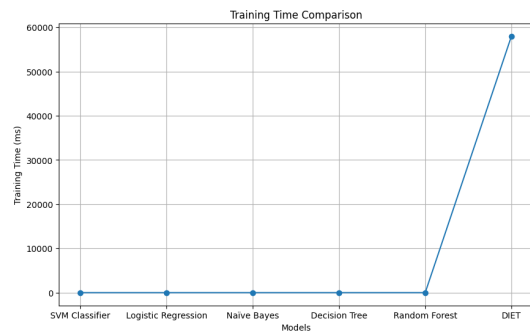


Figure 32: Computing Time Comparison for Intent Classification Models

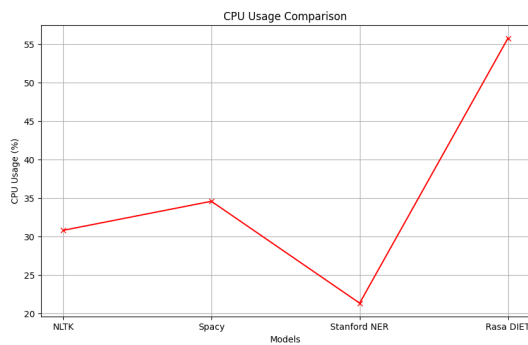


Figure 33: CPU_Usage Comparison for Entity Extraction Models

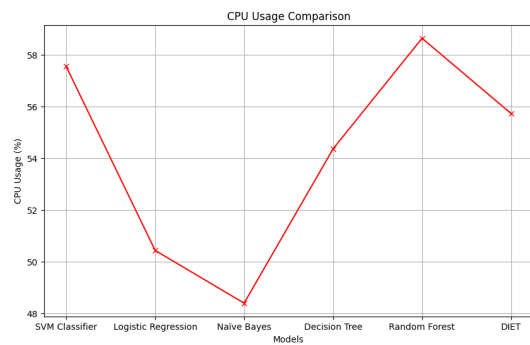


Figure 34: CPU_Usage Comparison for Intent Classification Models

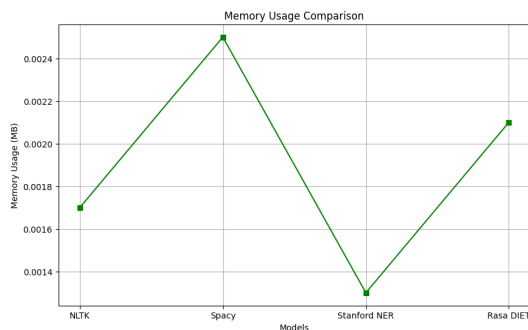


Figure 35: Memory_Usage Comparison for Entity Extraction Models

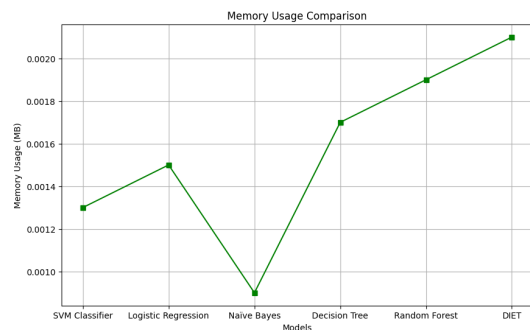


Figure 36: Memory_Usage Comparison for Intent Classification Models

For the entity recognition task, the RASA DIET model had the highest computing time of 58 seconds, followed by Spacy at 40.46 seconds. Stanford NER had the lowest computing time of 12.7 seconds. The high computing time for RASA DIET and Spacy can be attributed to their more complex architectures and additional training epochs.

Similarly, for the Intent recognition task, the DIET model had a significantly higher training time of 58000 ms, which is orders of magnitude higher than the other models. Among the other models, Random Forest had the highest training time of 5.12 ms, followed closely by Decision Tree at 4.49 ms. Logistic Regression had the lowest training time of 3.58 ms. It is important to note that the DIET model’s training time is considerably higher due to its more complex architecture and additional training epochs.

For the entity recognition task, the RASA DIET model had the highest CPU usage at 55.73%, which is considerably higher than the other models. Stanford NER had the lowest CPU usage at 21.35%. Despite its high computing time, Spacy had a moderate CPU usage of 34.57%.

In terms of CPU usage for the intent recognition task, the Random Forest model had the highest CPU usage at 58.64%, followed by the SVM Classifier at 57.55%. The Naïve Bayes model had the lowest CPU usage at 48.39%. Interestingly, despite its high training time, the DIET model had a moderate CPU usage of 55.73%, which is likely optimized for longer training cycles.

Similarly, for the entity recognition task, in terms of memory usage, Spacy consumed the most memory at 0.0025 MB, followed by RASA DIET at 0.0021 MB. NLTK had the lowest memory footprint at 0.0013 MB.

In the meanwhile, for the Intent recognition task, in terms of memory usage, the DIET model consumed the most memory at 0.0021 MB. The Random Forest model was a close second at 0.0019 MB. Naïve Bayes had the lowest memory footprint at 0.0009 MB. The memory usage generally correlates with the complexity of the models, with more complex models requiring more memory.

Additionally, none of our models utilized GPU resources, thus obviating the need to assess GPU usage.

7. Discussions:

The results presented in the previous section offer a comprehensive analysis of the performance, system metrics, and implications of various Named Entity Recognition (NER) and Intent Classification models. These findings provide valuable insights into the strengths, weaknesses, and potential improvements of each model, facilitating advancements in natural language interfaces for improved developer experiences and broader applications.

To fully appreciate the outcomes of our experimental findings, this section discusses and reviews the findings to reconnect them to the research questions we posed at the onset of this study.

- **RQ1: In the broader landscape of AI conversational assistants, which NLP algorithms, especially concerning entity extraction and intent classification, are prominently discussed?**

In addressing this question, our investigation delved into prevailing literature and discerned a range of algorithms that are widely discussed in the context of AI conversational systems. While many preceding investigations regarding entity and intent categorization in the realm of natural language interfaces lean heavily towards deep learning approaches, our exploration veered towards simpler algorithms. Notably, prior studies in the network-related domain often zoomed in on a singular model, predominantly the deep learning-based RASA. However, our research took a broader lens, examining multiple models. Through our literature review, we identified and evaluated four distinct entity recognition methods: MaxEnt Classifier with NLTK, Spacy, Conditional Random Fields with Stanford NER, and RASA DIET. Likewise, for intent classification, six distinct algorithms were analyzed: SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET. The examination of these models gives a comprehensive overview of the prevalent techniques in this space. The selection of these models was primarily based on their popularity and frequent mentions in previous research, coupled with their applicability to the domain of network management. With the variety of models evaluated, it is evident that while some models are general-purpose and applicable in diverse scenarios, others are optimized for specialized tasks or

environments. This diverse assessment underscores the importance of a model’s contextual relevance, particularly within the network management domain.

Given the specific use case of Nokia and the challenges associated with seamlessly integrating a natural language interface without compromising system agility and performance, this research fills a significant gap in the existing literature. Unlike previous studies that primarily assessed the performance of NLP algorithms in the network management domain based solely on performance metrics, this research goes a step further by incorporating system metrics, which are paramount in evaluating the feasibility of these models in real-world scenarios.

- **RQ2: Upon evaluation, how effective and efficient are these NLP models selected from the literature review in terms of system metrics and performance metrics?**

The assessment of NER models, including NLTK, StanfordNER, Spacy, and RASA DIET, revealed diverse performances across different entity types. The RASA DIET model demonstrated remarkable effectiveness by achieving the highest average F1 Score, indicating its ability to accurately identify a wide array of entity types. In contrast, NLTK exhibited high precision but suffered from low recall in certain categories, impacting its overall F1 Score. StanfordNER and Spacy displayed competitive performance, excelling or lagging in specific areas. StanfordNER faced difficulties in effectively identifying ‘service’ entities, while Spacy showcased balanced performance, except for recognizing ‘hour’ entities. Interestingly, these variations in performance were also reflected in the system metrics.

Within the realm of Intent Classification, a comparison between SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, and RASA DIET highlighted distinct performance trends. RASA DIET consistently outperformed other models, demonstrating high precision, recall, and F1-Scores across various intent labels. Notably, its superior precision and recall characteristics established RASA DIET as the most reliable model for this task. However, SVM Classifier and Logistic Regression models struggled with low recall for the ‘allow’ intent, impacting their F1-Scores. Similarly, Naïve Bayes and Decision Tree models exhibited limitations in accurately classifying the ‘allow’ intent. The Random Forest model displayed competitive performance, albeit with slightly lower F1-Scores for ‘allow’ and ‘limit’ intents. Notably, these differences in performance also correlated with the computational demands of the models.

When examining system metrics, it became evident that different models vary significantly in terms of computational requirements. The superior performance of RASA DIET came at the cost of increased computational resources, making it suitable for applications without strict resource constraints. Conversely, Spacy’s high computing time and memory usage could limit its feasibility in resource-constrained environments. Stanford NER demonstrated efficient computing time but with lower CPU usage. NLTK stood out for its low memory usage, rendering it suitable for lightweight applications.

The interplay between system metrics and model performance highlights the trade-offs that developers must consider. High-performing models, like RASA DIET, while offering accurate entity and intent recognition, demand more computational resources. This intricate balance between system metrics and performance metrics aligns with our findings that a model’s appropriateness is often dictated by the application’s specific constraints and requirements. Spacy and StanfordNER serve as examples of models that strike a balance between performance and resource usage for entity extraction tasks while Random Forest and Decision Tree exhibit a similar balance for intent classification. This observation is particularly noteworthy since most studies in the past focused solely on performance metrics. Our inclusion of system metrics offers a more holistic view, allowing for informed decisions that consider not just the accuracy but also the practicality of deploying these models in real-world systems, especially in scenarios similar to Nokia’s low-code/no-code environment.

The study, while conducted with a focus on the network management domain, covered algorithms and techniques that are widely used across various domains. The nature of Named Entity Recognition (NER) and Intent Classification is such that while training data may differ, the underlying techniques can often be adapted across industries and use cases. As a result, the

insights obtained on performance and efficiency can be of relevance to other sectors, albeit with necessary adjustments and fine-tuning.

This research’s broader implications, beyond Nokia’s specific case, provide a template for evaluating NLP algorithms for their suitability in other low-code/no-code platforms, which are inherently lightweight. The findings underscore the need for a holistic assessment of NLP algorithms, encompassing both their efficacy in task execution and their compatibility with the overarching system they are intended to be integrated with, especially where decisions related to computational costs versus performance trade-offs are important.

In summary, the observed results underscore the importance of employing a diverse range of NLP algorithms to enhance the accuracy and efficiency of NLP applications. The identified challenges in entity and intent recognition offer directions for future research. Models could be fine-tuned to improve performance on specific entity types or intent labels that proved challenging. Given the variability in model performance across different entity types and intent labels, there’s an opportunity to optimize results by combining models via an ensemble method. As demonstrated, some models excel in recognizing specific entities, while others perform better for different entities. By strategically amalgamating these models, their individual strengths can be leveraged for a more comprehensive entity extraction process. Similarly, in intent classification, models proficient in specific intent labels can be combined to improve overall accuracy. Additionally, expanding the study to larger and more diverse datasets would validate the generalizability of these models across various scenarios. The observed trade-off between performance and computational resources emphasizes the need to develop lightweight yet accurate models, especially for resource-constrained environments.

8. Limitations:

As is the case with any scholarly investigation, the present study is not without its limitations, which frame the scope and interpretability of the research findings. While this work offers a comprehensive exploration of various NLP algorithms for entity extraction and intent classification, as well as a comparative analysis of their efficiency in implementing a Natural Language Interface within a low-code/no-code software system, several constraints inevitably bound the investigation.

One notable limitation pertains to data availability. The low-code/no-code system developed by Nokia, which is the subject of this study, is still in its developmental stage and consequently lacks an adequate dataset for robust model training. Furthermore, due to confidentiality clauses, we were restricted from utilizing Nokia’s proprietary data for our work. To address this challenge, we collected publicly available network-related data from online sources. However, given the limited research conducted in this specific domain and for our particular use case, the amount of available online data was scarce. The data acquired from the Lumi Nile dataset was also insufficient in terms of size. When attempting to augment this dataset through synthetic means, we were unable to generate high-quality data, thereby leaving us with a final dataset that is still small in size.

Additionally, the obtained data was not pre-annotated to meet the specific requirements of our model, necessitating a time-consuming manual annotation process. The limitations imposed by the small and manually annotated dataset could compromise the generalizability of our findings.

Another constraint lies in the unavailability of pre-trained models suitable for our specific use case. This absence precluded us from harnessing the benefits of transfer learning for model fine-tuning. Lastly, the scope of this research was intentionally narrowed to focus on simpler NLP algorithms, thereby restricting us from exploring the potential advantages of more complex deep learning models.

9. Conclusion and Future Work

The principal objective of this thesis was to investigate the efficiency and effectiveness of various Natural Language Processing (NLP) algorithms in implementing Natural Language Interfaces (NLIs) to improve developer experience in low-code/no-code software systems, with specific application to a system developed by Nokia Corporation. This study pursued a comparative analysis of multiple NLP algorithms mainly for Entity Recognition (ER) and Intent Classification tasks,

emphasizing their performance, and computational requirements to understand their suitability for this specific use-case. Based on the understanding from the literature review, we selected four different Entity Recognition techniques: MaxEnt Classifier with NLTK, Spacy, Conditional Random Fields with Stanford NER, and RASA DIET, and six different algorithms for the task of intent classification: SVM Classifier, Logistic Regression, Naïve Bayes, Decision Tree, and Random Forest, along with RASA DIET. For training the model, network-related utterances were used as training data which was further annotated and processed to meet the training requirements of each model. The model, once trained, was subsequently assessed using the test dataset to gauge both its predictive accuracy and overall system performance. A thorough comparative analysis was performed based on the results of the model evaluation to provide Nokia with insights into the available models, their efficiency and effectiveness and the suitability of their application on Nokia's system.

Our empirical evaluations revealed the trade-offs that must be navigated when selecting an NLP algorithm for practical implementation. For example, the RASA DIET model, while excelling in terms of accuracy metrics, demands a higher computational load. Other models like Spacy and StanfordNER demonstrated balanced performance but varied in their strengths and weaknesses for specific entity types. In the realm of intent classification, models like Random Forest and Decision Tree showed potential despite certain limitations. These findings are of particular significance for industry applications, especially for companies like Nokia Corporation, where the implementation choices can have far-reaching implications for both user experience and operational overhead.

The comprehensive understanding offered by this thesis contributes to the scholarly dialogue on the adaptability and utility of NLP algorithms in low-code/no-code platforms. It provides an empirically-grounded foundation upon which practitioners can make informed decisions that take into account not just algorithmic performance but also computational efficiency. These findings inspire further research, enabling the development of more accurate and efficient natural language interfaces that enhance the developer experience and find applications in a broader spectrum of domains.

Several directions for future research emerge from this study. Firstly, the variability in algorithmic performance across different entity types and intent labels suggests that a hybrid or ensemble approach, leveraging the strengths of multiple algorithms, could offer improved overall performance. Further, fine-tuning the existing models for specific challenges encountered in our tests, such as low recall rates for certain entity types or intents, could also lead to performance improvements. Secondly, as our study was limited by the availability and size of the dataset, investigating the performance of these models on a more substantial and domain-specific corpus would add considerable value. The limited availability of domain-specific, annotated datasets for training and validation presents an opportunity for the creation and public release of such resources to further accelerate research in this area. Lastly, given that our work does not extend to integrating these algorithms into Nokia's existing system, future research could focus on the practical implementation aspects, studying the real-world applicability and scalability of these NLP algorithms within low-code/no-code platforms. Undertaking a user-centric study involving the actual users of these low-code/no-code platforms could provide additional insights into the real-world usability and effectiveness of implemented NLP algorithms.

Further research should also explore the potential advantages of more complex deep-learning models, and their feasibility in terms of computational cost and ease of integration, to paint a more comprehensive picture of the options available for implementing NLIs in low-code/no-code systems.

10. Acknowledgement:

This thesis would not have been possible without the help of many people in so many ways. I extend my heartfelt thanks to my mentor at Nokia, Mayank Darbari, for giving me such a great opportunity to work on this interesting thesis topic and for his constant encouragement, advice, and support throughout the journey. I am equally grateful towards my University supervisor, Prof. Wasif Afzal for providing me with constructive feedback, and guidance throughout my research project.

I'd also like to extend my gratitude to Nokia Corporation for providing me with the chance to be a part of their esteemed team. Special thanks to my manager, Tommi Lundell, for providing a supportive and efficient work environment.

A heartfelt appreciation to the Erasmus Mundus Association and EDISS for bestowing upon me the scholarship that enabled this academic endeavour. I am also eternally thankful to Jan Carlson and Radu Dobrin, my program coordinators at MDU University, for their guidance at every phase of my academic journey.

Finally, I would like to thank my family and friends for their continuous support and encouragement.

References

- [1] T. Beranic, P. Rek and M. Heričko, ‘Adoption and usability of low-code/no-code development tools,’ in *Central European Conference on Information and Intelligent Systems*, Faculty of Organization and Informatics Varazdin, 2020, pp. 97–103.
- [2] K. Rokis and M. Kirikova, ‘Challenges of low-code/no-code software development: A literature review,’ in *International Conference on Business Informatics Research*, Springer, 2022, pp. 3–17.
- [3] G. G. Hendrix, ‘Natural-language interface,’ *American Journal of Computational Linguistics*, vol. 8, no. 2, pp. 56–61, 1982.
- [4] B. AbuShawar and E. Atwell, ‘Usefulness, localizability, humanness, and language-benefit: Additional evaluation criteria for natural language dialogue systems,’ *International Journal of Speech Technology*, vol. 19, no. 2, pp. 373–383, 2016.
- [5] P. M. Nadkarni, L. Ohno-Machado and W. W. Chapman, ‘Natural language processing: An introduction,’ *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 2011.
- [6] S. S. Balgasem and L. Q. Zakaria, ‘A hybrid method of rule-based approach and statistical measures for recognizing narrators name in hadith,’ in *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, IEEE, 2017, pp. 1–5.
- [7] H. Ketmaneechairat and M. Maliyaem, ‘Natural language processing for disaster management using conditional random fields,’ *Journal of Advances in Information Technology*, vol. 11, no. 2, 2020.
- [8] W. Khan, A. Daud, J. A. Nasir and T. Amjad, ‘A survey on the state-of-the-art machine learning models in the context of nlp,’ *Kuwait journal of Science*, vol. 43, no. 4, 2016.
- [9] O. Etzioni, M. Cafarella, D. Downey *et al.*, ‘Unsupervised named-entity extraction from the web: An experimental study,’ *Artificial intelligence*, vol. 165, no. 1, pp. 91–134, 2005.
- [10] J. Schuurmans and F. Frasincar, ‘Intent classification for dialogue utterances,’ *IEEE Intelligent Systems*, vol. 35, no. 1, pp. 82–88, 2019.
- [11] M. Arevalillo-Herráez, P. Arnau-González and N. Ramzan, ‘On adapting the diet architecture and the rasa conversational toolkit for the sentiment analysis task,’ *IEEE Access*, vol. 10, pp. 107 477–107 487, 2022.
- [12] P. Sun, X. Yang, X. Zhao and Z. Wang, ‘An overview of named entity recognition,’ in *2018 International Conference on Asian Language Processing (IALP)*, IEEE, 2018, pp. 273–278.
- [13] G. Kaur, ‘Usage of regular expressions in nlp,’ *International Journal of Research in Engineering and Technology IJERT*, vol. 3, no. 01, p. 7, 2014.
- [14] P. Blunsom, ‘Hidden markov models,’ *Lecture notes, August*, vol. 15, no. 18-19, p. 48, 2004.
- [15] A. Ekbal and S. Bandyopadhyay, ‘Bengali named entity recognition using support vector machine,’ in *Proceedings of the IJCNLP-08 workshop on named entity recognition for south and south east Asian Languages*, 2008.
- [16] Y. Goldberg, ‘Neural network methods for natural language processing,’ *Synthesis lectures on human language technologies*, vol. 10, no. 1, pp. 1–309, 2017.
- [17] *Brat rapid annotation tool*. [Online]. Available: <https://brat.nlplab.org/index.html> (visited on 30/08/2023).
- [18] *Prodigy · Prodigy · An annotation tool for AI, Machine Learning & NLP*, en. [Online]. Available: <https://prodi.gy/> (visited on 30/08/2023).
- [19] *NER Annotator for SpaCy*. [Online]. Available: <https://tecoholic.github.io/ner-annotator/> (visited on 30/08/2023).
- [20] T. Pranckevičius and V. Marcinkevičius, ‘Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification,’ *Baltic Journal of Modern Computing*, vol. 5, no. 2, p. 221, 2017.

- [21] S. R. Gomes, S. G. Saroar, M. Mosfaiul *et al.*, ‘A comparative approach to email classification using naive bayes classifier and hidden markov model,’ in *2017 4th international conference on advances in Electrical Engineering (ICAEE)*, IEEE, 2017, pp. 482–487.
- [22] K. M. Tarwani and S. Edem, ‘Survey on recurrent neural network in natural language processing,’ *Int. J. Eng. Trends Technol*, vol. 48, no. 6, pp. 301–304, 2017.
- [23] D. Britz, ‘Understanding convolutional neural networks for nlp,’ URL: <http://www.wildml.com/2015/11/understanding-convolutional-neuralnetworks-for-nlp/> (visited on 11/07/2015), 2015.
- [24] F. Almeida and G. Xexéo, *Word Embeddings: A Survey*, arXiv:1901.09069 [cs, stat], May 2023. DOI: [10.48550/arXiv.1901.09069](https://doi.org/10.48550/arXiv.1901.09069). [Online]. Available: <http://arxiv.org/abs/1901.09069> (visited on 30/08/2023).
- [25] T. Young, D. Hazarika, S. Poria and E. Cambria, ‘Recent Trends in Deep Learning Based Natural Language Processing [Review Article],’ *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, Aug. 2018, Conference Name: IEEE Computational Intelligence Magazine, ISSN: 1556-6048. DOI: [10.1109/MCI.2018.2840738](https://doi.org/10.1109/MCI.2018.2840738).
- [26] *Word2Vec | Natural Language Engineering | Cambridge Core*. [Online]. Available: <https://www.cambridge.org/core/journals/natural-language-engineering/article/word2vec/B84AE4446BD47F48847B4904F0B36E0B> (visited on 30/08/2023).
- [27] L.-P. Jing, H.-K. Huang and H.-B. Shi, ‘Improved feature selection approach TFIDF in text mining,’ in *Proceedings. International Conference on Machine Learning and Cybernetics*, vol. 2, Nov. 2002, 944–946 vol.2. DOI: [10.1109/ICMLC.2002.1174522](https://doi.org/10.1109/ICMLC.2002.1174522).
- [28] J. Pennington, R. Socher and C. Manning, ‘GloVe: Global Vectors for Word Representation,’ in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: [10.3115/v1/D14-1162](https://doi.org/10.3115/v1/D14-1162). [Online]. Available: <https://aclanthology.org/D14-1162> (visited on 30/08/2023).
- [29] Y. Vasiliev, *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020.
- [30] *StanfordNLP 0.2.0 - Python NLP Library for Many Human Languages | StanfordNLP*. [Online]. Available: <https://stanfordnlp.github.io/stanfordnlp/> (visited on 30/08/2023).
- [31] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, ‘Scikit-learn: Machine learning in python,’ *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [32] *NLTK :: Natural Language Toolkit*. [Online]. Available: <https://www.nltk.org/> (visited on 30/08/2023).
- [33] *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. [Online]. Available: <https://dspace.mit.edu/handle/1721.1/7095> (visited on 30/08/2023).
- [34] J. Weizenbaum, ‘ELIZA—a computer program for the study of natural language communication between man and machine,’ en, *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, Jan. 1966, ISSN: 0001-0782, 1557-7317. DOI: [10.1145/365153.365168](https://doi.org/10.1145/365153.365168). [Online]. Available: <https://dl.acm.org/doi/10.1145/365153.365168> (visited on 30/08/2023).
- [35] *Computational Intelligence in Conversational UI, A BotLibre Case Study. A survey paper / Engineering Archive*. [Online]. Available: <https://engrxiv.org/preprint/view/633/> (visited on 30/08/2023).
- [36] A. Chernyavskiy, D. Ilvovsky and P. Nakov, *Transformers: "The End of History" for NLP?* arXiv:2105.00813 [cs], Sep. 2021. DOI: [10.48550/arXiv.2105.00813](https://doi.org/10.48550/arXiv.2105.00813). [Online]. Available: <http://arxiv.org/abs/2105.00813> (visited on 30/08/2023).
- [37] R. Lair, ‘Introduction to visual studio 2008,’ in *Beginning Silverlight 3*, Springer, 2009, pp. 13–37.
- [38] H. Zhang, A. Jain, G. Khandelwal, C. Kaushik, S. Ge and W. Hu, ‘Bing developer assistant: Improving developer productivity by recommending sample code,’ in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 956–961.

- [39] L. Floridi and M. Chiriatti, ‘Gpt-3: Its nature, scope, limits, and consequences,’ *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.
- [40] O. A. L. Lemos, S. K. Bajracharya, J. Ossher *et al.*, ‘Codegenie: Using test-cases to search and reuse source code,’ in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 525–526.
- [41] J. Czerwonka, M. Greiler, C. Bird, L. Panjer and T. Coatta, ‘Codeflow: Improving the code review process at microsoft: A discussion with jacek czerwonka, michaela greiler, christian bird, lucas panjer, and terry coatta,’ *Queue*, vol. 16, no. 5, pp. 81–100, 2018.
- [42] H. Kim, Y. Jiang, S. Kannan, S. Oh and P. Viswanath, ‘Deepcode: Feedback codes via deep learning,’ *Advances in neural information processing systems*, vol. 31, 2018.
- [43] R. D. Amori, ‘A natural language interface for task oriented activities,’ in *Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems-Volume 1*, 1990, pp. 553–562.
- [44] A. Sampaio, R. Chitchyan, A. Rashid and P. Rayson, ‘Ea-miner: A tool for automating aspect-oriented requirements identification,’ in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 352–355.
- [45] S. Wang, F. Song, Q. Qiao, Y. Liu, J. Chen and J. Ma, ‘A comparative study of natural language processing algorithms based on cities changing diabetes vulnerability data,’ *Healthcare*, vol. 10, p. 1119, Jun. 2022. DOI: [10.3390/healthcare10061119](https://doi.org/10.3390/healthcare10061119).
- [46] J.-H. Kim and P. C. Woodland, ‘A rule-based named entity recognition system for speech input,’ in *Sixth International Conference on Spoken Language Processing*, 2000.
- [47] K. Han, J. Chen, H. Zhang *et al.*, ‘Delta: A deep learning based language technology platform,’ *arXiv preprint arXiv:1908.01853*, 2019.
- [48] T. Gui, X. Wang, Q. Zhang *et al.*, ‘Textflint: Unified multilingual robustness evaluation toolkit for natural language processing,’ *arXiv preprint arXiv:2103.11441*, 2021.
- [49] J. Xu, Y. Zhang, J. Wang *et al.*, ‘Uth-ccb: The participation of the semeval 2015 challenge–task 14,’ in *proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 2015, pp. 311–314.
- [50] J. Pilault, A. Elhattami and C. Pal, ‘Conditionally adaptive multi-task learning: Improving transfer learning in nlp using fewer parameters & less data,’ *arXiv preprint arXiv:2009.09139*, 2020.
- [51] C. Di Sipio, D. Di Ruscio and P. T. Nguyen, ‘Democratizing the development of recommender systems by means of low-code platforms,’ in *Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings*, 2020, pp. 1–9.
- [52] R. Ramachandran and K. Arutchelvan, ‘Named entity recognition on bio-medical literature documents using hybrid based approach,’ *J Ambient Intell Humaniz Comput*, pp. 1–10, Mar. 2021, ISSN: 1868-5137. DOI: [10.1007/s12652-021-03078-z](https://doi.org/10.1007/s12652-021-03078-z). [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7947151/> (visited on 30/08/2023).
- [53] K. P. Johnson, P. J. Burns, J. Stewart, T. Cook, C. Besnier and W. J. B. Mattingly, ‘The Classical Language Toolkit: An NLP Framework for Pre-Modern Languages,’ in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Aug. 2021, pp. 20–29. DOI: [10.18653/v1/2021.acl-demo.3](https://doi.org/10.18653/v1/2021.acl-demo.3). [Online]. Available: <https://aclanthology.org/2021.acl-demo.3> (visited on 30/08/2023).
- [54] S. Vychezhnanin and E. Kotelnikov, ‘Comparison of Named Entity Recognition Tools Applied to News Articles,’ Dec. 2019, pp. 72–77. DOI: [10.1109/ISPRAS47671.2019.00017](https://doi.org/10.1109/ISPRAS47671.2019.00017).
- [55] R. Sharma, ‘An Analytical Study and Review of open source Chatbot framework, Rasa,’ *International Journal of Engineering Research and*, vol. V9, Jun. 2020. DOI: [10.17577/IJERTV9IS060723](https://doi.org/10.17577/IJERTV9IS060723).

- [56] J. W. Creswell and J. D. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.
- [57] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey and S. Linkman, ‘Systematic literature reviews in software engineering—a systematic literature review,’ *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.
- [58] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [59] D. Meurers, ‘Natural language processing and language learning,’ *Encyclopedia of applied linguistics*, pp. 4193–4205, 2012.
- [60] S. García, J. Luengo and F. Herrera, *Data preprocessing in data mining*. Springer, 2015, vol. 72.
- [61] J. Ruppenhofer, M. Ellsworth, M. Schwarzer-Petruck, C. R. Johnson and J. Scheffczyk, ‘Framenet ii: Extended theory and practice,’ International Computer Science Institute, Tech. Rep., 2016.
- [62] K. Cho, B. Van Merriënboer, C. Gulcehre *et al.*, ‘Learning phrase representations using rnn encoder-decoder for statistical machine translation,’ *arXiv preprint arXiv:1406.1078*, 2014.
- [63] R. Kohavi *et al.*, ‘A study of cross-validation and bootstrap for accuracy estimation and model selection,’ in *Ijcai*, Montreal, Canada, vol. 14, 1995, pp. 1137–1145.
- [64] J. Han, J. Pei and H. Tong, *Data mining: concepts and techniques*. Morgan kaufmann, 2022.
- [65] M. Sokolova and G. Lapalme, ‘A systematic analysis of performance measures for classification tasks,’ *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.
- [66] *Chatbot/nlu-engine/data at master · lumichatbot/chatbot*, en. [Online]. Available: <https://github.com/lumichatbot/chatbot/tree/master/nlu-engine/data> (visited on 30/08/2023).
- [67] A. S. Jacobs, R. J. Pfitscher, R. H. Ribeiro *et al.*, ‘Hey, Lumi! Using Natural Language for {Intent-Based} Network Management,’ en, 2021, pp. 625–639, ISBN: 978-1-939133-23-6. [Online]. Available: <https://www.usenix.org/conference/atc21/presentation/jacobs> (visited on 30/08/2023).
- [68] *Gretel.ai — The synthetic data platform for developers*. [Online]. Available: https://gretel.ai/?kw=gretel%20ai&cpn=19630461345&gclid=Cj0KCQjw0bunBhD9ARIsAAZ10E1a13W_a7NZpgpmHjkmP1agbQzz8onVOM1ezL9t4jDL8b81EvFqD1EaAtv0EALw_wcB (visited on 30/08/2023).
- [69] P. Tum, *NLP: Text Segmentation Using Maximum Entropy Markov Model*, en, Jan. 2020. [Online]. Available: <https://medium.com/@phylypo/nlp-text-segmentation-using-maximum-entropy-markov-model-c6160b13b248> (visited on 30/08/2023).