UTRECHT UNIVERSITY

Department of Information and Computing Science

Applied Data Science Master's Thesis

Named Entity Recognition and Relation Extraction with a Transformer-based Model

First examiner:
Mel W. Chekol

Second examiner:
Duygu S. Islakoğlu

Candidate:
Natalia Kurdanova

In cooperation with:
Knights Advanced Analytics

July 4, 2024

Abstract

One of the tasks of natural language processing (NLP) is information extraction, which aims to transform unstructured text data into structured information. Key components of this task include Named Entity Recognition (NER) and Relation Extraction (RE), which focus on the identification and classification of entities and the relations between them within the text. The areas of application for NER and RE models include constructing knowledge graphs and supporting applications like machine translation and automated question-answering systems [1], [2]. In recent years, deep learning models, including transformers, have achieved state-of-the-art performance in NER and RE tasks.

This thesis examines the performance of the transformer-based model LUKE (Language Understanding with Knowledge-based Embeddings) [3] for NER and RE tasks. We benchmark LUKE against the spaCy[1] solution for NER and the REBEL (Relation Extraction By End-to-End Language Generation) [4] model for RE using a manually labeled dataset of 335 news articles.

Here we show that while LUKE demonstrates strong performance in NER, it is outperformed by the fine-tuned spaCy model on our specific dataset. For RE tasks, LUKE's relation classifier shows outstanding performance, but the sequential LUKE-based RE pipeline does not match the performance of the end-to-end REBEL model. These results provide insights into the strengths and limitations of LUKE, guiding future efforts in model selection.

[1]`https://spacy.io/`

Contents

1. Introduction

In the fast-evolving field of natural language processing (NLP), extracting meaningful information from unstructured text is one of the key tasks. Industries and companies generate and receive vast amounts of unstructured text data daily, including emails, reports, customer reviews, and news articles. Efficiently processing and extracting valuable insights from this data can enhance decision-making, improve customer service, and streamline operations. For example, online news platforms can automatically tag articles with relevant keywords extracted from the texts, thereby enhancing content availability and improving the overall reading experience. These benefits of information extraction motivate companies and research institutions to develop and implement various methods aimed at transforming unstructured text data into structured information. One of the recently emerged approaches is the use of models based on transformer architecture, which will be discussed in detail later. This study, conducted in collaboration with Knights Advanced Analytics, aims to evaluate the performance of a transformer-based model for information extraction tasks using a dataset of online news articles provided by the company. The main goal of the company is to determine which model is more effective and can be further used for constructing a knowledge graph supporting an automated question-answering system.

Proceeding to the details of the information extraction task, it encompasses several subtasks, among which Named Entity Recognition (NER) and Relation Extraction (RE). NER focuses on identifying and classifying entities such as persons, organizations, and locations within a text. For example, in the sentence *"Apple Inc. was founded by Steve Jobs."* NER would identify *"Apple Inc."* as an organization and *"Steve Jobs"* as a person. RE goes a step further, aiming to identify and classify the relationships between these entities. For instance, it would identify the relationship $founded\_by$ between *"Apple Inc."* and *"Steve Jobs."* NER and RE are essential for tasks like building knowledge graphs, which organize information in a computer-processable form and act as a backbone for various applications including search engines [5]. Furthermore, together, NER and RE models underpin a wide range of solutions, from automated question answering to sophisticated text summarization and machine translation systems [1], [2].

While various solutions are available for NER and RE tasks, deep learning has emerged as a popular approach due to its greater performance compared to traditional methods [1], [2]. Unlike traditional methods such as rule-based approaches and supervised learning, deep learning models learn important patterns and features directly from data, reducing the need for manually designed features. A separate class of deep learning solutions is transformer-based models. These models utilize a self-attention mechanism, allowing them to capture dependencies between words in a input sequence more efficiently compared to, for example, recurrent neural networks (RNN) [6]. This capability helps transformer-based solutions achieve great performance in tasks such as NER and RE that require understanding the relations between entities in a large text corpora.

However, one of the main disadvantages of most transformer-based models for NER and RE is that they consider the entities on the word level, splitting them into multiple parts. As a result, these models may struggle to detect entities and relationships between them when entities consist of multiple words [3]. LUKE (Language Understanding with Knowledge-based Embeddings) [3] is a transformer based model that aims to solve this issue. It was specifically developed for entity-related tasks and its main distinction from other transformer-based models is that it treats not only words but also entities as tokens. As a result, LUKE is better suited for the detection of entities and relationships between them [3]. The underlying mechanisms of these models are further discussed in Section 2.

This thesis aims to compare LUKE with baseline models, specifically focusing on LUKE's performance in NER and RE tasks. For NER, we compare LUKE with the baseline spaCy[1] solution, while for RE, we compare LUKE with the REBEL (Relation Extraction By End-to-End Language Generation) [4] model. The models' performance is evaluated using a manually labeled dataset of 335 news articles. By evaluating the models, this study seeks to provide insights into the models' strengths and limitations, answering the following research questions:

- RQ1: How effective is the LUKE model for NER compared to the baseline spaCy solution?
- RQ2: How effective is fine-tuning the LUKE model for NER?
- RQ3: How effective is fine-tuning the LUKE model for RE compared to the REBEL model?

To answer these questions, we conducted a series of experiments using the dataset provided by Knights Advanced Analytics. To answer the first research question, we compared the performance of the LUKE model with the spaCy solution for NER without prior fine-tuning on the manually labeled dataset. To answer the second research question, we fine-tuned the models based on the labeled dataset and then compared their performance. Finally, to answer the last question, we fine-tuned and implemented a LUKE pipeline solution that includes first NER using LUKE and then relation classification between the identified entities, also using LUKE. We compared the results of the pipeline solution to the fine-tuned REBEL model, which performs the RE task directly. For the evaluation metrics, we used macro precision, recall, F1-score, and class-specific precision, recall, and F1-score.

The findings reveal that while LUKE demonstrates strong inherent performance in

[1]`https://spacy.io/`

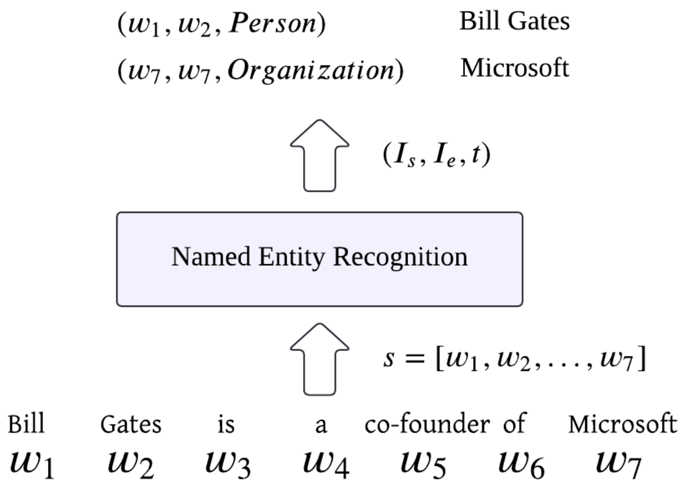
NER, the baseline spaCy model outperforms it when fine-tuned on our specific dataset due to its better ability to learn from a relatively small dataset. For RE tasks, the LUKE relation classifier excelled, but the sequential LUKE-based RE pipeline was less effective than the end-to-end REBEL model, highlighting the challenges in integrating NER and relation classification tasks in a pipeline approach. These results provide valuable insights into the strengths and limitations of the LUKE model for information extraction, guiding future efforts in model selection.

2. Related Work and Background

This section summarizes previous work and different techniques relevant to this research. Firstly, it provides an overview of named entity recognition (NER) and relation extraction (RE) tasks, including their goals, applications, and techniques. Secondly, it focuses on a transformer, which is a deep learning architecture that form the backbone of various machine learning models, including LUKE [3]. Finally, this section covers the LUKE model in greater detail.

2.1 Named Entity Recognition and Relation Extraction

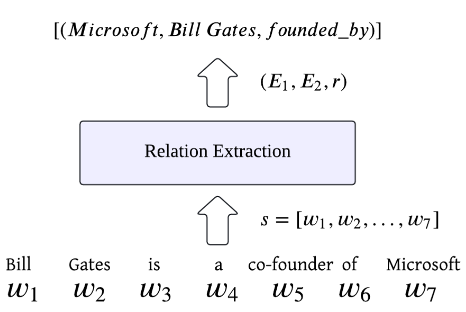
Information extraction is a task in natural language processing (NLP) that aims to transform unstructured or semi-structured text data into structured information, making it easier to analyze and understand [7]. Two key sub-tasks of information extraction are NER and RE. NER focuses on identifying and classifying named entities (NE) in the text into predefined categories. Named entities refer to specific items mentioned in the text that have unique attributes, such as persons, locations, and organizations in general contexts, or genes and diseases in specialized domains [1].



$(w_1, w_2, Person)$
Bill Gates
$(w_7, w_7, Organization)$
Microsoft
(I_s, I_e, t)
Named Entity Recognition
$s = [w_1, w_2, \ldots, w_7]$
Bill Gates is a co-founder of Microsoft
w_1 w_2 w_3 w_4 w_5 w_6 w_7

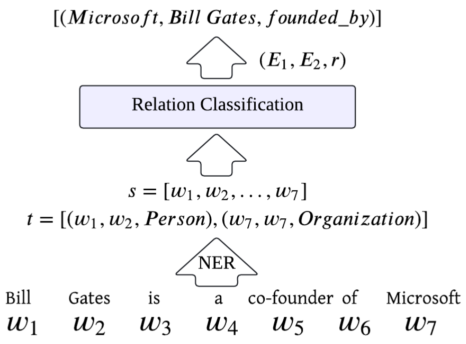
Figure 2.1: Named Entity Recognition task. The input of the NER model is a sequence of n tokens (encoded words or their parts) $s = [w_1, w_2, \ldots, w_n]$, and the output is a list of tuples (I_s, I_e, t), where I_s and I_e are the start and end indexes of the named entity, and t is the entity type.

Figure 2.1 provides an example of how an NER system extracts entities and their types from a given sentence. For example, if we consider the sentence *"Bill Gates is a co-founder* *of Microsoft"*, the input is a sequence of tokens that correspond to encoded words or their parts, while the output is the entities, for instance, *"Bill Gates"* of type *Person*.



$[(Microsoft, Bill\ Gates, founded\_by)]$
(E_1, E_2, r)
Relation Extraction
$s = [w_1, w_2, \ldots, w_7]$
Bill Gates is a co-founder of Microsoft
w_1 w_2 w_3 w_4 w_5 w_6 w_7

(a) Relation Extraction task. The input of the RE model is a sequence of n tokens (encoded words or their parts) $s = [w_1, w_2, \ldots, w_n]$. The output of the RE model is a list of tuples (E_1, E_2, r), where E_1 and E_2 are the first and second entities that participate in the relation, and r is the type of the relation.



$[(Microsoft, Bill\ Gates, founded\_by)]$
(E_1, E_2, r)
Relation Classification
$s = [w_1, w_2, \ldots, w_7]$
$t = [(w_1, w_2, Person), (w_7, w_7, Organization)]$
NER
Bill Gates is a co-founder of Microsoft
w_1 w_2 w_3 w_4 w_5 w_6 w_7

(b) Pipeline-based RE. The input of the RE model is a sequence of n tokens (encoded words or their parts) $s = [w_1, w_2, \ldots, w_n]$. It is passed to the NER, which detects the entities. Then the relation classifier predicts the relation type for each entity pair and outputs a list of tuples (E_1, E_2, r).

Figure 2.2: Relation Extraction task. (Left) Seq2Seq-based RE approach. (Right) The pipeline-based RE approach.

In contrast, the RE task is more complex as it involves not only the detection of entities but also the relationships between them. Figure 2.2a provides an example of how an RE system extracts relations from a given sentence. The input of the RE model is a sequence of tokens (encoded words or their parts). As an output, the RE model returns a $\langle head\_entity,\ tail\_entity,\ relationship \rangle$ triplet, which consists of two entities and the relation between them, such as the relation between *"Bill Gates"* and *"Microsoft"* of type $founded\_by$. Both NER and RE are foundational for numerous downstream applications, including question answering, text summarization, and machine translation [1], [2].

Li et al. [1] categorize **NER techniques** into several groups, including rule-based, feature-based supervised learning, and deep learning (DL). Rule-based approaches utilize handcrafted rules derived from domain-specific knowledge. For instance, a rule-based system might employ a predefined synonym dictionary to identify NEs [8]. However, these systems require significant domain expertise to construct comprehensive rules and are difficult to adapt to new domains. Additionally, they often yield high precision but low recall due to incomplete dictionaries, resulting in many false negatives. The second group of NER techniques is feature-based supervised learning, which can be framed as either multi-class classification or sequence labeling tasks. It requires annotated datasets with additional features for model training. These features might include word-level attributes like part-of-speech tags or morphology, and algorithms such as Decision Trees and Support Vector Machines are applied based on these features.

In contrast, **deep learning techniques for NER** have gained prominence in recent years. These methods automatically learn hidden features and patterns from raw data, significantly reducing the need for extensive manual feature crafting compared to rule-based and feature-based methods. According to Li et al. [1], the general architecture of a deep learning NER system consists of distributed representations of input, context encoder, and tag decoder. Firstly, distributed representations of input are embeddings of words or characters of an input sequence. Usually, embeddings are pre-trained over a large corpus. The embeddings aim to capture the syntactic and semantic properties of words. One of the popular algorithms for generating word embeddings is Word2Vec [9]. This model aims to learn the representation of words in a vector space in such a way that similar words are located closer together. Secondly, the context encoder is a neural network such as convolutional neural networks (CNN), recurrent neural networks (RNN), or transformers. The context encoder receives distributed representations as an input and creates context encoding by capturing the contextualized information. While distributed representation of input aims to learn word-level features, the context encoder learns sentence-level features. Finally, the token decoder classifies each context representation of tokens into the corresponding named entity category.

Similarly, Wang et al. [2] explore different **methods for RE**, which often parallel the techniques used for NER. They classify RE methods into several groups that include hand-built pattern methods and distant supervision methods as traditional approaches. They also consider methods based on deep neural networks (DNN) separately. Hand-built pattern RE methods involve matching preprocessed language fragments with patterns based on words, semantics, or parts of speech, recognizing a relation only if the text matches the pattern. As with NER, hand-built pattern RE methods also need extensive preliminary rule crafting. Distant supervision methods address the challenge of creating large labeled training datasets by using knowledge bases like Freebase to infer relationships between entities in text. This method assumes that if a sentence contains a pair of entities that also participate in the knowledge base relation, then this sentence is likely to express this relation as well [10]. Nevertheless, distant supervision can result in wrong labeling due to violation of this assumption.

Conversely, Wang et al. [2] highlight the advantages of **DNN-based methods in RE**, similar to their impact on NER. DNN-based methods can automatically learn hidden features, reducing the reliance on domain knowledge and manual effort. DNN-based methods outperform traditional approaches as they do not rely on the pre-defined patterns or datasets that limit the performance of the model and its ability to detect unseen relations. Similar to NER, the general architecture of an RE system comprises three parts: text representation, context encoding, and triplet prediction [11]. The part of the architecture that significantly differs from the NER system is triplet prediction. Its main goal is to detect the spans of the entities and predict the relation type for the pairs of identified entities.

Zhao et al. [11] distinguish two main categories of triplet prediction approaches for RE: the pipeline-based and joint approaches. The pipeline-based RE performs the task in two separate stages. First, it extracts entities from the input sequence and then performs relation classification by predicting the relation type for each possible pair. Figure 2.2b illustrates an example of pipeline-based RE. Firstly, the NER model detects all entities in the sequence and then the relation classifier predicts the relation type for all possible pairs. The main disadvantage of this approach is that it can suffer from error propagation, when errors from NER are passed to the relation classifier, worsening the performance of the whole RE pipeline. On the contrary, joint RE approaches do not suffer from this problem as they perform NER and relation classification at the same time. One of these approaches is Seq2Seq-based RE, which receives a sequence of tokens as an input and directly generates triples with entity pairs and relation types as shown in the Figure 2.2a.

2.2 Transformer

As previously mentioned, apart from traditional methods for NER and RE, deep learning algorithms are widely used because of their significant performance across many applications. In this section, we consider them in greater detail, giving special attention to the transformer architecture on which the LUKE model is based.

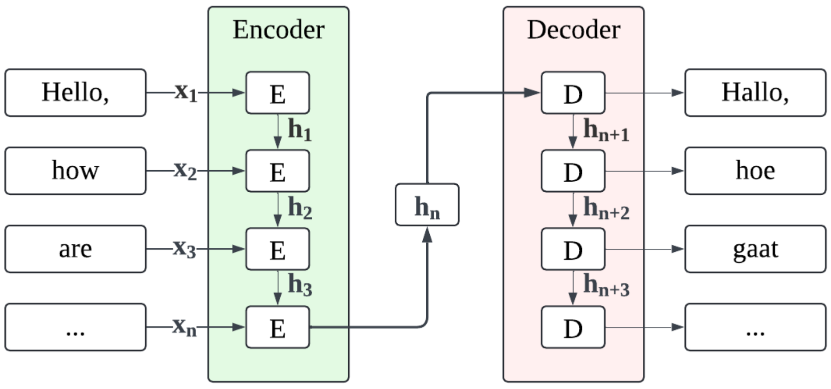


Figure 2.3: 'Unrolled' Recurrent Neural Network. The figure illustrates how the RNN processes a sequence of n tokens $X = [x_1, x_2, \ldots, x_n]$. The sequence is fed sequentially through the encoder network E, which receives x_i as an input. As output, the encoder generates the hidden state h_i. Then, the encoder passes the hidden state back through the feedback loop along with the next input x_{i+1}. Once all the tokens are processed by the encoder E, the last hidden state is passed to the decoder D. The decoder outputs tokens sequentially, also passing the hidden state through the feedback loop.

Before transformers, **Recurrent Neural Networks** (RNNs) were the state-of-the-art for modeling sequential data, used in tasks such as translation and language modeling [6], [12]. The main feature of RNNs is that this architecture contains a feedback loop that allows it to transfer information from one step to another. Thus, the model tracks information from previous steps and uses it for predictions. Figure 2.3 illustrates an example of how an RNN model is used for a translation task. The model consists of two main components: the **encoder**, which creates a numerical representation of the input sequence, and the **decoder**, which receives the encoded input and generates an output sequence [12]. For example, to translate the sentence *"Hello, how are you?"* into Dutch using an RNN, we pass each token (encoded word or its part) to the model one by one. Then, the encoder generates the last hidden state vector and passes it to the decoder, which produces the Dutch translation *"Hallo, hoe gaat het?"*. The main weakness of that approach is that the last hidden state produced by the encoder creates a bottleneck, as it should represent the whole input sequence. This might result in the loss of some information, especially if the input is long [12]. Moreover, because of the sequential nature of the network, parallel processing is not possible [6].

Attention is a mechanism designed to improve the performance of RNNs by allowing them to focus on different parts of the input sequence when generating each element of the output [13]. In the traditional RNN approach, the encoder passes only the final hidden state to the decoder, which can cause the model to lose important information from earlier in the sequence. With attention, instead of relying solely on the final hidden state, the encoder provides an extended representation of the input. This allows the model to pay attention to various parts of the input for each word produced by the decoder.

Transformer, proposed in the paper "Attention is All You Need" by Vaswani et al. [6], improves the network described before even further. Similar to RNN, a transformer is based on an encoder-decoder structure. However, instead of recurrent units, it has feed-forward networks and self-attention layers. Thus, the encoder is composed of a sequence of identical layers that consist of a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The self-attention mechanism in the encoder allows it to focus on different parts of the input sequence when encoding a token, grasping the dependencies regardless of the tokens' positions. The output of the encoder is a set of contextualized embeddings (vectors). The decoder also consists of a stack of identical layers, but with a third additional component: multi-head attention over the output passed from the encoder. The decoder generates an output sequence one token at a time [6]. The main benefit of the transformer architecture is that it allows parallel processing of input, as the initial sequence is fed all at once through the model. Moreover, the transformer architecture has greater performance compared to RNN when it comes to long input sequences, as it can attend to different parts of the input sequence based on their relevance to each other and to the current output being generated [12].

Initially, the transformer architecture was developed for sequence-to-sequence tasks like translation. However, both the encoder and decoder can be used separately. Decoder-only models, such as GPT (Generative Pre-trained Transformer) [14], are used for text-generation tasks such as language modeling and consist of only a decoder that generates the output tokens. Encoder-only models, such as BERT (Bidirectional Encoder Representations from Transformers) [15], convert an input text into a numerical representation which can be used for named entity recognition [12]. Thus, further, we are going to focus on encoder-only models that are applicable for the NER and RE tasks.

2.3 BERT and RoBERTa

When it comes to deep learning solutions for NER and RE, one of the greatest challenges is for a model to learn effective representations of entities. One way of doing this is to use contextualized word representations (CWRs) based on encoder-only models like BERT [15] and RoBERTa (Robustly optimized BERT approach) [16].

BERT "is a multi-layer bidirectional Transformer encoder" [15] that is based on the original transformer architecture proposed by Vaswani et al. [6]. Its bidirectional nature is characterized by the self-attention mechanism considering the content of the given word from both sides. Another feature of BERT is that it uses a two-step training framework: **pre-training** and **fine-tuning**. During the first stage, BERT is trained for two tasks using an unlabeled dataset. As a result, the model learns foundational knowledge and understanding of the language. Then the fine-tuning stage aims to tune the pre-trained BERT model to the specific task (e.g., NER) using labeled data. This allows the general understanding of the language captured during the pre-training stage to be utilized during downstream tasks [15]. The two tasks used for BERT pre-training are **masked language modeling** (MLM) and **next sentence prediction** (NSP). The idea of the MLM task is that some of the input tokens are randomly masked, and the model is supposed to predict the masked tokens. This aims to train a bidirectional representation as the model has access to the context from both sides of the masked token. The second task is binary classification. Two sentences, A and B, are passed to the model, and it should classify whether B is the next sentence after A. NSP helps the model capture the relation between sentences [15].

RoBERTa [16] builds on BERT but modifies the pre-training process to improve performance. It removes the NSP task, focusing solely on the MLM task. Moreover, it uses dynamic masking instead of the static masking used by BERT. The idea of static masking is that it is performed only once during the data preprocessing stage, while in dynamic masking, masked tokens are changed every time a sequence is passed to the model [16].

Even though the bidirectional nature and pre-training of BERT and RoBERTa models allow them to produce great performance in entity-related tasks, their architecture still fails to produce accurate representations of entities [3]. Firstly, they only output token-level representations of entities. For example, if the entity consists of multiple tokens, the model cannot produce such output. Because of that, the model needs to learn how to produce span-level representations of entities during the fine-tuning stage, which features a much smaller dataset than the pre-training stage. Secondly, for the transformer, it is difficult to detect the relationships between entities that consist of multiple tokens because the self-attention mechanism wasn't trained to connect groups of interconnected tokens. Finally, the MLM task is not suitable for entity representation learning, as predicting a masked word inside the entity is easier than predicting the whole entity. For example, consider the entity *"Singapore Symphony [MASK]"*; for the model, it would be easier to predict *"Orches-*

tra" than the whole entity that consists of three words [3]. The LUKE [3] model, which we focus on in the next section, aims to solve these problems.

2.4 LUKE

LUKE is a model based on RoBERTa [16]. Unlike the models described before, LUKE treats both words and entities as independent tokens. This allows LUKE to achieve state-of-the-art performance in entity-related tasks, as the model makes a distinction between words and entities in the input tokens [3].

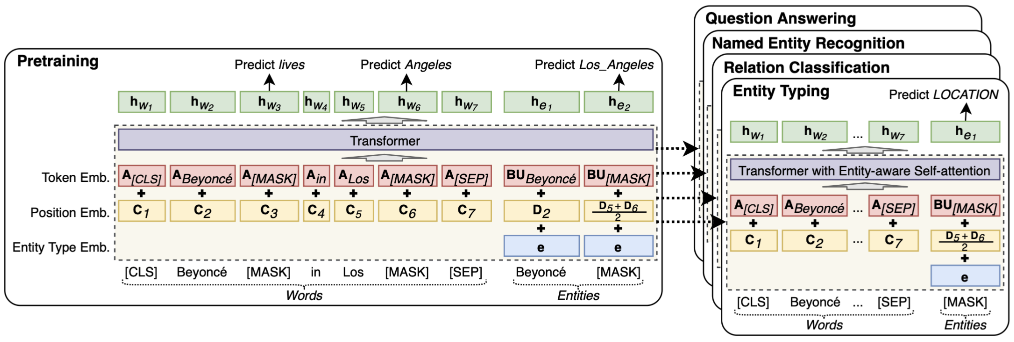


Figure 2.4: LUKE architecture. The model receives words and entities tokens as input. It consists of token embedding, positional embedding, and entity type embedding. Then the transformer computes contextualized representations of each token. Finally, representations are used for entity-related downstream tasks. Source: [3]

One of LUKE's key innovations is an entity-aware self-attention mechanism. Unlike the traditional self-attention proposed by Vaswani et al. [6], which treats all tokens uniformly, LUKE's self-attention differentiates between words and entities. LUKE has different attention scores for all possible pairs of token types: word-word, word-entity, entity-word, and entity-entity. This allows LUKE to generate more contextually rich and accurate representations for both words and entities.

The input to LUKE is comprised of three types of embeddings: token embeddings, positional embeddings, and entity type embeddings. Token embeddings are obtained from a shared vocabulary of words and entities. Each word and entity is mapped to a unique embedding vector. Positional embeddings capture the position of the token in a sequence, helping the model understand the order of words and entities. Entity type embedding indicates whether the token is an entity and corresponds to a specific entity type. These embeddings are used only for entities and are learned during the training process.

For LUKE to effectively learn representations of both entities and words, it was trained using two pre-training tasks. Similar to BERT and RoBERTa models, LUKE employs MLM, where random words are masked, and the model is trained to predict masked tokens. Additionally, an extension of MLM is used to learn entity representations. Certain entities are also masked, and the model is trained to predict those masked entities. This allows the model to develop a deep understanding of entity contexts and relationships. Consider the example illustrated in Figure 2.4: *"Beyoncé lives in Los Angeles."* The entities in this sentence are *"Beyoncé"* (Person) and *"Los Angeles"* (Place). During the training, *"lives"*, *"Angeles"* words, and *"Los Angeles"* entity are masked. Then the input, as a sum of entity type embeddings, positional embeddings, and token embeddings, is passed to the model. Finally, the model is trained to predict correct words and entities for the masked tokens.

3. Data

This section describes the data used for this study. We present the results of the data exploration stage, and then proceed to the details of data preparation and labeling. This data is further used for fine-tuning and evaluation of the models.

3.1 Description of the Data

The dataset contains information about 2,422,789 news articles issued during the years 2023 and 2024 by digital magazines such as *MarTech Series*, *Yahoo*, and *GlobeNewswire*. The attributes of the dataset include:

- Tags: Keywords that characterize the main idea of the article.
- Abstract: A short paragraph that contains the beginning of the article.
- Header: The title of the article.
- URL: The link to the article's webpage.
- Timestamp: The date and time of publication.
- Body: The main text of the article.
- Domain: The website of the digital magazine that published the article.

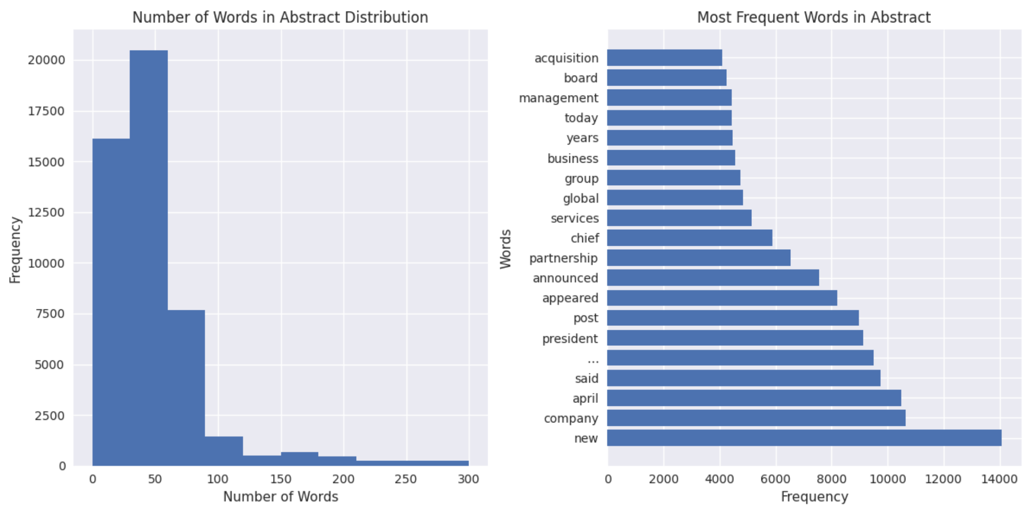


Figure 3.1: Analysis of abstract text. (Left) Distribution of the number of words in an abstract, limited to 0 - 300 words for clarity. (Right) Most frequent words in the abstracts.

We conducted an exploratory data analysis (EDA) on the first 50,000 articles from the dataset. This subset was selected due to computational limitations, as analyzing the entire dataset would have been resource-intensive. Initial preprocessing steps include removing stop words, digits, punctuation, and extra spaces, and converting text to lowercase. These steps help normalize the text data, making it more suitable for analysis. Data exploration revealed that more than half (33,088 out of 50,000) of the records have a null body variable. Additionally, some articles were published by multiple magazines with identical content. The number of duplicates with the same abstract in the considered sample is 12,458. For instance, the article titled *"Roblox Taps PubMatic to Offer Programmatic Immersive Video Ads on Its Platform"* was published on 2024-04-10 by both *YahooFinance* [17] and *MarketScreener* [18].

Given the prevalence of null values in the body variable, constraints on model input length, and limited computational resources, we decided to focus on the abstract and title. This decision was based on preliminary analysis, which showed that abstracts and titles alone contain sufficient information about entities of our interest. Figure 3.1 illustrates an overview of the abstracts: the majority of articles have around 50 words in the abstract, though some outliers have up to 3,500 words. Furthermore, the most frequent words are *company*, *president*, and *partnership*, indicating that the majority of the news is about politics or the corporate world.

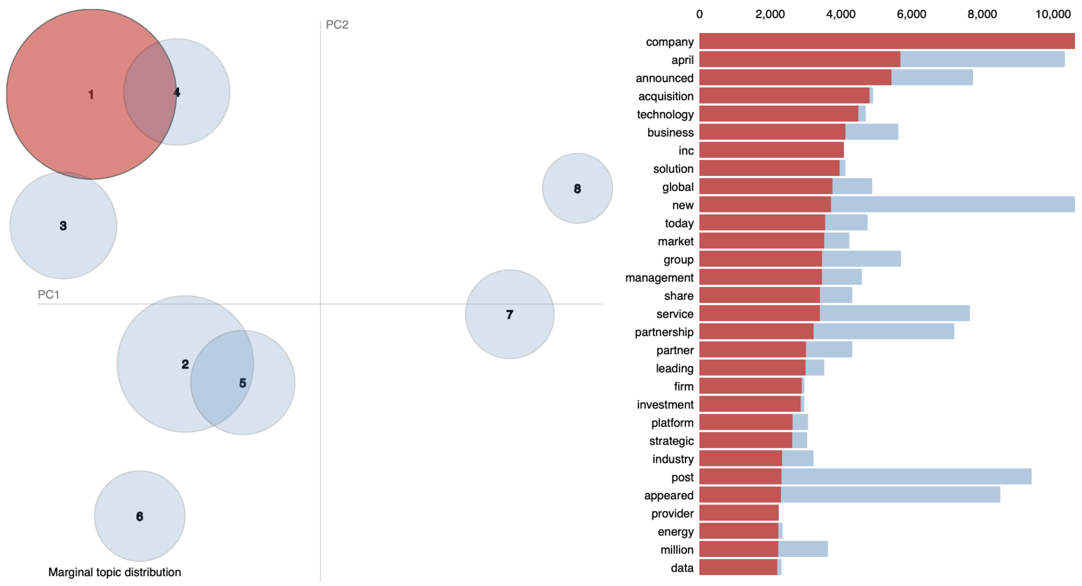


Figure 3.2: Topic modeling results. (Left) Inter-topic distance map. (Right) Top salient terms for the most significant topic.

To gain further insights, we performed topic modeling using Latent Dirichlet Allocation (LDA), a method that assumes each document is a collection of a small number of topics, and each word in the document can be attributed to one of these topics [19]. Figure 3.2 shows the results of modeling eight clusters of articles. Each bubble on the left side of the figure represents a cluster. The larger the bubble, the more articles in the corpus are related to this cluster. Additionally, the distance between bubbles indicates the similarity between clusters. For example, clusters one and four are similar, while clusters one and seven are completely different. On the right side, the histogram presents the most relevant words for all articles (in blue) and for the biggest cluster, number one (in red).

As a result of topic modeling, we observe the following topics in the corpus:

- **Business**: Clusters one, three, and four, with words like *company*, *acquisition*, *business*, *partnership*, and *post*.
- **Sport**: Clusters six, with words like *team*, *season*, *game*, and *coach*.
- **Legal**: Clusters seven and eight, with words like *act*, *party*, *court*, and *correctional*.
- **Politics**: Clusters two and five, with words like *Trump*, *president*, *Israel*, and *Biden*.

3.2 Preparation and Labeling of the Data

To fine-tune and evaluate NER and RE models, we manually labeled 335 articles using the Label Studio[1] platform. We labeled all entities and relations in the title and abstract variables using the following types:

- Entity *ORG*: Company or organization.
- Entity *PER*: Person.
- Relation *acquired_by*: When one organization acquires (part of) another organization. For example, in the sentence *"Apple, Inc. Acquires 50% of NeXT."* the relation between entities *"NeXT"* and *"Apple, Inc."* is of type *acquired_by*.
- Relation *partners_with*: When two organizations partner. For example, in the sentence *"Balenciaga and Crocs Renew Partnership Launching New Collection"* the relation between entities *"Balenciaga"* and *"Crocs"* is of type *partners_with*.
- Relation *works_at*: When a person is affiliated with a company. For example, in the sentence *"Bob Smith appointed ScotRail Managing Director."* the relation between entities *"Bob Smith"* and *"ScotRail"* is of type *works_at*

To maintain consistency in the labeling process, we developed a set of guidelines based on best practices in the field [20]. Firstly, a word or phrase can be considered an entity if it has a specific reference, meaning it refers to a single entity in the world and remains constant over time. For example, *"Bob the tennis player"* is not an entity, as its reference can change depending on the context and time, whereas *"Bob Smith"* is an entity, as its reference is constant. Secondly, the genitive marker is not considered part of an entity; for instance, in

[1]`https://labelstud.io/`

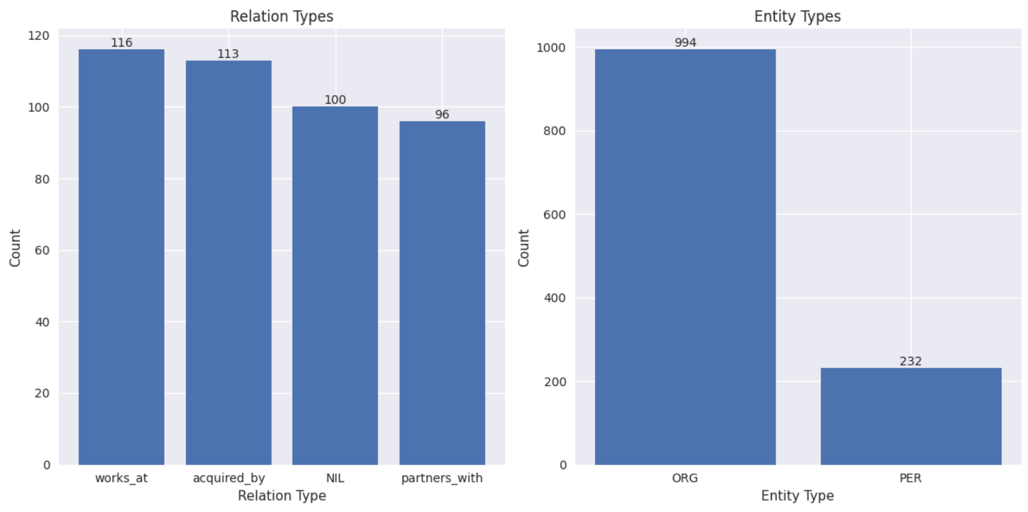


Figure 3.3: Results of data labeling. (Left) Number of relation types. (Right) Number of entity types.

"Bob's office" only *"Bob"* would be considered an entity. Thirdly, we label relations that are only within the boundaries of one sentence and do not spread across multiple sentences. For example, if two partnering companies are mentioned in separate sentences, we do not label this relation. We decided to do it this way because later, while preparing data for fine-tuning, we separate articles into sentences for greater efficiency. Finally, some of the NER models are incapable of identifying overlapping entities; therefore, we decided to label only entities that have non-overlapping spans.

We manually labeled 335 news articles and the result is shown in Figure 3.3. A total of 325 relations (excluding the NIL class) and 1,226 entities of various types were labeled. This dataset is split into train and test sets with a 67% and 33% ratio, respectively (see Appendix A.1). These datasets are used for model fine-tuning and evaluation. Also, after labeling, we added one more type of relation, NIL, *No Information Label*. We randomly created 100 pairs of entities that do not belong to any relation class and assigned them to the NIL class. This label will help LUKE entity pair classifier model to learn to detect entity pairs that do not have a relation. The details of the input data for each model, as well as the entire methodology, are discussed in the next section.

3.3 Ethical Considerations

All data used in this study was obtained from publicly available sources. No personal information beyond what is publicly available in the news articles was used or exposed during this study.

4. Method

This section is comprised of three parts. First, we focus on the models that we use for NER and RE. Then we proceed to the description of the pipelines for different models, and evaluation metrics. In each section we cover both our procedures for NER and RE.

4.1 Models

4.1.1 NER Models

To address RQ1 and RQ2, we examine and compare the performance of four NER models based on our test set. We utilize two LUKE-based models: `luke-large-finetuned-conll-2003` and a custom fine-tuned `luke-base` model on our dataset. We compare the performance of these models to two baselines: the standard spaCy[1] `en_core_web_lg` model and a fine-tuned version of the same model on our dataset.

The `luke-large-finetuned-conll-2003` model is a LUKE variant that has been specifically fine-tuned on the CoNLL-2003 dataset [21], a benchmark dataset for NER. This model is based on `luke-large`, which has 484 million parameters. As mentioned earlier, the LUKE model utilizes a transformer architecture and distinguishes between entities and words as independent tokens. We expect LUKE to outperform the baseline spaCy model due to this distinction.

We also fine-tune the `luke-base` model for the NER task on our dataset. `luke-base` has 253 million parameters. We select `luke-base` over `luke-large`, which has 484 million parameters, because of better computational efficiency and shorter training time, which is particularly important when working with limited hardware capabilities. Moreover, while `luke-large` might offer better performance in some cases, `luke-base` often provides sufficiently good performance for NER task, making it a balanced choice between performance and efficiency. Fine-tuning of `luke-base` involves adjusting the parameters of the pre-trained LUKE model to our task-specific data. We anticipate that this will improve the model's ability to recognize and classify entities that are specific to our dataset. However, the performance of this model might be relatively low due to the small size of the training dataset. Further details on fine-tuning parameters and input/output formats are covered in subsequent sections.

The spaCy `en_core_web_lg` is a fine-tuned English-language pipeline that comprises

[1]`https://spacy.io/`

multiple components, including Tok2Vec and NER. The Tok2Vec component of the pipeline generates a representation of an input sequence by mapping tokens into vector representations. The NER component of the pipeline utilizes a convolutional neural network (CNN) architecture. To further enhance the spaCy model's performance, we fine-tuned the `en_core_web_lg` model on our annotated dataset.

The selection of these models was driven by several factors. First, all four models are relatively straightforward to implement using the spaCy library for `en_core_web_lg` models and the Hugging Face Transformers[2] library for LUKE-based models. Both libraries provide detailed documentation that facilitated the implementation process. Secondly, both `en_core_web_lg` and `luke-large-finetuned-conll-2003` were chosen for their capability to recognize *Person (PER)* and *Organization (ORG)* entities.

4.1.2 RE Models

To answer RQ3, we compare the performance of two RE models: a LUKE pipeline-based RE model and a Seq2Seq REBEL (Relation Extraction By End-to-end Language generation) [4] model. Our goal is to evaluate and compare their effectiveness in identifying and classifying relationships between entities within our dataset.

The LUKE model is inherently not designed for direct RE tasks, because of that we implement LUKE in a two-stage pipeline to create an RE solution. Initially, we perform NER using one of the LUKE-based models from the previous section. Subsequently, we classify the relationships between the identified entities. To achieve this, we fine-tune the `luke-base` entity pair classifier specifically based on our training dataset. This fine-tuning involves adjusting LUKE's parameters to optimize its performance for our specific relation types, which include *partners_with*, *acquired_by*, *works_at*, and *NIL*.

From the REBEL family [4], we employ the `rebel-large` model and fine-tune it on our dataset. It has 406 million parameters and is an encoder-decoder model developed specifically for relation extraction. REBEL was pre-trained on a large dataset and achieves state of art results on variety of benchmark datasets for RE, including CoNLL04 [22]. REBEL processes text to generate text sequences that describe relationships between entities. By fine-tuning REBEL, we tailor the model to classify relations into the required classes specific to our dataset.

The selection of these models was driven by our goal to evaluate the effectiveness of LUKE for relation extraction tasks and compare its pipeline-based approach with a direct Seq2Seq solution using REBEL. LUKE's pipeline approach leverages its entity recognition capabilities to sequentially infer relationships between entities, while REBEL's direct Seq2Seq architecture aims to capture relational semantics directly from textual input, offer-

[2]`https://huggingface.co/docs/transformers/index`

ing complementary perspectives on the task of relation extraction.

4.2 Pipelines

As was described in the previous section, we manually labeled entities and relations in 335 news articles with labels *ORG*, *PER* for entities and *works_at*, *acquired_by*, *partners_with* for relations. Subsequently, we split the articles into sentences to ensure that the input length does not exceed the maximum limit for the models. Finally, we divided this dataset into train and test sets with a ratio of 67% for fine-tuning and 33% for evaluation purposes.

4.2.1 Pipeline for NER

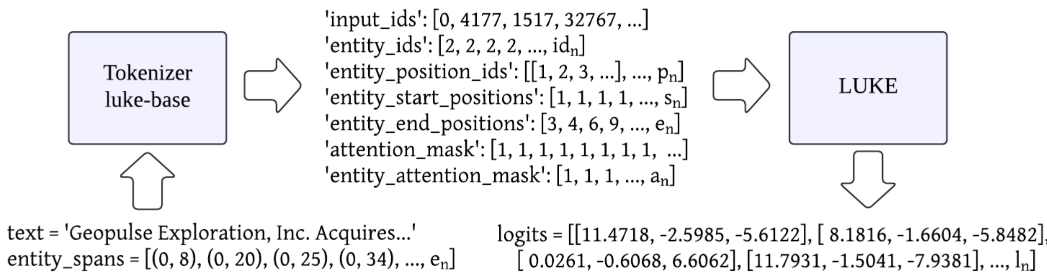

Tokenizer
luke-base
'input_ids': [0, 4177, 1517, 32767, ...]
'entity_ids': [2, 2, 2, 2, ..., id$_n$]
'entity_position_ids': [[1, 2, 3, ...], ..., p$_n$]
'entity_start_positions': [1, 1, 1, 1, ..., s$_n$]
'entity_end_positions': [3, 4, 6, 9, ..., e$_n$]
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, ...]
'entity_attention_mask': [1, 1, 1, ..., a$_n$]
LUKE
text = 'Geopulse Exploration, Inc. Acquires...'
entity_spans = [(0, 8), (0, 20), (0, 25), (0, 34), ..., e$_n$]
logits = [[11.4718, -2.5985, -5.6122], [8.1816, -1.6604, -5.8482],
[0.0261, -0.6068, 6.6062], [11.7931, -1.5041, -7.9381], ..., l$_n$]

Figure 4.1: LUKE pipeline for NER. The Tokenizer receives *text* and a list of *n* entity spans as inputs. It produces: *input_ids*, *entity_ids*, *entity_position_ids*, *entity_start_positions*, *entity_end_positions*, *attention_mask*, and *entity_attention_mask*. These variables are then passed to the LUKE model, which returns the *logits*, a list of classification scores of size *n*.

LUKE-based models perform NER by classifying all possible entity spans into three classes: *ORG*, *PER*, and *NIL* (indicating the span does not belong to any predefined class). An entity span is a tuple that contains start and end character indices of an entity in an input sentence. Thus, as an input for the LUKE-based model, we pass not only the text, but also a list of all possible entity spans in the sentence. We create this list by splitting the input sentence into words and constructing all possible spans that are not longer than 6 words. We limit the number of words in the candidate entity in order to reduce the number of values that the model needs to classify and, thus, reduce the number of computations. Figure 4.1 illustrates the whole NER pipeline for LUKE-based models. We use a tokenizer for the `luke-base` model, which receives *text* and *entity_spans* as input. The tokenizer breaks down the text into subwords, and encodes them into a format that the model can process. As an output, the tokenizer produces a dictionary with the following items:

- *input_ids*: List of token IDs that represent the input after tokenization. Each ID corresponds to the subword in the input sequence as per the tokenizer's vocabulary.
- *entity_ids*: List of IDs that represent entities in the input text. The size of *entity_ids* matches the size of the input *entity_spans* list.
- *entity_position_ids*: List of lists, where each inner list contains the indices of positions

of input entities in *input_ids* list.

- *entity_start_positions*: List of indices of tokens in the *input_ids* list that indicate the start of the entities.
- *entity_end_positions*: List of indices of tokens in the *input_ids* list that indicate the end of the entities.
- *attetnion_mask*: List of binary values (0 or 1) that indicate which tokens should be attended by models. This helps models to ignore padding tokens.
- *entity_attention_mask*: Similar to *attention_mask*, but specifically for entities.

The output of the tokenizer is passed to the model that outputs the logits, unnormalized scores, for each input entity span. Logits represent a score for each class, which indicates how strongly the model predicts that the input entity span belongs to each class. For example, in Figure 4.1, the model predicts that the first, second, and fourth entity spans belong to the first class, while the third entity span belongs to the third class. In this example, we have the following relation between classes and indices in the prediction lists: `{0:"NIL", 1:"PER", 2:"ORG"}`. Thus, in the considered example, the model correctly predicted that the third entity, *"Geopulse Exploration, Inc."*, is an organization, while the rest of the entities cannot be identified as an organization or person.

In addition to the implementation of the `luke-large-finetuned-con_ll-2003` model as was described above, we also fine-tuned the `luke-base` model using our dataset. We performed fine-tuning with 4 epochs, meaning we go over the training set four times. To train the model, we iterate over our train set in batches of size 4, padding each batch to the longest sequence in the batch. We used those settings for the fine-tuning, as they were recommended in one of the tutorials by the LUKE developers. Each iteration involves the model making the prediction, computing the loss by comparing the model's prediction to the true labels, and performing backpropagation, an algorithm that computes the network parameters updates. As a result of the fine-tuning process, we receive the LUKE model with updated weights that are optimized for our data. Finally, we implement this fine-tuned model as was described in the previous paragraphs.

The procedure for spaCy models slightly resembles the procedure for LUKE models. We use the already fine-tuned model `en_core_web_lg` for prediction. Also, we perform our own fine-tuning of `en_core_web_lg` by configuring a custom training setup using spaCy's configuration system.

4.2.2 Pipeline for RE

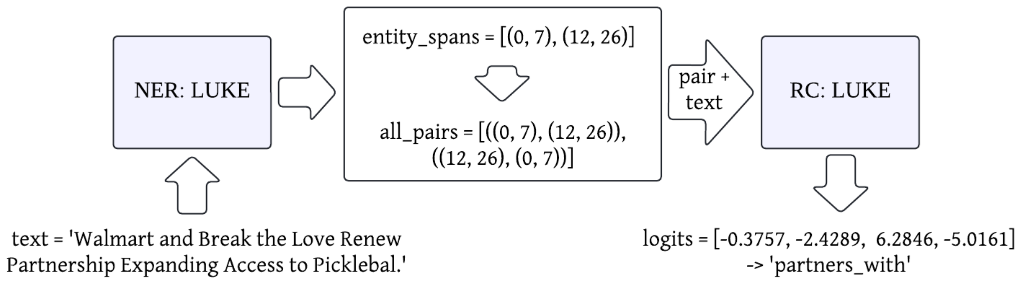

NER: LUKE
entity_spans = [(0, 7), (12, 26)]
all_pairs = [((0, 7), (12, 26)), ((12, 26), (0, 7))]
pair + text
RC: LUKE
text = 'Walmart and Break the Love Renew Partnership Expanding Access to Picklebal.'
logits = [-0.3757, -2.4289, 6.2846, -5.0161] -> 'partners_with'

Figure 4.2: LUKE pipeline for RE. First, the NER model detects all the entities of types *ORG* and *PER* in the input sentence. Then, we create all possible permutations of the detected entities. Finally, the relation classification (RC) model makes a prediction for each possible pair.

Figure 4.2 explains the RE pipeline based on LUKE models in greater detail. First, we implement an NER LUKE model that we fine-tuned on our data. After it classifies all the entity spans, we create all possible permutations of size two out of entity spans of type *ORG* or *PER*. Then, we iterate over the list of permutations and use the RE model to classify the relation between the entities in each pair:

1. The tokenizer for `luke-base` receives text and a pair of entity spans as input and outputs *input_ids*, *entity_ids*, *entity_position_ids*, *attention_mask*, *entity_attention_mask*.
2. The encoded input is passed to the LUKE model, fine-tuned on our data, which classifies the pair into one of the classes: *works_at*, *acquired_by*, *partners_with*, and *NIL*.

To implement the second part of the pipeline, relation classification, we fine-tune the `luke-base` model with the same settings and procedure as for NER.

Unlike the solution based on LUKE models, the implementation of the REBEL model involves fewer steps, as it performs both NER and relation classification. The REBEL tokenizer receives text as input and outputs *input_ids* and *attention_mask*. Then, the model generates a sequence in the following format:

```
"<triplet> head1 <subj> tail1 <obj> relation1 <triplet> head2 <subj> tail2
                        <obj> relation2 ...",
```

where `head` and `tail` are entities that participate in the relation, and `relation` is the relation type. Thus, as a result, we receive triplets extracted by the model in one output. In order for REBEL to detect the required relations, we fine-tune `rebel-large` with the same settings and procedure as for the LUKE model.

4.3 Evaluation

To estimate the performance of the NER and RE models, we compare the models' predictions with our annotations using the "exact-match" evaluation strategy. This strategy

Entity	Ground truth	Prediction	For 'ORG' class	For 'PER' class
CS Disco, Inc.	ORG	ORG	TP	
Board of Directors	NIL	NIL		
Eric Friedrichsen	PER	PER		TP
Presindent	NIL	ORG	FP	
Chief	NIL	PER		FP
Macy	ORG	PER	FN	FP
proxy fight	NIL	NIL		
Arkhouse	ORG	NIL	FN	
Ernst Rustenhoven	PER	ORG	FP	FN
CEO	NIL	NIL		
ILGM	ORG	ORG	TP	

Table 4.1: Example of NER evaluation. The *Entity* column contains entities that are classified by the model. The *Ground truth* column contains the true labels for each entity. The *Prediction* column contains the predictions made by the model. The *For 'ORG' class* column contains indicators showing whether the prediction is TP, FP, or FN for the *ORG* class. The *For 'PER' class* column contains indicators showing whether the prediction is TP, FP, or FN for the *PER* class.

assumes that a named entity or relation is correctly recognized if the model correctly identifies both the entity spans and type [1]. We calculate precision, recall, and F1-score based on the number of true positives (TP), false positives (FP), and false negatives (FN) with respect to class i.

$$P_i = \frac{TP_i}{TP_i + FP_i} \quad (4.1) \qquad R_i = \frac{TP_i}{TP_i + FN_i} \quad (4.2) \qquad F1_i = \frac{2 \times P_i \times R_i}{P_i + R_i} \quad (4.3)$$

- Precision (4.1) is the proportion of correctly classified items (TP) among all items assigned by the model to class i.
- Recall (4.2) is the proportion of correctly classified items (TP) among all items that belong to class i.
- F1-score (4.3) is the harmonic mean of precision and recall.

To obtain the final performance metrics for the model, we compute macro precision, recall, and F1-score by averaging each metric for all the classes except NIL class:

$$P^M = \frac{\sum_{i=1}^{C} P_i}{C} \quad (4.4) \qquad R^M = \frac{\sum_{i=1}^{C} R_i}{C} \quad (4.5) \qquad F1^M = \frac{\sum_{i=1}^{C} F1_i}{C} \quad (4.6)$$

where C is total number of classes except NIL class.

Table 4.1 provides an example of how the performance of the NER model is estimated. According to this example, the recall for the ORG class is 0.5, as we have two cases where the ORG entity was recognized correctly and two cases where the ORG entity wasn't recognized by the model. The precision for the ORG class is also 0.5, as we have two cases where the ORG label was assigned incorrectly. Similarly, the precision for the PER class is 0.333. Finally, to compute macro-precision, we take the average of the precision values for each class and get 0.417. The evaluation for the RE follows the same logic, except that we have entity pairs instead of entities in the left column, and the classes are *works_at*, *acquired_by*, and *partners_with*.

5. Results

This section presents the results of the NER and RE models tested in this study. To evaluate the models, we use the test set to compare the predictions made by the models with the true labels. Then we calculate macro precision (4.4), recall (4.5), F1-score (4.6) metrics, and class-specific precision (4.1), recall (4.2), and F1 (4.3) scores.

5.1 Overview of the NER results

Model	Macro-P	Macro-R	Macro-F1	P (PER)	P (ORG)	R (PER)	R (ORG)	F1 (PER)	F1 (ORG)
LUKE-conll-2003	0.732	0.854	0.788	0.809	0.655	0.905	0.804	0.854	0.722
LUKE-fine-tuned	0.719	0.573	0.635	0.780	0.659	0.548	0.598	0.643	0.627
spaCy	0.603	0.612	0.607	0.686	0.519	0.702	0.521	0.694	0.520
spaCy-fine-tuned	0.823	0.824	0.823	0.882	0.763	0.893	0.756	0.888	0.759

Table 5.1: Performance metrics for NER models.

Table 5.1 displays the performance metrics for the four NER models evaluated in this study:

- `LUKE-conll-2003`: LUKE model fine-tuned on the CoNLL-2003 dataset [21].
- `LUKE-fine-tuned`: LUKE model fine-tuned on the training dataset that we prepared.
- `spaCy`: spaCy `en_core_web_lg` model.
- `spaCy-fine-tuned`: spaCy `en_core_web_lg` model fine-tuned on the training dataset that we prepared.

The `spaCy-fine-tuned` model achieved the highest macro-F1 score (0.823), demonstrating superior overall performance in recognizing both *PER* and *ORG* entities. This is followed by the `LUKE-conll-2003` model with a macro-F1 score of 0.788. The `LUKE-fine-tu_-ned` model performed moderately with a macro-F1 score of 0.635, while the `spaCy` model has the lowest performance with a macro-F1 score of 0.607.

Comparing LUKE-based models with the spaCy solutions, we can observe that `LUKE-_-conll-2003` performed significantly better than `spaCy`. However, models fine-tuned on the news dataset have the reverse relation; the `spaCy-fine-tuned` model is noticeably more effective than `LUKE-fine-tuned` in identifying both *ORG* and *PER* classes. Moreover, if we compare `LUKE-conll-2003` and `spaCy-fine-tuned`, we observe that the spaCy model outperforms LUKE according to F1-scores, while the LUKE model has higher recall and lower precision metrics. This means that it detected a greater share of the true entities but made more mistakes than spaCy.

In terms of individual class performance, the `spaCy-fine-tuned` model exhibits high precision and recall for both *PER* and *ORG* classes. Meanwhile, the `LUKE-conll-2003` model shows strong precision for *PER* (0.809) but relatively lower precision for *ORG* (0.655). This indicates that the model is better at classifying the *PER* class and has more false positives for the *ORG* class in its predictions. In general, this behavior is not unique to the `LUKE-conll-2003` model, and we observe that the majority of models are better at recognizing persons than organizations.

It is important to note that the dataset used for NER had a class imbalance, with 994 instances of *ORG* and only 232 instances of *PER*. This imbalance could have impacted the models' ability to accurately identify entities, particularly for the underrepresented class. The higher number of *ORG* entities might have contributed to the models' lower precision for this class, as they had more opportunities to make false positive predictions.

5.2 Overview of the RE results

Model	Macro-P	Macro-R	Macro-F1	P (W)	P (P)	P (A)	R (W)	R (P)	R (A)	F1 (W)	F1 (P)	F1 (A)
LUKE-RC-fine-tuned	0.952	0.984	0.966	1.000	0.973	0.882	0.978	0.973	1.000	0.989	0.973	0.938
LUKE-RE-fine-tuned	0.589	0.383	0.453	0.917	0.556	0.294	0.458	0.357	0.333	0.611	0.435	0.313
REBEL-fine-tuned	0.828	0.695	0.752	0.903	0.794	0.786	0.622	0.730	0.733	0.737	0.761	0.759

Table 5.2: Performance metrics for RE and relations classification models.

Table 5.2 displays the performance metrics for the one relation classification model and the two RE models:

- `LUKE-RC-fine-tuned`: LUKE model for relation classification fine-tuned on the training dataset we prepared.
- `LUKE-RE-fine-tuned`: LUKE pipeline for relation extraction, with both NER and relation classifier models fine-tuned on the training dataset we prepared.
- `REBEL-fine-tuned`: REBEL model for relation extraction fine-tuned on the training dataset.

It is important to note that comparing the performance of `LUKE-RC-fine-tuned` with the other models may not be entirely fair due to the differences in the tasks they perform. The `LUKE-RC-fine-tuned` is specifically designed for relation classification, which involves identifying the type of relationship between two entities given that they are already identified. In contrast, the `LUKE-RE-fine-tuned` and `REBEL-fine-tuned` models perform relation extraction, a more complex task that requires both identifying the entities and determining the relationship between them.

We observe that the `LUKE-RC-fine-tuned` model achieves great performance across all metrics. It scored a macro-F1 of 0.966, with macro-precision and macro-recall values of 0.952 and 0.984, respectively. Notably, it achieved perfect precision (1.000) in the *works_at*

(W) class and perfect recall (1.000) in the *acquired_by* (A) class. These perfect scores indicate that the model made no false positive errors in identifying the *works_at* relationships and no false negative errors in identifying the *acquired_by* relationships. This probably signals that the test set is too small and there might be not enough examples for the model to make mistakes.

The `LUKE-RE-fine-tuned` model achieves a macro-F1 score of 0.453. It performs best in the *works_at* (W) class, attaining an F1-score of 0.611, albeit with a lower recall of 0.458. Performance for *partners_with* (P) and *acquired_by* (A) classes is notably lower, with macro-F1 scores of 0.435 and 0.313, respectively. In contrast, the `REBEL-fine-tuned` model demonstrated better overall performance with a macro-F1 score of 0.752. It excelled in the *works_at* (W) class with an F1-score of 0.737. In *partners_with* (P) and *acquired_by* (A) relations, it also performed well, achieving F1-scores of 0.761 and 0.759, respectively.

6. Conclusion and Discussion

In this section, we conclude the study by answering the research questions. Then, we cover the limitations of the study and suggest future work that can build on this study.

6.1 Answering the Research Questions

In this study, we attempt to answer the following research questions:

- RQ1: How effective is the LUKE model for NER compared to the baseline spaCy solution?
- RQ2: How effective is fine-tuning the LUKE model for NER?
- RQ3: How effective is fine-tuning the LUKE model for RE compared to the REBEL model?

To answer these questions, we implemented different models for NER and RE tasks and analyzed their performance on a manually labeled dataset comprised of news articles. For RQ1, the results indicate that the LUKE model fine-tuned on the CoNLL-2003 dataset [21] is relatively effective for our NER task and performs better than the baseline spaCy model. This suggests that LUKE's transformer-based architecture, which distinguishes entities and words, might contribute to its higher performance. However, when comparing the LUKE-based models to the fine-tuned spaCy model on our specific dataset, spaCy outperforms them according to almost all metrics. This indicates that while LUKE is inherently strong, spaCy adapts better to fine-tuning on this specific dataset. The reason for these observations might be due to the fact that there are not enough observations in our training dataset to properly fine-tune the transformer, while at the same time, it is a sufficient amount of data for training the CNN, on which the spaCy solution is based.

Considering RQ2, we probably cannot provide an unambiguous answer. The LUKE model fine-tuned based on our training dataset performed worse when comparing it to the LUKE model fine-tuned on the CoNLL-2003 [21]. There are several reasons for that. Firstly, this suggests that the dataset used for fine-tuning might not have been large enough for LUKE to learn, pointing to a limitation in the fine-tuning process. Secondly, the LUKE model fine-tuned on the CoNLL-2003 dataset benefits from the significantly larger training dataset with diverse examples. Finally, this model is based on `luke-large`, which has 484 million parameters, whereas our LUKE model is based on `luke-base`, with 253 million parameters. The larger model size of `luke-large` likely allows it to capture more complex patterns and nuances in the data, contributing to its superior performance.

For RQ3, the LUKE relation classifier demonstrated exceptional performance and its performance is significantly higher than that of the LUKE pipeline for RE, which performed relatively low. The stark difference suggests that the overall performance of the pipeline solution is affected by the NER model, which fails to detect entities in the first step, and those errors are propagated further to the relation classification stage. For that reason, the performance of the LUKE RE pipeline is less robust compared to the REBEL model. REBEL's end-to-end approach appears to handle the complexity of RE task more efficiently than the sequential LUKE-based pipeline.

6.2 Limitations

Within this study, there are several limitations that impacted the outcomes of this study. Firstly, the results of the LUKE fine-tuning indicate that the size and diversity of the training dataset were insufficient for optimal fine-tuning. The limited dataset posed a significant challenge as transformer-based models typically require vast amounts of data to learn patterns. The insufficiency in the dataset's volume resulted in the model's inability to generalize and perform well on unseen data. A larger, more diverse dataset could potentially enhance the final performance of the model.

Secondly, the disparity in model architectures and training data volume also impacted performance comparisons. The LUKE model fine-tuned on the CoNLL-2003 dataset used the `luke-large` variant, which possesses significantly more parameters than the `luke-base` variant used in our fine-tuning process. This discrepancy likely contributed to the performance gap observed between the models. The `luke-large` model, benefiting from a bigger architecture and training dataset, was able to achieve better performance metrics, thus highlighting the limitations imposed by the smaller `luke-base` variant and our dataset.

Moreover, the sequential nature of the LUKE RE pipeline introduced significant limitations due to error propagation. In this pipeline approach, the performance of the relation classification step is heavily dependent on the success of the initial NER step. If the NER model fails to accurately detect entities, these errors are propagated to the relation classification stage, compounding the overall error rate. In our study, this was the case, and the NER model used in the pipeline failed to show great performance. The performance of the entire pipeline could potentially be improved by replacing the NER model with another model that has higher performance.

6.3 Future Work

To build on the findings of this study, future research might concentrate on several key areas. Firstly, expanding the size and diversity of the training dataset is essential. A more extensive and varied dataset could significantly improve the fine-tuning process and overall model performance, particularly for transformer-based models like LUKE that require large amounts of data.

Parameter tuning is another possible area for future research. In this study, we did not test different model parameters when fine-tuning them. The decision was influenced by the scope and focus of the current research, which prioritized the comparison of LUKE model performance for NER and RE tasks with other solutions. However, systematically exploring different hyperparameters, such as learning rates, batch sizes, and dropout rates, could yield significant performance gains. Automated hyperparameter optimization techniques like grid search can be used to obtain optimal settings for the LUKE and other models.

Additionally, future research could explore the integration of external knowledge bases to enhance model performance. Leveraging structured knowledge sources could provide models with additional context and improve their ability to recognize and classify entities and relationships accurately.

6.4 Ethical Considerations

Throughout the course of this study, several ethical principles have been central to our approach. We endeavored to maintain transparency at every stage, from data preparation through to model fine-tuning and evaluation, by comprehensively documenting all the processes. Additionally, we focused on bias mitigation by examining the dataset to ensure adequate representation of less commonly occurring entities and relations. This approach aimed to prevent the underrepresentation of certain entities and relations, which could otherwise lead to biased model outputs and evaluations. Finally, accountability has been upheld by carefully recording all decisions and methodologies, aiming for reproducibility in our findings.

7. Code Availability

The code used in this study is available at `https://github.com/NataliaKurd/NK_ADS_thesis`. Please be aware that the GitHub repository is managed and updated by the author, and it may feature more recent versions or improvements of the code used in this study.

A. Appendix

A.1 Test, Train split

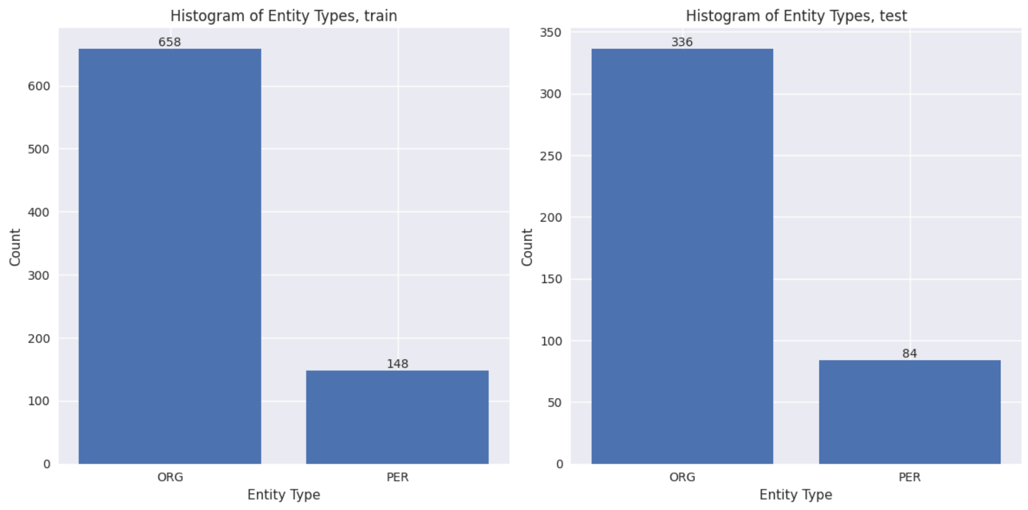


Figure A.1: Test, Train split NER. Distribution of entity types among test and train sets.

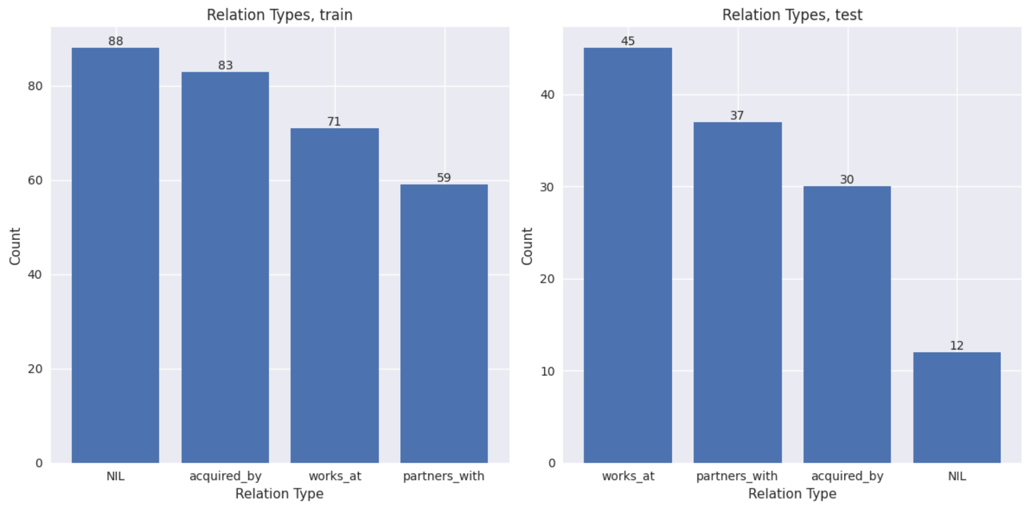


Figure A.2: Test, Train split RE. Distribution of relation types among test and train sets.

Bibliography

[1] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE transactions on knowledge and data engineering*, vol. 34, no. 1, pp. 50–70, 2020.

[2] H. Wang, K. Qin, R. Y. Zakari, G. Lu, and J. Yin, "Deep neural network-based relation extraction: An overview," *Neural Computing and Applications*, pp. 1–21, 2022.

[3] I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto, "Luke: Deep contextualized entity representations with entity-aware self-attention," *arXiv preprint arXiv:2010.01057*, 2020.

[4] P.-L. H. Cabot and R. Navigli, "Rebel: Relation extraction by end-to-end language generation," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 2370–2381.

[5] T. Al-Moslmi, M. G. Ocaña, A. L. Opdahl, and C. Veres, "Named entity extraction for knowledge graphs: A literature overview," *IEEE Access*, vol. 8, pp. 32 862–32 881, 2020.

[6] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] G. Simoes, H. Galhardas, and L. Coheur, "Information extraction tasks: A survey," *Simpósio de Informática*, vol. 540, pp. 1–550, 2009.

[8] D. Hanisch, K. Fundel, H.-T. Mevissen, R. Zimmer, and J. Fluck, "Prominer: Rule-based protein and gene entity recognition," *BMC bioinformatics*, vol. 6, pp. 1–9, 2005.

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[10] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," pp. 1003–1011, 2009.

[11] Z. Xiaoyan, D. Yang, Y. Min, *et al.*, "A comprehensive survey on deep learning for relation extraction: Recent advances and new frontiers," *arXiv preprint arXiv:2306.02051*, 2023.

[12] L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, 2022, ISBN: 9781098103248. [Online]. Available: `https://books.google.nl/books?id=pNBpzwEACAAJ`.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[14] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[16] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[17] YahooFinance. "Roblox taps pubmatic to offer programmatic immersive video ads on its platform." (2024), [Online]. Available: `https://finance.yahoo.com/news/roblox-taps-pubmatic-offer-programmatic-130500197.html?guccounter=1&guce_referrer=aHR0cHM6Ly9kdWNrZHVja2dvLmNvbS8&guce_referrer_sig=AQAAABui11RSDl41zemLzC-C8bC8cCWMqGJwv6IF_jtGBsyh3hZppf54jnLSEGvhkTy1lXJIYcKMMau3hG3eY3Ge59OGgG3JuFu63yjo6kyMWTp5R6Gt9ydV_pfwIM13op7jnO09_E7DgOmD3oG6hDJM-bfXSkXgKKk-bjRnX791atIX` (visited on 07/03/2024).

[18] MarketScreener. "Roblox taps pubmatic to offer programmatic immersive video ads on its platform." (2024), [Online]. Available: `https://www.marketscreener.com/quote/stock/PUBMATIC-INC-116353435/news/Roblox-Taps-PubMatic-to-Offer-Programmatic-Immersive-Video-Ads-on-its-Platform-46406483/` (visited on 07/03/2024).

[19] U. Chauhan and A. Shah, "Topic modeling using latent dirichlet allocation: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–35, 2021.

[20] S. Mayhew, *Universal ner, annotation guidelines*, `http://www.universalner.org/guidelines/`, Accessed: 2024-05-31.

[21] E. F. Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," *arXiv preprint cs/0306050*, 2003.

[22] D. Roth and W.-t. Yih, "A linear programming formulation for global inference in natural language tasks," in *Proceedings of the eighth conference on computational natural language learning (CoNLL-2004) at HLT-NAACL 2004*, 2004, pp. 1–8.