MASTER THESIS

NEURAL INFORMATION EXTRACTION PIPELINE FOR CYBER FORENSICS WITH PRE-TRAINED LANGUAGE MODELS

Fillipe Barros Rodrigues

Brasília, May 2022

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

NEURAL INFORMATION EXTRACTION PIPELINE FOR CYBER FORENSICS WITH PRE-TRAINED LANGUAGE MODELS

PIPELINE DE EXTRAÇÃO DE INFORMAÇÕES NEURAIS PARA FORENSE CIBERNÉTICA COM MODELOS DE LINGUAGEM PRÉ-TREINADOS

FILLIPE BARROS RODRIGUES

ORIENTADOR: WILLIAM FERREIRA GIOZZA, DR.
COORIENTADOR: ROBSON DE O. ALBUQUERQUE, DR.

DISSERTAÇÃO DE MESTRADO PROFISSIONAL EM
ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPEE.MP.017
BRASÍLIA/DF: MAIO – 2022

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

NEURAL INFORMATION EXTRACTION PIPELINE FOR CYBER FORENSICS WITH PRE-TRAINED LANGUAGE MODELS

Fillipe Barros Rodrigues

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Robson de O. Albuquerque, Ph.D, FT/UnB _____
Presidente

Prof. João José Costa Gondim, Ph.D, CIC/UnB _____
Examinador interno

Profa. Ana Lucila Sandoval Orozco, Ph.D, UCM _____
Examinadora externa

Prof. Georges Daniel Amvame Nze, Ph.D, FT/UnB _____
Suplente

FICHA CATALOGRÁFICA

RODRIGUES, FILLIPE BARROS
NEURAL INFORMATION EXTRACTION PIPELINE FOR CYBER FORENSICS WITH PRE-TRAINED LANGUAGE MODELS [Distrito Federal] 2022.
xvi, 113 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2022).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Information Extraction
2. Natural Language Processing
3. Cyber Forensics
4. Pre-trained Language Models

I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

RODRIGUES, F. B. (2022). *NEURAL INFORMATION EXTRACTION PIPELINE FOR CYBER FORENSICS WITH PRE-TRAINED LANGUAGE MODELS* . Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 113 p.

Fillipe Barros Rodrigues
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATION

I dedicate this work to those who have the courage to pursue their dreams.

ACKNOWLEDGEMENTS

First of all, I thank God, who enabled me for this achievement.

I would like to thank professors Robson de Oliveira Albuquerque and William Ferreira Giozza, for their guidance, dedication and support, without which this work would not have been possible.

A special thanks to professor Luis Javier García Villalba from Universidad Complutense de Madrid (UCM), Madrid, Spain, for his support and guidance in the publication of this work.

I am grateful for the tips, support and resources provided by the fellow researchers from the LATITUDE laboratory, represented by Fábio Mendonça and Kelly Santos.

A wholehearted acknowledgment to my parents and my sister, whose faith and confidence in me made this dream possible.

I also wish to thank friends and fellow students who helped and encouraged me during the development of this research, in particular, my friends Lucas Amaro and Trajano Melo.

Finally, I wish to thank the University of Brasília and the Professional Post-Graduate Program in Electrical Engineering (PPEE) and its staff, for their exceptional quality, patience and support.

ABSTRACT

Digital investigation is a challenging task composed of several steps and procedures that are often slow and error-prone. From collection to analysis and reporting, many sub-tasks are dependent on technological solutions that aim to reduce the overhead of information the investigators face. Some forensic tools offer mechanisms to automate searches and categorizations of relevant objects and data, but they still lack support for complex and dynamic scenarios, especially regarding Big Data applications. Most of this data is unstructured, meaning it is harder to organize and find meaningful insights. Recent advances in Machine Learning (ML) and, more specifically, in Natural Language Processing (NLP) brought to existence new architectures and language models that can be used to significantly increase the performance of Information Extraction (IE) tasks, such as Named Entity Recognition (NER) and Relation Extraction (RE). This work proposes a reproducible setup to build, test and fine-tune an information extraction pipeline that can be applied to cyber forensics investigations using NLP of texts in different languages. The tasks of NER and RE are deeply explored and discussed, based on the latest state-of-the-art language models such as BERT and RoBERTa. It is also discussed how to optimize the performance of an NLP model by tuning its hyper-parameters, validating their performance with standard evaluation metrics and comparisons with well-known benchmarks. The proposed pipeline is also applied to different application scenarios with hypothetical yet realistic examples, culminating with the presentation of comprehensive knowledge graphs for structured information analysis. The components of the pipeline are organized in such a way that makes it possible to configure both entity and relationship classes for different application domains with minimal changes, not limited to the cyber forensics context nor a specific language. The results presented in this work for both Portuguese and English achieved state-of-the-art performance following the steps proposed for each task, corroborating the idea that staged processing has the potential to further improve the final result.

Keywords: Information Extraction, Named Entity Recognition, Relation Extraction, Natural Language Processing, Cyber Forensics, Knowledge Graphs.

RESUMO

A investigação digital é uma tarefa desafiadora composta por várias etapas e procedimentos que são muitas vezes lentos e propensos a erros. Da coleta de informações à análise e comunicação de resultados, muitas subtarefas são dependentes de soluções tecnológicas que visam reduzir a sobrecarga de informações que os investigadores enfrentam. Algumas ferramentas forenses oferecem mecanismos para automatizar pesquisas e categorizações de objetos e dados relevantes, mas ainda não têm suporte para cenários complexos e dinâmicos, especialmente em aplicações de *Big Data*. A maioria desses dados não é estruturada, o que significa que é mais difícil organizar e encontrar *insights* significativos. Avanços recentes em *Machine Learning* (ML) e, mais especificamente, em Processamento de Linguagem Natural (NLP) trouxeram à existência novas arquiteturas e modelos de linguagem que podem ser usados para aumentar significativamente o desempenho de tarefas de Extração de Informações (IE), como Reconhecimento de Entidades Mencionadas (NER) e Extração de Relacionamentos (RE). Este trabalho propõe uma configuração reproduzível para construir, testar e ajustar um *pipeline* de extração de informação que pode ser aplicado a investigações forenses cibernéticas usando NLP de textos em diferentes idiomas. As tarefas de NER e RE são profundamente exploradas e discutidas, com base nos mais recentes modelos de linguagem natural, como BERT e RoBERTa. Também é discutido como otimizar o desempenho de um modelo de NLP ajustando seus hiperparâmetros, validando seu desempenho com métricas de avaliação padrão e comparações com *benchmarks* bem conhecidos. O *pipeline* proposto também é aplicado a diferentes cenários com exemplos hipotéticos mas realistas, culminando com a apresentação de grafos de conhecimento abrangentes para análise de informações. Os componentes do *pipeline* são organizados de forma que seja possível configurar classes de entidade e de relacionamento para diferentes domínios de aplicação com alterações mínimas, não limitadas ao contexto da análise forense cibernética nem a uma linguagem específica. Os resultados apresentados neste trabalho tanto para o português quanto para o inglês alcançaram níveis de desempenho do estado da arte seguindo as etapas propostas para cada tarefa, corroborando a ideia de que o processamento por etapas tem potencial para melhorar ainda mais o resultado final.

Palavras-chave: Extração de Informações, Reconhecimento de Entidades Mencionadas, Extração de Relacionamentos, Processamento de Linguagem Natural, Forense Cibernética, Grafos de Conhecimento.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ACRONYMS

General Acronyms

ACE	Automated Concatenation of Embeddings
AD	Access Data
AI	Artificial Intelligence
AFF	Advanced Forensics Format
APFS	Apple File System
API	Application Programming Interface
APP	Application
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long-Short Term Memory
CD	Compact Disk
CLI	Command Line Interface
CoNLL	Conference of Natural Language Learning
CMD	Command
CNN	Convolutional Neural Network
CPU	Central Process Unit
CR	Coreference Resolution
CRF	Conditional Random Fields
CSV	Comma Separated Values
CWR	Contextualized Word Representation
DB	Database
DFaaS	Digital Forensics as a Service
DocRED	Document-level Relation Extraction Dataset
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately
EM	Entity Mask
ER	Entity Ruler
ERNIE	Enhanced Representation through kNowledge IntEgration
ET	Entity Typing
F1	F1 Score
FTK	Forensic Toolkit
GB	Gigabyte
GLUE	General Language Understanding Evaluation
GPU	Graphical Processing Unit
HAREM	*Avaliação de Reconhecimento de Entidades Mencionadas*
HMTL	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure

General Acronyms

IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IOB	Inside-Outside-Beginning
IPED	*Indexador Processador de Evidência Digital*
ISO	International Organization for Standardization
JS	JavaScript
JSON	JavaScript Object Notation
KG	Knowledge Graph
KNN	K-Nearest-Neighbors
LM	Language Model
LOC	Location
LSTM	Long-Short Term Memory
LUKE	Language Understanding with Knowledge-based Embeddings
MB	Megabyte
MD5	Message-Digest algorithm 5
MISC	Miscellaneous
ML	Machine Learning
MLM	Masked Language Modeling
MNLI	Multi-Genre Natural Language Inference
MTB	Matching The Blanks
MUC	Message Understanding Conference
NE	Named Entity
NEI	Named Entity Input
NER	Named Entity Recognition
NIE	Neural Information Extraction
NLI	Natural Language Inference
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NN	Neural Network
NOTA	None-Of-The-Above
NRE	Neural Relation Extraction
NSP	Next Sentence Prediction
OIE	Open Information Extraction
ORG	Organization
OS	Operating System
OSI	Open Systems Interconnection
PER	Person
PLM	Pre-trained Language Model
PoS	Part of Speech
QA	Question Answering

General Acronyms

RAM	Random Access Memory
RC	Relation Classification
RE	Relation Extraction
RECENT	RElation Classification with ENtity Type restriction
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimized BERT Approach
RTE	Recognizing Textual Entailment
SHA1	Secure Hash Algorithm 1
SNLI	Stanford Natural Language Inference
SOP	Sentence Order Prediction
SQuAD	Stanford Question Answering Dataset
TACRED	TAC Relation Extraction Dataset
TEM	Typed Entity Mask
TL	Transfer Learning
UDF	Universal Disk Format
URL	Uniform Resource Locator
USB	Universal Serial Bus
VHD	Virtual Hard Disk
VMDK	Virtual Machine Disk
WNUT	Workshop on Noisy User-generated Text
XNLI	Cross-lingual Natural Language Inference

1 INTRODUCTION

Text data play an important role in every forensic analysis process. Most of the data in the cyber world is unstructured, consisting of texts, photos and videos. Even though the digital forensic process model is not standardized, there is an abstract-level consensus surrounding it. Kohn *et al.* [1] proposed an overview of the most significant models described over the years, consisting of six processes: *documentation, preparation, incident, incident response, digital forensic investigation* and *presentation*. This thesis focuses on improving the efficiency of the digital forensic investigation process (as shown in Figure 1.1), which is based on several examination and analysis sub-processes. These sub-processes require a great amount of time and effort to be completed and the process as a whole could benefit from a structured computational solution.

Figure 1.1: Digital forensics investigation process.

In this context, Natural Language Processing (NLP) algorithms became a relevant approach to deal with such huge and diverse volume of data and to extract useful insights [2]. More specifically, Named Entity Recognition (NER) systems have been adopted by several languages to extract entities such as locations, organizations and people. These entities alone can lead to the identification of key elements of an investigation, for example. However, a more refined analysis could benefit from the extraction of relations between entities, like the member of an organization, a list of relatives or the location a crime took place.

The recent advances in deep neural networks have enabled researchers to develop more powerful NLP models based on BERT (Bidirectional Encoder Representations from Transformers) [3], which can then be fine-tuned for better performance in specific tasks. Multiple tasks can benefit from this, including NER and RE (Relation Extraction). With a couple of thousand sentences, a new NLP model can be fine-tuned from a pre-trained BERT-based model in any language, overcoming the performance of traditional approaches that do not use neural networks.

Training a new NER or RE model usually requires many annotated data, following either a supervised learning paradigm or a distantly supervised learning paradigm. Some preprocessing techniques are often necessary to prepare the data and achieve better results. Nevertheless, there are tools that already offer functional models for several languages and use cases, like SpaCy [4]. In order to build a customized workflow for specific applications or goals, these tools may not be sufficient nor achieve the desired results in practice. Therefore, a more personalized approach is useful in such cases.

In this work, it is proposed a knowledge-based information retrieval system that combines NER and RE models enhanced by neural networks and transformers architectures. These models with a baseline

setup were compared with new ones obtained from hyper-parameters tuning for English and Portuguese. Preliminary results showed that it was possible to improve a Portuguese NER model on the Paramopama [5] corpus by 2%. By outputting entities and their relations in the form of an interactive graph, it was demonstrated how the pipeline can help to automate the information extraction analysis, not only in the fields of digital forensics and digital investigation, but also for big data analytics and business intelligence applications.

Despite the good preliminary results achieved, the performance of the models proposed in this work is deeply dependent on the quality of the datasets used for training. Additional cleaning and other preprocessing steps are usually required to start developing a good NLP model, and these steps may demand great effort and time. Besides, some scenarios and domain-specific data are more difficult to process, like social media content and chat conversations. The results presented in this thesis for different application scenarios considered structured texts, without any typography or grammar errors, which may not be sufficient for some other applications. Long texts tend to present complex relations between named entities, which are not always in the same sentence, making it difficult to predict and extract semantic relationships. Extracting information for multilingual texts is also a great challenge. In this work, English and Portuguese NLP models and applications were prioritized over other languages, mainly due to the relevance of the English language worldwide and the lack of support for Portuguese in many NLP applications.

1.1 MOTIVATION

Digital forensics tools for analyzing textual documents are often restricted to some application scenarios, such as the identification of key terms and languages, as well as the identification of named entities. In the latter case, for example, NLP models are used for the recognition and extraction of such terms. However, the performance of these algorithms is closely related to the training process adopted, including the datasets and hyper-parameters used. In addition, most of the solutions available for commercial use focus on the English language, with few consolidated applications of NER and relationships extraction for Portuguese, for example.

In this sense, this work proposes a pipeline for extracting information from text documents, as a way to expand and diversify NLP applications in the context of cyber forensics. It is expected that new NLP models with performance at the state-of-the-art level can optimize the analysis process as a whole, by automating the retrieval of relevant information elements including the correct identification and classification of entities, as well as the relationships between them.

1.2 OBJECTIVES

The main objective of this work is to propose an information extraction pipeline, whose goal is to automate part of the forensics analysis process by the development and application of Natural Language Processing models. The concept of the pipeline, including methods for evaluating its performance and applying it to different input examples, was carefully designed to make its components modular and flexible in terms of implementation details, making it possible to extend the pipeline's application taking into account other languages, training data and relationship schemas.

1.3 RESEARCH CONTRIBUTIONS

Some of the results obtained through this work are also presented in the paper "Natural Language Processing Applied to Forensics Information Extraction with Transformers and Graph Visualization", published in the IEEE (Institute of Electrical and Electronics Engineers) Transactions on Computational Social Systems journal [6]. The main contributions of this work are as follows:

- Definition of relationship schemas between named entities for structured information extraction;
- Automation of part of the cyber forensics analysis process through the application of Natural Language Processing techniques and Pre-trained Language Models;
- Functional implementation of the proposed information extraction pipeline and its integration with an open source forensics software.

1.4 OUTLINE

Besides this introduction, this thesis is divided as follows: Chapter 2 explores the background for digital forensics, pre-trained language models and natural language processing tasks; Chapter 3 proposes the information extraction pipeline concept and its components, alongside the relationship schemas necessary for its implementation; Chapter 4 depicts the baseline setup usage possibilities, using hypothetical scenarios, and it presents the testing methodology for validating the proposal and the testing data results are shown and discussed in terms of evaluation metrics and comparison with state-of-the-art works. Finally, Chapter 5 draws conclusions based on the information presented in this thesis.

2 BACKGROUND AND RELATED WORKS

2.1 DIGITAL FORENSICS

Digital forensics is a branch of forensic science concerned with the use of digital information (produced, stored and transmitted by computers) as source of evidence in investigations and legal proceedings. Digital Forensic Research Workshop has defined digital forensics as "The use of scientifically derived and proven methods toward the preservation, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations" [7].

2.1.1 Investigative Process

Investigative process of digital forensics can be divided into several stages. According to [7] and [8], there are five major stages: preservation, collection, examination, analysis and reporting.

- **Preservation:** The preservation stage corresponds to "freezing the crime scene". It consists in stopping or preventing any activities that can damage digital information being collected. Preservation involves operations such as preventing people from using computers during collection, stopping ongoing deletion processes, and choosing the safest way to collect information;
- **Collection:** The collection stage consists in finding and collecting digital information that may be relevant to the investigation. Since digital information is stored in computers, collection of digital information means either collection of the equipment containing the information, or recording the information on some medium. Collection may involve removal of personal computers from the crime scene, copying or printing out contents of files from a server, recording of network traffic, and so on;
- **Examination:** The examination stage consists in an "in-depth systematic search of evidence" relating to the incident being investigated. The output of examination are data objects found in the collected information. They may include log files, data files containing specific phrases, timestamps, and so on;
- **Analysis:** The aim of analysis is to "draw conclusions based on evidence found";
- **Reporting:** The final stage's objective is to summarize and provide explanation of conclusions.

The aims of preservation and collection are twofold. First they aim to provide examination and analysis with as much relevant information as possible. Second they aim to ensure integrity of the collected information. Preservation is not discussed in this thesis and collection is briefly discussed in Chapters 3 and 4, because this research is primarily concerned with the analysis stage of the investigative process. In the rest of this thesis it is simply assumed that all necessary information has been collected, and that the integrity of the collected information has been preserved. Interested readers can obtain more information about collection and preservation stages of investigative process from different works and guidelines, such

as [9]. An approach to testing correctness of tools used during collection of information is described in [10]. Methods for detecting tampering after collection are described in [11], which are based on checksums and one-way hashing of collected information.

2.1.2 Digital Analysis Types

A digital investigation may encounter many formats of digital data and therefore there exist several types of analysis. The different analysis types are based on interpretation, or abstraction, layers, which are generally part of the data's design. For example, consider the data on a hard disk, which has been designed with several interpretation layers. The lowest layer may contain partitions or other containers that are used for volume management. Inside of each partition there is data that has been organized into a file system or database (DB). The data in a file system is interpreted to create files that contain data in an application-specific format. Each of these layers has its own analysis techniques and requirements. Examples of common digital analysis types include [12]:

- **Media Analysis:** The analysis of the data from a storage device. This analysis does not consider any partitions or other operating system specific data structures. If the storage device uses a fixed size unit, such as a sector, then it can be used in this analysis;
- **Media Management Analysis:** The analysis of the management system used to organize media. This typically involves partitions and may include volume management or systems that merge data from multiple storage devices into a single virtual storage device;
- **File System Analysis:** The analysis of the file system data inside of a partition or disk. This typically involves processing the data to extract the contents of a file or to recover the contents of a deleted file;
- **Application Analysis:** The analysis of the data inside of a file. Files are created by users and applications and the format of the contents are application specific;
- **Network Analysis:** The analysis of data on a communications network. Network packets can be examined using the OSI (Open Systems Interconnection) model to interpret the raw data into an application-level stream.

Application (APP) analysis is a large category of analysis techniques because there are so many application types. Some of the more common ones are listed here:

- **OS Analysis:** An operating system is an application, although it is a special application because it is the first one that is run when a computer starts. This analysis examines the configuration files and output data of the OS (Operating System) to determine what events may have occurred;
- **Executable Analysis:** Executables are digital objects that can cause events to occur and they are frequently examined during intrusion investigations because the investigator needs to determine what events the executable could cause;
- **Image Analysis:** Digital images are the target of many digital investigations because some are contraband. This type of analysis looks for information about where the picture was taken and who or what is in the picture. Image analysis also includes examining images for evidence of steganography;

- **Video Analysis:** Digital video is used in security cameras and in personal video cameras and web-cams. Investigations of on-line predators can sometimes involve digital video from web-cams. This type of analysis examines the video for the identify of objects in the video and location where it was shot;

- **Text Analysis:** Digital documents contain huge amounts of precious data for an investigation. This type of analysis searches for information about people, organizations, locations, objects and dates of events and aims to establish a relationship to make sense of this data.

2.1.3 Search Techniques

This group of techniques searches collected information to answer the question whether words of given type, such as names, or pictures of certain kind, are present in the collected information. According to the level of search automation, this thesis classifies techniques into manual browsing and automated searches. Automated searches include keyword search, regular expression search, approximate matching search and machine learning search.

- **Manual Browsing:** Manual browsing means that the forensic analyst browses collected information and singles out objects of desired type. The only tool used in manual browsing is a viewer of some sort. It takes a data object, such as file or network packet, decodes the object and presents the result in a human-comprehensible form. Most investigations collect large quantities of digital information, which makes manual browsing of the entire collected information unacceptably time consuming;

- **Keyword Search:** Keyword search is an automatic search of digital information for data objects containing specified keywords. It is the earliest and the most widespread technique for speeding up manual browsing. The output of keyword search is the list of found data objects (or locations thereof). Keywords are rarely sufficient to specify the desired type of data objects precisely. As a result, the output of keyword search can contain false positives, objects that do not belong to the desired type even though they contain specified keywords. To remove false positives, the forensic scientist has to manually browse the data objects found by the keyword search. Another problem with keyword search is false negatives. They are objects of the desired type that are missed by the search. False negatives occur if the search utility cannot properly interpret the data objects being searched. It may be caused by encryption, compression, or the inability of the search utility to interpret novel data format;

- **Regular Expression Search:** Regular expression search is an extension of keyword search. Regular expressions provide a more flexible language for describing objects of interest than keywords [13]. Apart from formulating keyword searches, regular expressions can be used to specify searches for e-mail addresses and files of specific type. Regular expression searches suffer from false positives and false negatives just like keyword searches, because not all types of data can be adequately defined using regular expressions;

- **Approximate Matching Search:** Approximate matching search is a development of regular expression search. It uses a matching algorithm that permits character mismatches when searching for keywords or patterns. The user must specify the degree of mismatches allowed. Approximate

matching can detect misspelled words, but mismatches also increase the number of false positives. One of the utilities used for approximate search is *agrep*, described in [14];

- **Machine Learning Search:** Machine learning models have the capability to learn language representations and execute different natural language processing tasks, including the detection of names, companies, locations and dates. ML models are useful to automate searches and decrease the time required for human review. This work focuses on this search technique and describes how to prepare ML models to process text and extract relevant information.

2.1.4 Investigative Reconstruction

Crime is rarely committed in a straightforward or clear manner. It can be difficult to prove what is suspected to have occurred based on the evidence left behind. Crimes can involve an innumerable amount of varying factors where only the perpetrator knows all the information. Investigative reconstruction is the piecing together of evidence and information obtained in an investigation in an attempt to understand the events that transpired. Evidence used to reconstruct crime falls into three categories: temporal (when), relational (who, what, where), and functional (how) [15].

2.1.4.1 Temporal Analysis

Temporal analysis involves creating a chronological list of events to help an investigator gain insight into a crime. Digital investigations have an advantage not present in real-world crime scenes: log files. Log files are an incredibly rich source of temporal information because many actions are recorded. By piecing together information from various log files it is often possible to lay out exactly what a perpetrator did or was trying to do. Counter-forensics in this area attempts to destroy the integrity of log files and prevent the temporal analysis of a system.

2.1.4.2 Relational Analysis

Relational analysis involves the determining of where or how an object or person was in relation to other objects or people. Creating a diagram depicting the associations between people and computers can help clarify what has occurred. In a cybercrime case, for example, a link analysis might show how the suspect obtained information about the victim. This knowledge could be used to prevent additional information from being obtained, to present false information as a trap to incriminate the suspect, or just to monitor and gather evidence. As the number of entities and links increases, it becomes harder to identify important connections, especially if using only manual techniques. Relational analysis is the primary object of this work and different application examples of this type of analysis, including visualization of links and entities, are shown and discussed in Chapter 4.

2.1.4.3 Functional Analysis

This aspect of analysis considers what conditions were necessary for certain aspects of the crime to be possible. For instance, if a suspect is accused of distributing child pornography from his home by mailing CDs (Compact Disks), it would be important to verify that the suspect's computer had the capabilities to produce such CDs in the first place. Similarly, if a suspect's computer does not have the correct *codec* to

open and view an incriminating `.avi` file, then that file is not useful as incriminating evidence.

2.1.5 Digital Evidence Processor and Indexer: IPED

IPED – Digital Evidence Processor and Indexer [16] (translated from Portuguese) is a tool implemented in Java and originally and still developed by digital forensic experts from Brazilian Federal Police since 2012. Although it was always open source, only in 2019 its code was officially published. Figure 2.1 shows an overview of IPED's user interface.

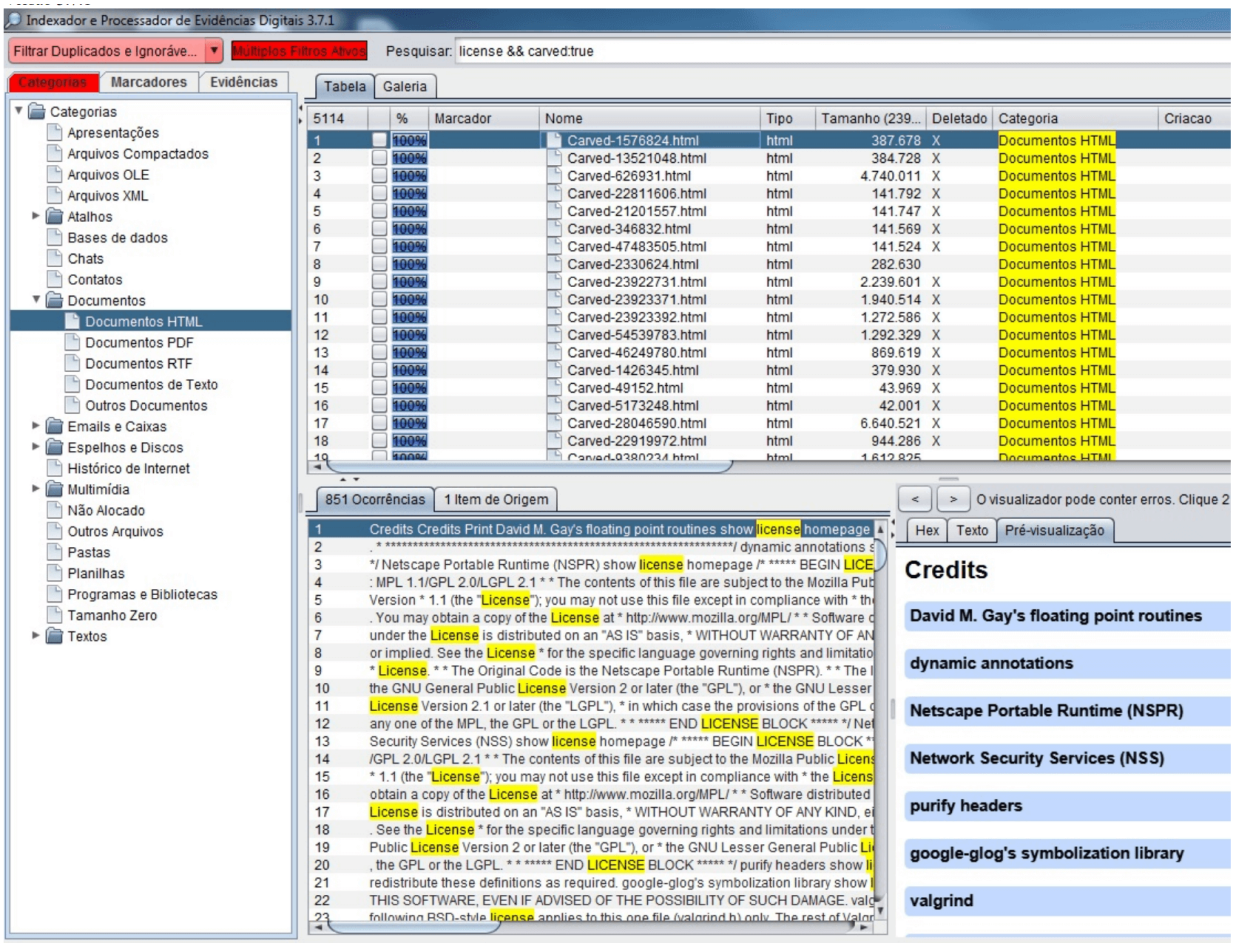

Figure 2.1: IPED's Interface (Source: [17]).

Since the beginning, the goal of the tool was efficient data processing and stability. Some key characteristics of the tool are:

- Command line data processing for batch case creation;
- Multi-platform support, tested on Windows and Linux systems;
- Portable cases without installation, that can run from removable drives;
- Integrated and intuitive analysis interface;

- High multi-thread performance and support for large cases.

Currently, IPED supports several image formats, including the following: AFF (Advanced Forensics Format), VHD (Virtual Hard Disk) and VMDK (Virtual Machine Disk). There is also support for UDF (Universal Disk Format), AD1 (Access Data) and APFS (Apple File System) formats. Some of IPED's several features are listed below:

- Signature analysis;
- Categorization by file type and properties;
- Recursive container expansion of dozens of file formats;
- Regex searches with optional script validation for credit cards, e-mails, URLs (Uniform Resource Locator), money values, Bitcoin, Ethereum, ripple wallets, and more;
- File content and metadata indexing and fast searching, including unknown files and unallocated space;
- Optical Character Recognition;
- Encryption detection for known formats and using entropy test;
- Detection for more than 70 languages;
- Named Entity Recognition with Stanford CoreNLP;
- Customizable filters based on any file metadata;
- Similar document search with configurable threshold;
- Powerful file grouping (clustering) based on any metadata;
- Extensible with JavaScript and Python scripts;
- External command line tools integration for file decoding;
- Audio Transcription;
- Graph analysis for communications (calls, e-mails, instant messages and more);
- Web API (Application Programming Interface) for searching remote cases, get file metadata, raw content, decoded text, thumbnails and posting bookmarks;
- Creation of bookmarks/tags for interesting data;
- HTML (HyperText Markup Language) or CSV (Comma Separated Values) reports and portable cases with tagged data.

2.2 PRE-TRAINED LANGUAGE MODELS

This section introduces some of the main transformer-based language models from the last years, describing their architectures, applications and characteristics. Table 2.1 shows a comparison between some of the most relevant language models mentioned throughout this work in terms of number of layers and parameters, model size and target language.

Table 2.1: Comparison between Transformer-based Models.

Model	Language	Size	Layers	Hidden Size	Parameters	Reference
BERT	English	Base	12	768	110M	[3]
BERT	English	Large	24	1024	340M	[3]
SpanBERT	English	Base	12	768	110M	[18]
SpanBERT	English	Large	24	1024	340M	[18]
DistilBERT	English	–	6	768	66M	[19]
ALBERT	English	Base	12	768	11M	[20]
ALBERT	English	Large	24	1024	17M	[20]
ALBERT	English	XLarge	24	2048	58M	[20]
BERT	Multilingual	Base	12	768	110M	[19]
BERT	Portuguese	Base	12	768	110M	[21]
BERT	Portuguese	Large	24	1024	335M	[21]
RoBERTa	English	Base	12	768	125M	[22]
RoBERTa	English	Large	24	1024	355M	[22]
DistilRoBERTa	English	–	6	768	82M	[19]
ELECTRA	English	Small	12	256	14M	[23]
ELECTRA	English	Base	12	768	110M	[23]
ELECTRA	English	Large	24	1024	335M	[23]
LUKE	English	Base	12	768	253M	[24]
LUKE	English	Large	24	1024	483M	[24]

Devlin *et al.* [3] explain BERT's model architecture as a multi-layer bidirectional Transformer encoder based on the original implementation described in [25]. The dominating trend in these models is to build complex, deep text representation models, for example, with Convolutional Neural Networks (CNNs) [26] or Long Short-Term Memory (LSTM) networks [27] with the goal of deeper sentence comprehension. While these approaches have yielded impressive results, they are often computationally very expensive, and result in models having millions of parameters (excluding embeddings). The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs (Recurrent Neural Networks) or convolution.

2.2.1 Transformer Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure as described in [25]. Here, the encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, ..., ym) of symbols, one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 2.2, respectively.

Vaswani *et al.* also shows some advantages to choosing self-attention layers over recurrent and convolutional layers. One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required. The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies.

Vaswani *et al.* describes self-attention, sometimes called intra-attention, as an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations.

2.2.1.1 Encoder

The encoder component is a stack of encoders (six of them on top of each other, according to [25]) all identical in structure (yet they do not share weights). Each one has two sub-layers: the first is a multi-head self-attention mechanism, and the second is a simple position-wise fully connected feed-forward network. The encoder's inputs first flow through a self-attention layer (a layer that helps the encoder look at other words in the input sentence as it encodes a specific word). The outputs of the self-attention layer are fed to a feed-forward Neural Network (NN). The exact same feed-forward network is independently applied to each position. The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence. It is added a residual connection around each of the two sub-layers, followed by layer normalization.

2.2.1.2 Decoder

The decoder component is also composed of a stack of identically structured decoders (with the same size of the encoder component). In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. The self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions so it will only consider tokens that were already decoded. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

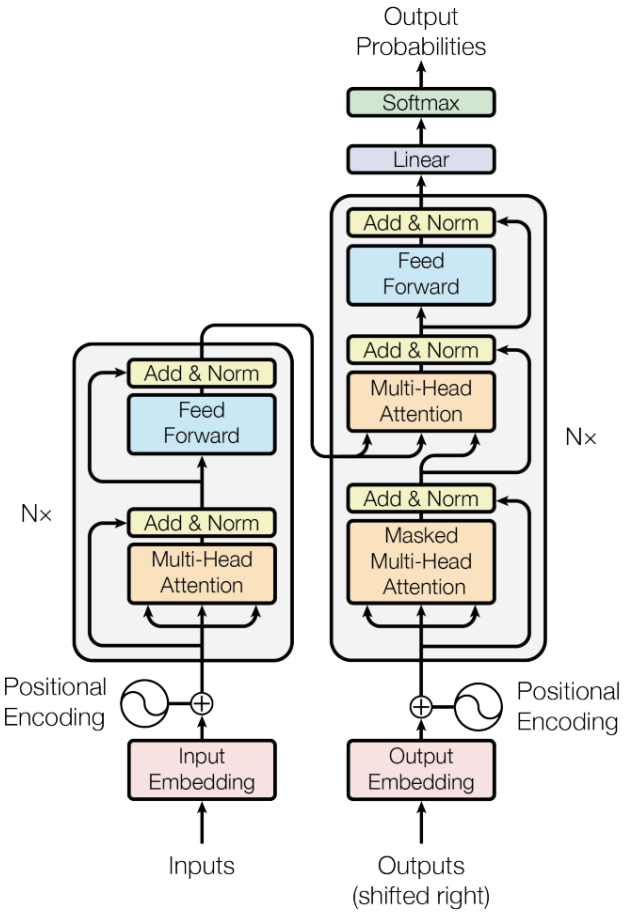

Figure 2.2: The Transformer model architecture (Source: Vaswani *et al.* [25]).

2.2.1.3 Attention

Vaswani *et al.* [25] describes the attention function as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. Two attention mechanisms are explained in [25]: Scaled Dot-Product Attention and Multi-Head Attention (Figure 2.3).

2.2.1.4 Scaled Dot-Product Attention

In this particular attention mechanism, the input consists of queries and keys of dimension d_k, and values of dimension d_v as shown in [25]. The dot products of the query with all keys is computed, each of which are divided by $\sqrt{d_k}$, and a softmax function is applied to obtain the weights on the values, packing all together into a matrix Q (for queries). The keys and values are also packed together into matrices K (for keys) and V (for values). The matrix of outputs is computed as:

$$Attention(Q, K, V) = softmax(\frac{Q.K^T}{\sqrt{d_k}}).V \tag{2.1}$$

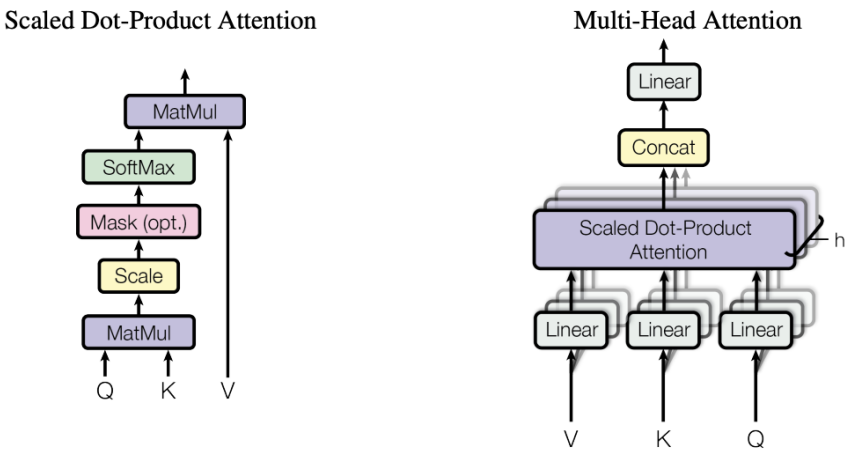

Figure 2.3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel (Source: [25]).

2.2.1.5 Multi-Head Attention

Instead of performing a single attention function with $d_{model} - dimensional$ keys, values and queries, it is beneficial to linearly project the queries, keys and values h times with different, learned, linear projections to d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values it is performed multiple attention functions in parallel, yielding $d_v - dimensional$ output values. Multi-head attention allows the model to jointly attend to information from different representation sub-spaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_n)W^O \tag{2.2}$$

Where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. In [25], the authors employ $h = 8$ parallel attention layers, or heads. For each of these they use $d_k = d_v = d_{model}/h = 64$.

A self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece and byte-pair representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in the input sequence centered around the respective output position. This would increase the maximum path length to $O(n/r)$ [25].

2.2.1.6 Feed-forward networks

In addition to attention sub-layers, each of the layers in the encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically as described in [25]. This consists of two linear transformations with a ReLU (Rectified Linear Unit) activation in between.

$$FNN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.4}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{model} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

2.2.1.7 Positional encoding

In order for the model to make use of the order of the sequence, Vaswani *et al.* explain the need to inject some information about the relative or absolute position of the tokens in the sequence. To this end, it is added *positional encodings* to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed.

The transformer's original positional encoding scheme has two key properties. First, every position has a unique positional encoding, allowing the model to attend to any given absolute position. Second, any relationship between two positions can be modeled by an transform between their positional encodings. The positional encodings take the form:

$$PE_{(pos,2i)} = sin(\frac{pos}{1000^{\frac{2i}{d_{model}}}}) \tag{2.5}$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{1000^{\frac{2i}{d_{model}}}}) \tag{2.6}$$

Where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid.

Shiv *et al.* [28] explain that positional encodings address the power limitations of bag-of-words representations by upgrading the bag of words to a bag of annotated words. The transformer's core attention mechanism is order-agnostic, treating keys as a bag. The calculations performed on any given element of a sequence are entirely independent of the order of the rest of that sequence in that layer. This leaves most of the work of exploiting positional information to the positional encodings showed by [25], although decoder-side self-attention masking and auto-regression also play a role.

2.2.2 BERT

As showed in [3], BERT is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of NLP tasks. BERT is designed to pre-train deep bidirectional represen-

tations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as Question Answering (QA), Named Entity Recognition (NER) and Natural Language Inference (NLI), without substantial task-specific architecture modifications.

There are two steps in this framework, explained in [29]: pre-training and fine-tuning (Figure 2.4). During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture.

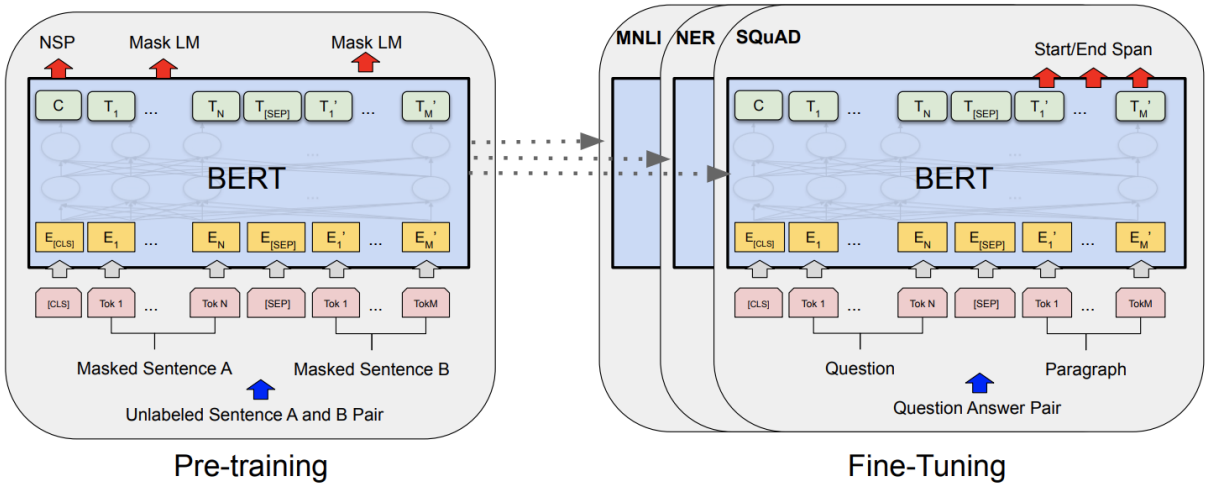

Figure 2.4: BERT's pre-training and fine-tuning architectures (source: [3]).

Devlin *et al.* [3] explains the difference between pre-training BERT and others models that used traditional left-to-right or right-to-left language models for pre-training. BERT represents inputs using three different levels of information (Figure 2.5), including token embeddings, segment embeddings and position embeddings, and it uses two unsupervised tasks: Masked Language Modeling (MLM) and Next Sentence Prediction(NSP):

- **Masked Language Modeling:** In order to train a deep bidirectional representation, some percentage of the input tokens (15% for BERT) is masked at random, and then the task consists of predicting the words (tokens) from the original corpus corresponding to those masked tokens.This procedure is called Masked Language Modeling (MLM);

- **Next Sentence Prediction:** Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationships, the model is pre-trained based on a binary classification task that consists of detecting whether a sentence A follows a sentence B in the text, given a pair of sentences A and B. Specifically, when choosing the sentences A and B for each pre-training example, 50% of

the time B is the actual next sentence that follows A, and 50% of the time it is a random sentence from the corpus.

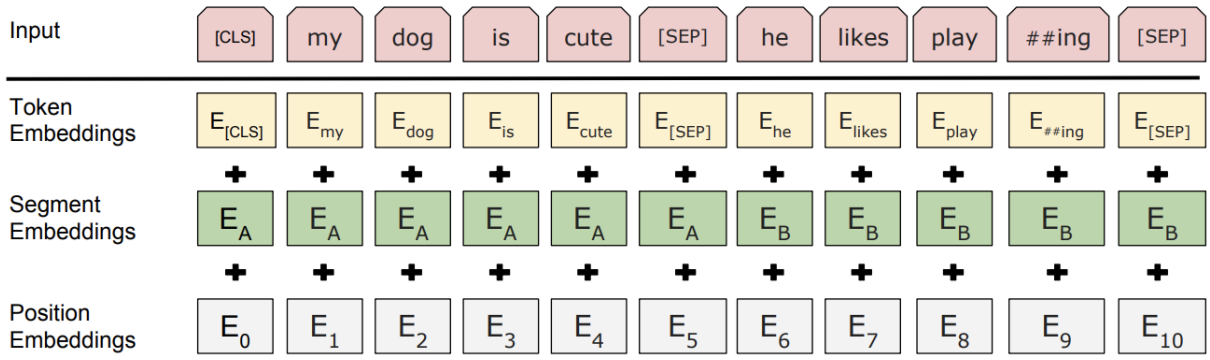

Figure 2.5: BERT's input representation (source: [3]).

This way, the model learns an inner representation of the language that can then be used to extract features useful for downstream tasks. The raw model can be used for either masked language modeling or next sentence prediction, but it is mostly intended to be fine-tuned on a downstream task. Listing 2.1 shows an application of BERT's MLM for masked token prediction given an arbitrary sentence:

Listing 2.1: BERT's Masked Language Modeling usage example.

```
>>> from transformers import pipeline
>>> unmasker = pipeline('fill-mask', model='bert-base-uncased')
>>> unmasker("The [MASK] is on the table.")

[{'sequence': '[CLS] the coffee is on the table. [SEP]',
  'score': 0.08524157851934433,
  'token': 4157,
  'token_str': 'coffee'},
 {'sequence': '[CLS] the phone is on the table. [SEP]',
  'score': 0.05590500310063362,
  'token': 3042,
  'token_str': 'phone'},
 {'sequence': '[CLS] the food is on the table. [SEP]',
  'score': 0.050095196813344955,
  'token': 2833,
  'token_str': 'food'},
 {'sequence': '[CLS] the money is on the table. [SEP]',
  'score': 0.04748654365539551,
  'token': 2769,
  'token_str': 'money'},
 {'sequence': '[CLS] the book is on the table. [SEP]',
  'score': 0.028954964131116867,
  'token': 2338,
  'token_str': 'book'}]
```

As shown in Listing 2.1, a BERT transformer (`bert-base-uncased`) was used to predict a masked token from an example sentence. In this example, the sentence was "*The [MASK] is on the table*". In order to do that, the transformer model expects the masked token to be in a specific format (`[MASK]`), then it will output a finite number of predicted tokens for the particular sentence in question (in the format of a JSON list, in which the predicted token can be retrieved by its code, from the `token` parameter, or by its string, from the `token_str` parameter), along with the scores for each possibility (`score` parameter). For this example, five predictions were made and shown in descending order of score, with "coffee" being the first predicted token and "book" being the last. The `[CLS]` and `[SEP]` tokens from the `sequence` parameter represent the beginning and the end of a sentence, respectively.

Training a model for a specific domain can benefit a fine-tuning sub-task with a specific vocabulary. As shown in different works, such as SciBERT [30], ClinicalBERT [31] and BioBERT [32], it is possible to use a pre-trained model in one language and feed it with a non-labeled domain-specific database, using unsupervised deep learning methods to specialize the model. This model can then be fine-tuned for specific sub-tasks and achieve an improvement over the baseline BERT model without advanced training for the specific domain.

This process can be done in multiple ways: it is possible to train from scratch, using the domain-specific dataset to train all layers from the model, or choosing to re-train a pre-trained model using a domain-specific dataset. SciBERT and ClinicalBERT were trained using the original code from BERT and BioBERT used a pre-trained model in English language to take vantage from weights and the vocabulary. In all cases cited above, training a BERT model using a domain-specific dataset yielded better results, outperforming BERT for general purpose on downstream-tasks for the chosen domain.

2.2.3 SpanBERT

SpanBERT [18] is a pre-training method that is designed to better represent and predict spans of text. This approach extends BERT by (1) masking contiguous random spans, rather than random tokens, and (2) training the span boundary representations to predict the entire content of the masked span, without relying on the individual token representations within it. Figure 2.6 shows and illustration of SpanBERT's training approach.

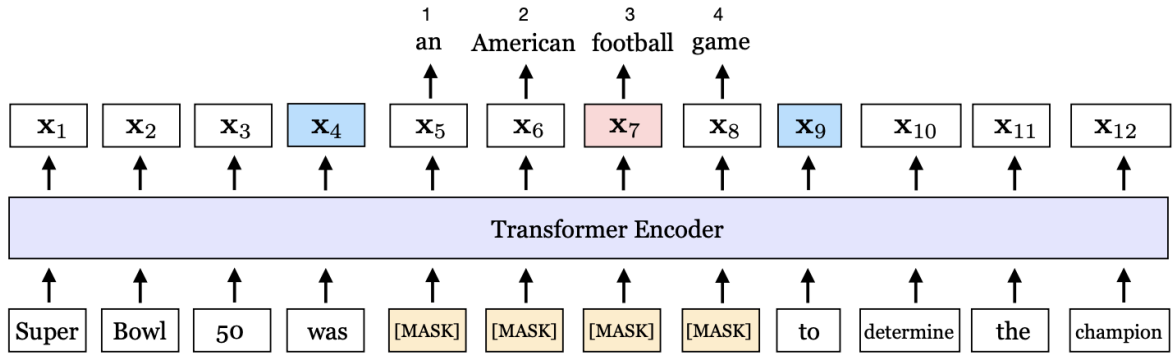

Figure 2.6: An illustration of SpanBERT training. The span *an American football game* is masked. The span boundary objective (SBO) uses the output representations of the boundary tokens, x_4 and x_9 (in blue), to predict each token in the masked span. The equation shows the MLM and SBO loss terms for predicting the token, *football* (in pink), which as marked by the position embedding p_3, is the *third* token from x_4. (Source: [18]).

SpanBERT consistently outperforms BERT and other better-tuned baselines, with substantial gains on span selection tasks such as question answering and coreference resolution. In particular, with the same training data and model size as BERT-large, this single model obtains 94.6% and 88.7% F1 on SQuAD (Stanford Question Answering Dataset) 1.1 and 2.0 respectively. It achieves a new state of the art on the OntoNotes coreference resolution task (79.6% F1), strong performance on the TACRED (TAC Relation Extraction Dataset) relation extraction benchmark, and even gains on GLUE [33].

2.2.4 DistilBERT

DistilBERT is a small, fast, cheap and light Transformer model pre-trained with knowledge distillation. It has 40% less parameters than bert-base-uncased , runs 60% faster while preserving over 95% of BERT's performances as measured on the General Language Understanding Evaluation (GLUE) language understanding benchmark [19]. Knowledge distillation [34] [35] is a compression technique in which a compact model (the student) is trained to reproduce the behaviour of a larger model (the teacher) or an ensemble of models. In their work, the student (DistilBERT) has the same general architecture as BERT (the master). The token-type embeddings and the pooler are removed while the number of layers is reduced by a factor of 2. Most of the operations used in the Transformer architecture (linear layer and layer normalisation) are highly optimized in modern linear algebra frameworks and their investigations showed that variations on the last dimension of the tensor (hidden size dimension) have a smaller impact on computation efficiency (for a fixed parameters budget) than variations on other factors like the number of layers. Thus the authors focus on reducing the number of layers. DistilBERT was trained on the same corpus as the original BERT model: a concatenation of English Wikipedia and Toronto Book Corpus [36].

2.2.5 ALBERT

BERT is an expensive model in terms of memory and time consumed on computations, even with GPU. The original BERT contains 110M parameters to be fine-tuned, which takes a considerable amount of time to train the model and excellent memory to save the model's parameters. Therefore, a recent article introduces a new version of BERT named ALBERT [20]. The authors of ALBERT claim that their model brings an 89% parameter reduction compared to BERT with almost the same performance on the benchmark. The main innovations of ALBERT compared to BERT are the following:

- **Cross-layer parameter sharing:** This is the most significant change in BERT architecture that created ALBERT. ALBERT architecture still has 12 transformer encoder blocks stacked on top of each other like the original BERT. Still, it initializes a set of weights for the first encoder that is repeated for the other 11 encoders. This mechanism reduces the number of unique parameters, while the original BERT contains a set of unique parameters for every encoder;

- **Embedding Factorization:** The embedding size in BERT is equal to the size of the hidden layer (768 in original BERT). ALBERT adds a smaller size layer between the vocabulary and the hidden layer to decompose the embedding matrix of size |V|x|H| (between the vocabulary of size |V| and a hidden layer of size |H|) into two small matrices of size |V|x|E| and |E|x|H|. This idea reduces the number of parameters between vocabulary and the first hidden layer from O(|V|x|H|) to O(|V|x|E| + |E|x|H|), where |E| is the size of the new embedding layer between the hidden layer and vocabulary;

- **Sentence Order Prediction (SOP):** Predicts "+1" for consecutive pairs of sentences in the same document, and it predicts "-1" if the order of sentences is swapped or sentences are from separate documents. The idea is to replace the NSP loss by SOP loss. SOP loss will leverage topic prediction in BERT to coherence prediction in ALBERT.

2.2.6 RoBERTa

RoBERTa (Robustly Optimized BERT Approach) builds on BERT's language masking strategy, wherein the system learns to predict intentionally hidden sections of text within otherwise unannotated language examples. RoBERTa, which was implemented in PyTorch, modifies key hyper-parameters in BERT, including removing BERT's next-sentence pre-training objective, and training with much larger mini-batches and learning rates. This allows RoBERTa to improve on the masked language modeling objective compared with BERT and leads to better downstream task performance [22]. The authors also explore training RoBERTa with significantly more data than BERT, for a longer amount of time. They used existing unannotated NLP datasets as well as CC-News, a novel set drawn from public news articles. After implementing these design changes, their model delivered a sizable performance improvement on the GLUE benchmark. With a score of 88.5, RoBERTa reached the top position on the GLUE leader-board, matching the performance of the previous leader, XLNet-Large [37]. These results highlight the importance of previously unexplored design choices in BERT training and help disentangle the relative contributions of data size, training time, and pre-training objectives.

The masked language model task is the key to BERT and RoBERTa. However, they differ in how they prepare such masking. BERT relies on randomly masking and predicting tokens. The original BERT implementation performed masking once during data preprocessing, resulting in a single static mask. To avoid using the same mask for each training instance in every epoch, the training data was duplicated 10 times so that each sequence is masked in 10 different ways over the 40 epochs of training. As a consequence, each training sequence was seen with the same mask four times during training.

It is possible to compare this strategy with dynamic masking, where the masking pattern is generated every time a sequence is fed to the model. This becomes crucial when pre-training for more steps or with larger datasets. In this sense, in BERT, the masking is performed only once at data preparation time, and they basically take each sentence and mask it in 10 different ways. Therefore, at training time, the model will only see those 10 variations of each sentence. On the other hand, in RoBERTa, the masking is done during training. Consequently, each time a sentence is incorporated in a mini-batch, it gets its masking done, and thus the number of potentially different masked versions of each sentence is not bounded like in BERT.

Their results show that tuning the BERT training procedure can significantly improve its performance on a variety of NLP tasks while also indicating that this overall approach remains competitive with other alternatives. More generally, this research further demonstrates the potential for self-supervised training techniques to match or exceed the performance of more traditional, supervised approaches.

2.2.7 ELECTRA

Masked Language Modeling (MLM) pre-training methods such as BERT corrupt the input by replacing some tokens with `[MASK]` and then train a model to reconstruct the original tokens. While they produce

good results when transferred to downstream NLP tasks, they generally require large amounts of compute to be effective. As an alternative, it was proposed a more sample-efficient pre-training task called replaced token detection. Instead of masking the input, ELECTRA's approach corrupts it by replacing some tokens with plausible alternatives sampled from a small generator network. Then, instead of training a model that predicts the original identities of the corrupted tokens, a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not was trained. Thorough experiments demonstrate this new pre-training task is more efficient than MLM because the task is defined over all input tokens rather than just the small subset that was masked out. As a result, the contextual representations learned by ELECTRA's approach substantially outperform the ones learned by BERT given the same model size, data, and compute.

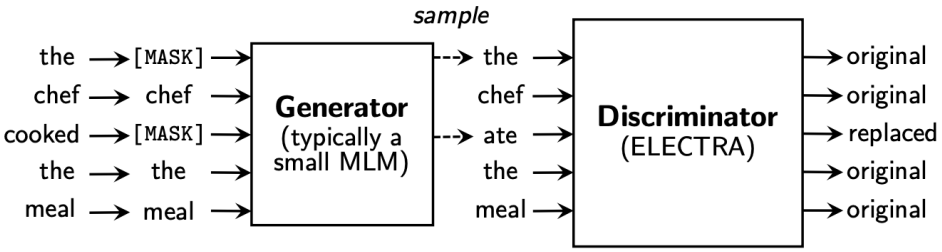

Figure 2.7: ELECTRA's pre-training architecture (source: [23]).

In BERT, the input is replaced by some tokens with `[MASK]` and then a model is trained to reconstruct the original tokens. In ELECTRA, instead of masking the input, the approach corrupts it by replacing some input tokens with plausible alternatives sampled from a small generator network (Figure 2.7).

2.2.8 LUKE

Entity representations are useful in natural language tasks involving entities. Based on this premise, a new pretrained approach was proposed, based on the bidirectional transformer, called LUKE (Language Understanding with Knowledge-based Embeddings). This proposed model treats words and entities in a given text as independent tokens, and outputs contextualized representations of them through a mechanism known as Contextualized Word Representations (CWR). The pretraining task is based on the masked language model of BERT and it involves predicting randomly masked words and entities in a large annotated corpus retrieved from Wikipedia. Since entities are treated as tokens, LUKE can directly model the relationships between entities, which makes it a strong choice for entity-related tasks, such as named entity recognition and relation extraction (as shown in Figure 2.8).

LUKE also introduces a new entity-aware self-attention mechanism, an effective extension of the original attention mechanism of the transformer, which considers the type of the tokens (words or entities) when computing attention scores. LUKE obtains state-of-the-art results on several datasets, including Open Entity (Entity Typing), TACRED (Relation Extraction), CoNLL-2003 – Conference of Natural Language Learning (Named Entity Recognition) and SQuAD (Question Answering). It is currently one of the best options for supervised NLP on these particular tasks.

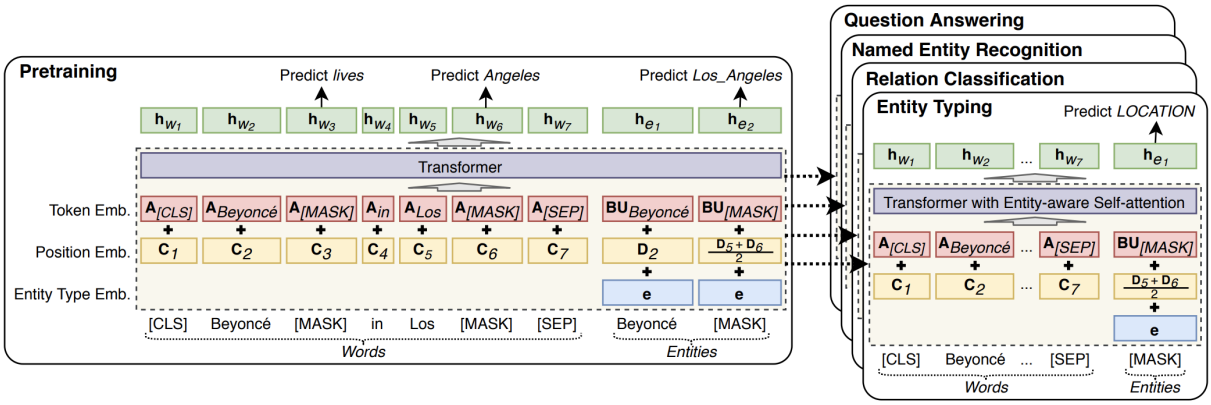

Figure 2.8: Architecture of LUKE using the input sentence "*Beyonce lives in Los Angeles*". LUKE outputs contextualized representation for each word and entity in the text. The model is trained to predict randomly masked words (e.g., *lives* and *Angeles* in the figure) and entities (e.g., *Los_Angeles* in the figure). Downstream tasks are solved using its output representations with linear classifiers (Source: [24]).

2.3 NATURAL LANGUAGE PROCESSING

2.3.1 Information Extraction

Information Extraction is a task of automatically extracting structured information from unstructured or semi-structured documents. In most of the cases, this activity concerns processing human language texts by means of NLP. Figure 2.9 shows an information extraction example in which unstructured text data is converted into a structured semantic graph. A broad goal of information extraction is to extract knowledge from unstructured data and use that obtained knowledge for various other tasks.

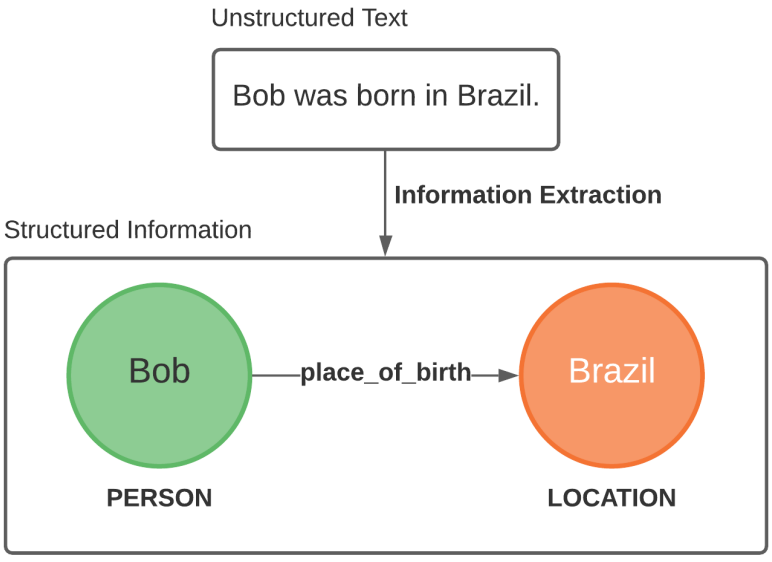

Figure 2.9: Information Extraction example.

2.3.2 Named Entity Recognition

Named Entity Recognition (NER) is a task in information extraction, consisting of finding, extracting and classifying named entities in natural language texts (Figure 2.10). Named entities are objects that can be designated by a proper noun and fit predefined classes such as persons, locations and organizations, among other expressions, like monetary values and dates, for example. NER systems are often used as the first step in question answering, information retrieval, relation extraction and topic modeling, to name a few.

Figure 2.10: Named Entity Recognition example.

This term was first cited at the sixth Message Understanding Conference (MUC) in 1996. One of the first research papers in the field was presented by Lisa F. Rau (1991) at the Seventh IEEE Conference on Artificial Intelligence (AI) Applications [38]. Rau's paper describes a system to "extract and recognize (company) names". Early NER systems were based on handcrafted rules, lexicons, orthographic features and ontologies. These systems were followed by NER systems based on feature-engineering and machine learning, which greatly improved their performance and application scenarios. Modern NER systems are able to recognize not only names of companies, but also several customized categories for general-purpose and domain-specific applications, even though `Person`, `Organization` and `Location` remain the most popular categories in different datasets.

Figure 2.11 shows a two-level entity schema developed by Ding *et al.* [39]. In this schema, for each entity type, there is a corresponding sub-type, which provides additional information and can then be used in further NLP tasks. For example, it might be relevant to differentiate between a politician and an athlete for the `Person` entity type, or to differentiate between a country or a city for the `Location` entity type. This all comes down to how the schema and dataset are organized, and it is usually not an easy task since obtaining enough fine-grained entity types is still a challenging task and requires further research, especially in order to deploy a production-ready solution.

Despite being conceptually simple, NER is not an easy task. The category of a named entity is highly dependent on textual semantics and its surrounding context. Moreover, there are many definitions of named entity and evaluation criteria, introducing evaluation complications.

2.3.3 Coreference Resolution

Coreference Resolution is the task of finding all expressions that refer to the same entity in a text. It is an important step for a lot of downstream NLP tasks, such as information extraction.

Figure 2.12 shows an example text with highlighted coreference clusters. In this example, two mentions were detected (highlighted in blue and pink), one referring to "Nikola Tesla", and another referring to "Continental Edison Company".

There are basically two types of terms commonly used as features for neural coreference resolution

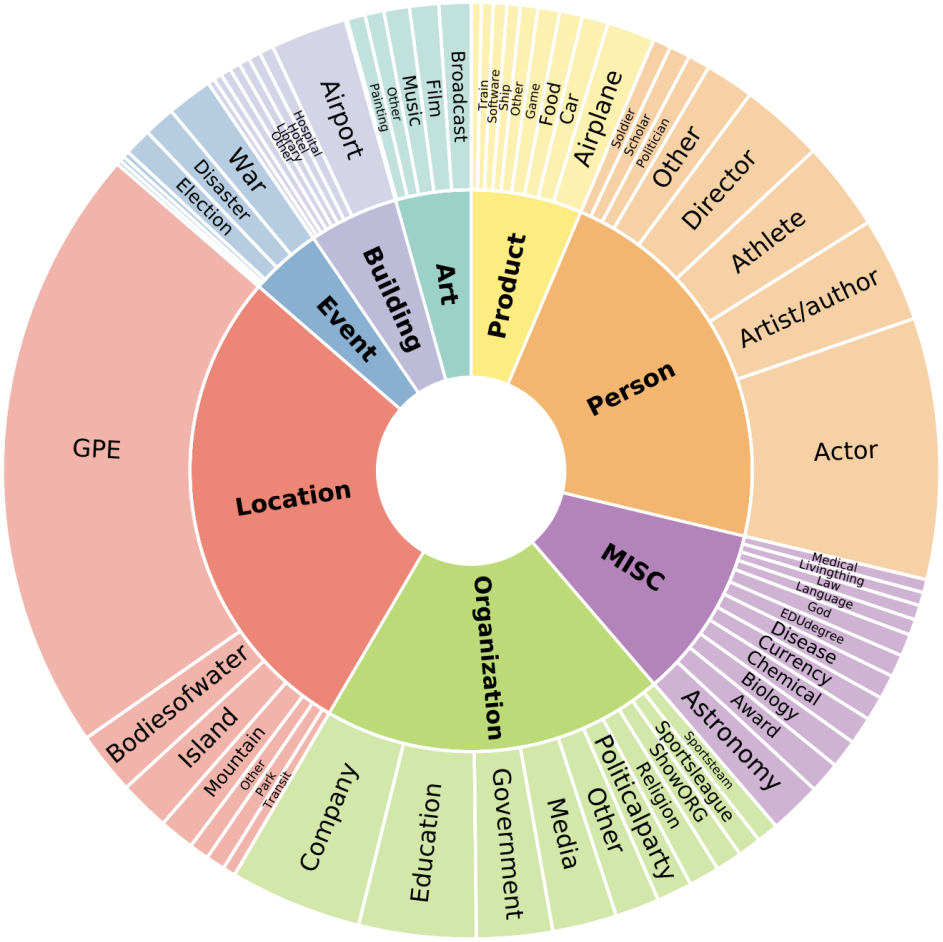

Figure 2.11: An overview of FEW-NERD. The inner circle represents the coarse-grained entity types and the outer circle represents the fine-grained entity types, some types are denoted by abbreviations (Source: [39]).

models: Anaphoric references and Cataphoric references. Anaphoric references occur when a word refers back to other entities in the text for its meaning. For example, in the sentence "Paul went to the concert. He said it was an amazing experience.", **He** refers to *Paul* and **it** refers to *the concert*. Cataphoric references, in turn, occur when a word refers to entities later in text. For example, in the sentence "Every time I visit her, my mother celebrates.", **her** refers to *my mother*.

The coreference task can be divided into two sub-tasks:

- **Mention Detection:** In this sub-task, the main objective is to find all the candidate spans (tokens) referring to entities. Usually these spans are either pronouns, named entities or noun phrases;
- **Mention Clustering:** Once the mentions are detected, the goal of the second sub-task is to identify which ones refer to the same entity. Then, these common mentions are merged into a cluster corresponding to the entities presented in the text.

Algorithms which resolve coreferences commonly look for the nearest preceding mention that is compatible with the referring expression. Instead of using rule-based dependency parse trees, neural networks can also be trained which take into account word embeddings and distance between mentions as features.

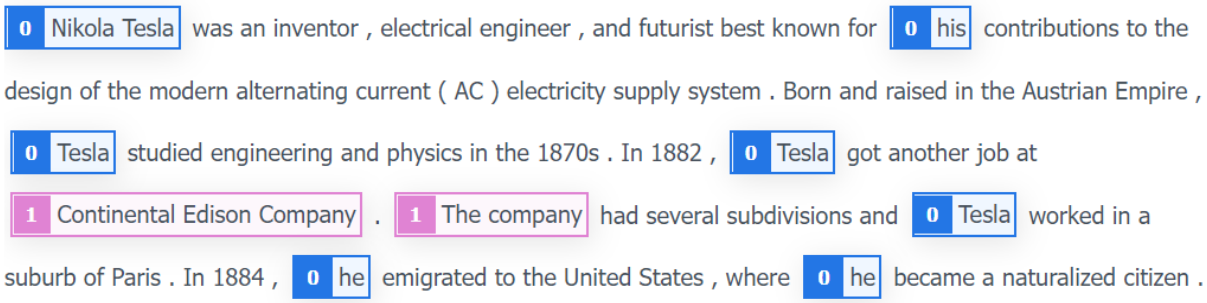

Figure 2.12: Coreference clusters visualization example.

2.3.4 Relation Extraction

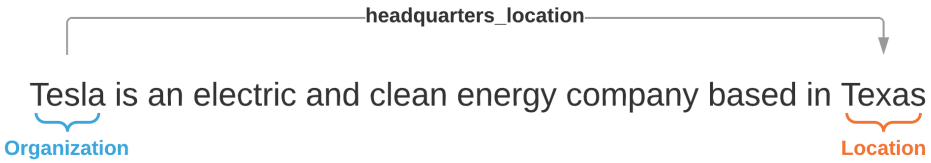

Figure 2.13: Relation Extraction example.

Relation Extraction aims to predict relations between two or more entities from plain text. The sentence "Tesla is an electric and clean energy company based in Texas" (Figure 2.13) contains two named entities and a relation that can be represented in a triple: `(Tesla, headquarters_location, Texas)`. RE can extract structured information from several different sources that can then be applied on specific downstream tasks. In order to improve the performance of such relevant applications, Neural Relation Extraction (NRE) models have been proposed and research in this field gained attention in the last years.

Different techniques have been proposed to tackle the problem of detecting and extracting relationships from textual documents. Figure 2.14 depicts my proposed taxonomy for organising such techniques. These techniques can be divided in two main branches, Traditional Information Extraction and Open Information Extraction [40].

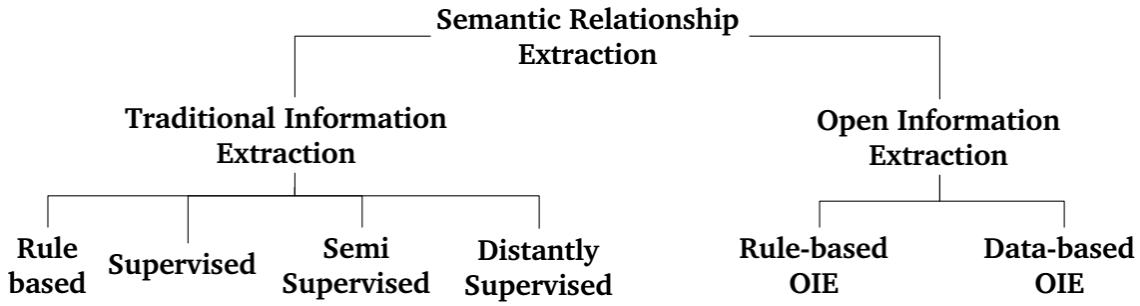

Figure 2.14: Relationship Extraction methods (Source: [40]).

Traditional RE techniques extract relationship instances that belong to a predefined set of relationship types. These techniques include:

- **Rule-based** approaches, which usually aim at extracting one type of relationship by relying on manually-crafted rules. These were the first to be devised for extracting relationships from text;
- **Supervised** approaches, which are based on manually annotated documents. For each pair of named-entities in a sentence, a label indicates the type of relationship between the two entities. An annotated collection of documents is used to train classifiers. Then, for any given sentence, a trained classifier can detect the presence of a relationship type;
- **Semi-Supervised** approaches, which make use of known relationships to iteratively extract new relationships. From the textual contexts of seed relationships, the approaches derive patterns, which are used in turn to derive new relationships;
- **Distantly Supervised** approaches use a knowledge base of known relationships to automatically collect large amounts of training data. The collected data is used to train RE classifiers. If a relation is expressed between two entities in a knowledge base, there is a high probability that the same relationship holds for a given sentence where those two same entities are referred. A supervised classifier can then be trained after collecting a large number of these sentences.

Another approach is Open Information Extraction (OIE), introduced by Etzioni *et al.* [41]. OIE is suited when the target relations are unknown and the textual data is heterogeneous. OIE is mainly directed to perform RE over massive and heterogeneous web corpora which are, in which the relations of interest are unanticipated, and their number can be large. OIE techniques typically make a single pass over a corpus and extract a large set of relational triples without requiring any human input. OIE can be divided into two main categories, data and rule-based:

- **Rule-based OIE** relies on hand-crafted patterns from Part of Speech (PoS)-tagged text or rules operating on dependency parse trees;
- **Data-based OIE** generates patterns based on training data represented by means of dependency tree or PoS-tagged text.

There are different scenarios to which NRE models can be applied (Figure 2.15), including the following:

- **Sentence-level RE:** A conventional method that usually handles RE in a supervised learning paradigm and extracts a predefined relation between a pair of entities present in a sentence;
- **Bag-level RE:** It is a variation from the sentence-level approach in which the same pair of named entities mentioned in different sentences are put together into an entity-pair bag, considering that there is a high probability that a relation can be extracted for this pair of entities from one or more sentences in the bag [43] [44];
- **Document-level RE:** A complex RE scenario that takes into account the fact that some entities in documents often present inter-sentence relations, which are not necessarily retrieved from a sentence-level or bag-level RE approach. This scenario is not widely explored yet and remains an open problem for future research and exploration, as discussed in Yao *et al.* [45].

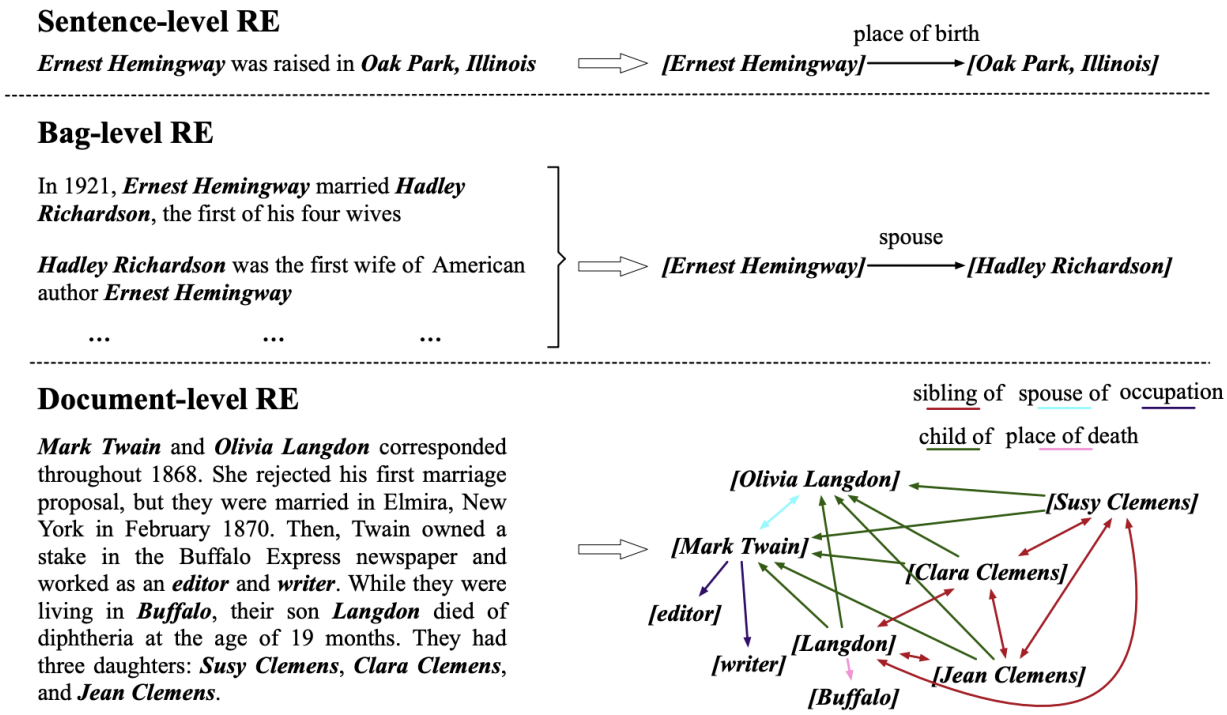

Figure 2.15: Relation Extraction application scenarios (Source: [42]).

Recently, the idea of meta-learning is proposed [46], which encourages the models to learn fast-learning abilities from previous experience and rapidly generalize to new concepts [47]. Many meta-learning models achieve state-of-the-art on several few-shot benchmarks [48]. Nevertheless, most of these works concentrate on image classification, whereas many works in NLP focus on zero-shot and semi-supervised scenarios, which incorporate extra information to classify objects never appearing in the training sets. In a few-shot approach, however, models should be able to classify objects with few instances without any extra information (Figure 2.16).

In this sense, many efforts have been devoted to formalize Relation Classification (RC) as a Few-Shot problem [49]. In a few-shot RC, the intent is to obtain a function $F : (\mathcal{R}, \mathcal{S}, x) \mapsto y$. Here $\mathcal{R} = \{r_1, ..., r_m\}$ defines the relations that the instances are classified into. $\mathcal{S}$ is a support set defined as:

$$\begin{aligned} \mathcal{S} = \{&(x_1^1, r_1), (x_1^2, r_1), ..., (x_1^{n_1}, r_1), \\ &..., \\ &(x_m^1, r_m), (x_m^2, r_m), ..., (x_m^{n_m}, r_m)\} \end{aligned} \tag{2.7}$$

including n_i instances for each relation $r_i \in \mathcal{R}$. For relation classification, a data instance x_i^j is a sentence accompanied with a pair of entities. The query data x is an unlabeled instance to classify, and $y \in \mathcal{R}$ is the prediction of x given by F. Recent researches on few-shot learning adopt a N way K shot setting, where

$$N = m = |\mathcal{R}|, K = n_1 = ... = n_m \tag{2.8}$$

which means that, in a 5-way 1-shot setting, for example, there are 5 classes (N) in the supporting set with 1 instance (K) for each class, and the model is asked to classify a test instance into one of these classes.

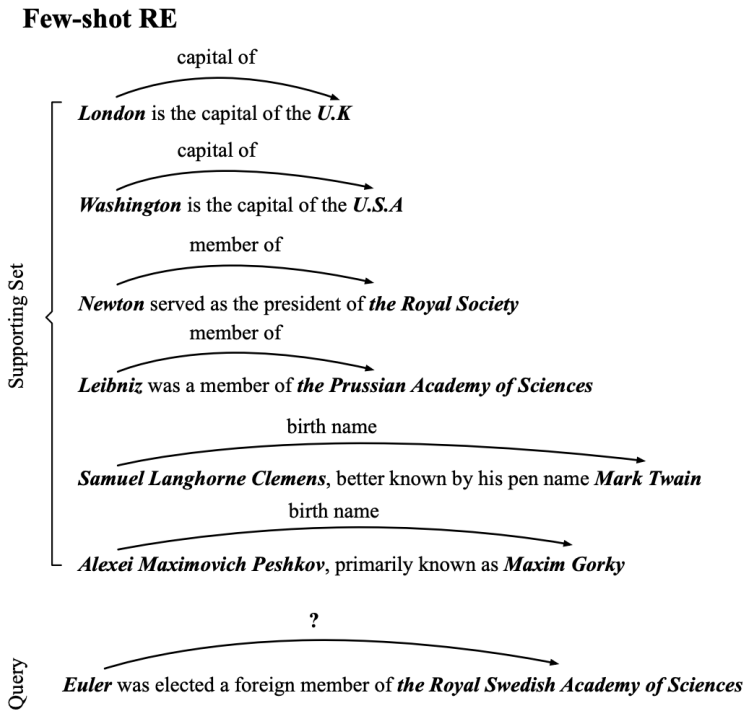

Figure 2.16: Few-Shot Relation Extraction example (Source: [42]).

As the number of instances per class increases, it is natural to deduce that the performance of the model increases as well (a model with a 5-way 10-shot setting usually performs better than another with a 5-way 2-shot setting, for instance). However, increasing the number of classes (N) results in the opposite effect, since the model is now faced with a greater number of possible choices. Thus, the basic idea behind few-shot learning is to train a function that predicts similarity. After training, the learned similarity function can be used for making predictions for unseen queries. It can be used to compare the query with every sample in the support set and calculate the similarity scores, choosing the sample with the highest similarity score as the prediction output.

Another approach to Zero-Shot and Few-Shot Relation Classification is described by Sainz *et al.* [50], which is based on Natural Language Inference (NLI) and entailment. In this paradigm, both subject and object entities in a given sentence are applied in a pre-defined candidate relation template, generating an hypothesis whose test produces an entailment probability for each candidate relation. The goal of this NLI model is, thus, to select the relation with the maximum entailment score among all the candidates (Figure 2.17).

2.3.5 Natural Language Inference

Natural Language Inference (NLI), also known as Recognizing Textual Entailment (RTE), is the task of determining the inference relation between two (short, ordered) texts: entailment, contradiction, or neutral [51]. Determining semantic relationships between sentences is essential for machines that understand and reason with natural language. Much work has been done for inference in English, and not so much for Portuguese. With the success of new deep learning techniques and language models, the inference tasks ascended once again to become one of the most popular topics of Natural Language Processing. Large

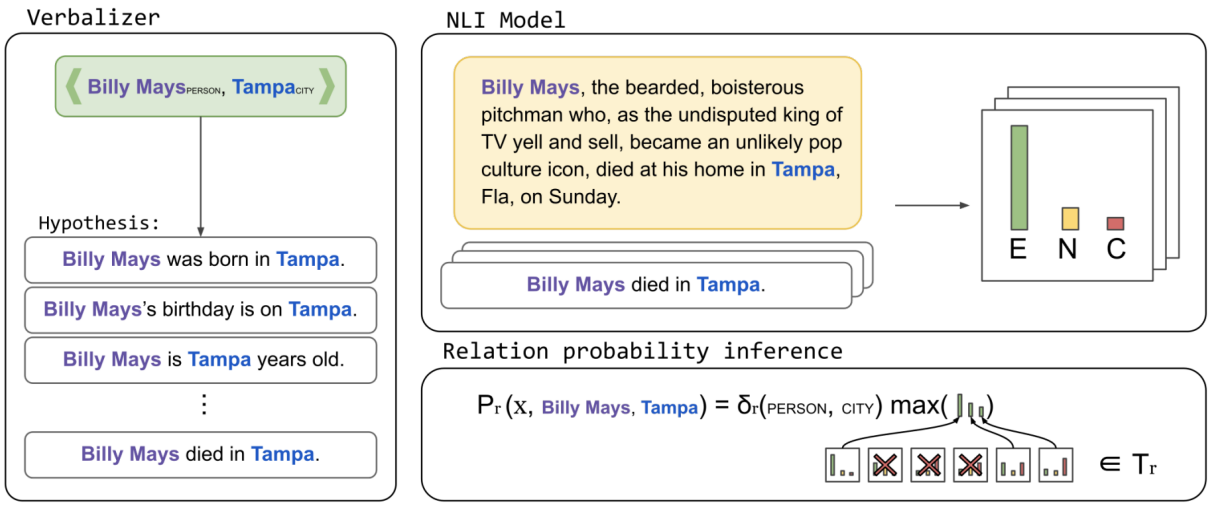

Figure 2.17: General workflow of the entailment-based RE approach (Source: [50]).

datasets have been constructed to serve as input for supervised systems whose goal is to learn how to perform inference.

The Stanford Natural Language Inference (SNLI) corpus [52] (version 1.0) is a collection of 570.000 human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral. It serves both as a benchmark for evaluating representational systems for text, especially including those induced by representation-learning methods, as well as a resource for developing NLP models of any kind. Another work introduced the Multi-Genre Natural Language Inference (MNLI) corpus [53], one of the largest corpora available for natural language inference, improving upon available resources in both its coverage and difficulty by offering data from ten distinct genres of written and spoken English. The work of [54] introduced SICK-BR, a Brazilian Portuguese corpus annotated with inference relations and semantic relatedness between pairs of sentences, consisting of around 10.000 sentence pairs annotated for neutral, contradiction and entailment relations. In an attempt to evaluate cross-lingual sentence representations, the work of [55] proposed the XNLI (Cross-lingual Natural Language Inference) corpus.

Table 2.2 shows some examples to illustrate the judgment of an hypothesis based on a premise, extracted from the SNLI corpus.

Table 2.2: NLI examples (Source: [52]).

Premise	Hypothesis	Judgment
A black race car starts up in front of a crowd of people.	A man is driving down a lonely road.	Contradiction
A smiling costumed woman is holding an umbrella.	A happy woman in a fairy costume holds an umbrella.	Neutral
A soccer game with multiple males playing.	Some men are playing a sport.	Entailment

2.4 RELATED WORKS

Currently, BERT-based architectures have become a standard for several Natural Language Processing applications, including Named Entity Recognition and Relation Extraction, for instance.

2.4.1 Digital Forensics

Nowadays, huge amounts of heterogeneous data have become the new normal all over the cyberworld. Due to this fact, investigating cybercrimes is a task undoubtedly difficult and time consuming. Caviglione *et al.* [56] explore the future of digital forensics and discuss how to maintain societies secure and pursue criminals effectively. Similar aspects are discussed by Ukwen *et al.* [57], addressing a review of NLP-based systems in digital forensics and cybersecurity, to serve as a basis for researchers and practitioners in these fields and also to provide a roadmap for the future.

In this context, technology has been an important ally in the fight against crime. In particular, some softwares help investigators and security experts save time and be more productive, like the IPED (Digital Evidence Processor and Indexer) tool [16]. IPED is an open source software developed by digital forensic experts from Brazilian Federal Police that can be used to process and analyse digital evidence, often seized at crime scenes by law enforcement or in a corporate investigation by private examiners. It offers several functionalities for digital forensics, like language detection, signature analysis, audio transcription and Named Entity Recognition, for example. However, some modules do not yet support other languages or use cases, like the NER module, which does not offer support for Portuguese out of the box. It is expected that the results of this work will serve as a basis for the integration of new NLP models in the tool, in particular to meet the demand for text analysis and information extraction in Portuguese.

In [58], the authors explore different implementations of the digital forensic process and analyse factors that impact the efficiency of this process. They point out that digital investigators should not be tasked with system administrative tasks. In the traditional process, they are responsible for the entire investigation environment, which leads to a lot of administrative overhead. Besides, digital investigators are either underqualified or overqualified for many of the tasks they often perform. That is why it was proposed a Digital Forensics as a Service (DFaaS) model. In the DFaaS setup, digital investigators focus on the forensics tasks (seizing material and extracting data from it), whereas the extracted data is sent to a centralized system that automatically extracts information from the data and give this information back to investigators and analysts (which also proves to be effective especially in the era of big data [59]). Nevertheless, their proposed systems lack descriptions on specific tasks that could be automated by the centralized system. For instance, no details of machine learning solutions or NLP models were mentioned. DFaaS is a new way of working and cooperating, particularly for governmental organizations, that can be adapted to support artificial intelligence, dynamic reporting and other solutions [60].

2.4.2 Named Entity Recognition

In [5], the authors presented a new NE-tagged corpora for Brazilian Portuguese named Paramopama, due to the lack of corpus for this language. They also evaluated the quality of the dataset by measuring Precision, Recall and F-measure of a NER classifier trained on this corpus. Their results show that their dataset has yielded better results than other well-known Portuguese NE-tagged corpora. However, by the

time of their publication, there was not a BERT transformer that could be used to boost performance.

In [61], the authors compared the performance of some well-known NER softwares on the market, such as Stanford NLP, NLTK, Spacy and OpenNLP. Their goal was to address the difficulty encountered by NLP practitioners to clearly and objectively identify which software performs the best, due to the lack of transparency preventing the reproducibility of experiments. The comparison was limited to fewer entity types and languages, mainly English.

Considering the advent of such architectures, in [62] it is proposed a methodology to improve the performance of NER systems in every language. They made experiments over five different datasets, using two high-resourced languages (English and Spanish) and three low-resourced languages (Croatian, Slovene and Finnish), managing to improve the state-of-the-art F-Score for them. However, this work did not contemplate how to tune hyper-parameters for the models nor how to preprocess the datasets.

In [63], it is presented a new NER dataset for Brazilian Portuguese legal texts, consisting of six entity types: Person, Legal cases, Time, Location, Legislation and Organization. The authors also presented a model for NER trained over a LSTM-CRF (Long Short-Term Memory and Conditional Random Fields) architecture with the LeNER-Br dataset, which achieved an average F-Score of 86%. However, their corpus is domain-specific and does not work very well for general-purpose NER applications.

New transformer-based state-of-the-art NER models have emerged in recent years. FLERT [64] introduces a way to capture document-level features to enhance performance by modeling document context. Another approach is proposed by [65], which uses an Automated Concatenation of Embeddings (ACE) mechanism to extract better pre-trained contextualized embeddings of word representations for structured prediction tasks. FLERT is a strong baseline for NER, while the ACE model is the current state-of-the-art for NER in the CoNLL03 benchmark [66], with a F-Score of 94.6%.

2.4.3 Relation Extraction

Early efforts in Relation Extraction had focus on predicting relationships between entities within a given sentence by modeling the possible interactions [67]. This approach does not consider interactions between sentences or distant entities in the text. Extracting relations on document-level is a considerably more difficult task, since several sentences and their relations must be considered [68]. Nevertheless, both approaches are useful in different scenarios and sentence-level relation extraction is usually easier and faster to implement as part of an NLP system [69].

In [69] it is proposed BERT-based models for relation extraction, demonstrating how these models can achieve state-of-the-art performance without external features, serving as the basis for future work on other downstream tasks. However, they did not explore further improvements for the models on the experimental setup, as such for example feature engineering techniques and hyper-parameters tuning.

The work of [70] presents an adaptation of the most recent state-of-the-art few-shot learning methods for Relation Classification (RC), conducting an evaluation of these methods and proposing a new dataset (FewRel) for this task. Figure 2.16 shows an example of an application scenario for few-shot RE.

A more challenging task for few-shot RC is presented by [71] to investigate two aspects of few-shot RC models: (1) If they can adapt to a new domain with only a handful of instances and (2) If they can detect none-of-the-above (NOTA) relations. These sub-tasks aim to address real-world issues, in which the model may be used outside the domain it was trained on and it also should be able to recognize when there

is not a relation between entities. Empirical results show that even the most competitive few-shot models still struggle on this task, especially if compared with humans, which indicates that few-shot relation classification remains an open problem and still requires further research.

In [42], the authors created an open-source and extensible framework to implement neural models for relation extraction. Thanks to their design pattern, they showed it is easy to extend the production-ready available models that come with the framework and to train custom models based on other datasets in any language. Since the focus of their work was on the RE part, they did not release details on how to extract the entities in the text or to build a customized NER system, leaving that part for the users.

Figure 2.18 shows the architecture of OpenNRE, and Listing 2.2 shows an example code of OpenNRE in Python. Based on OpenNRE, one can use only a few lines of code to define, train and evaluate RE models of different scenarios.

Listing 2.2: Example code of OpenNRE (Adapted from: [42]).

```
rel2id = 'relation->id dictionary'
# Define encoder
sentence_encoder = opennre.encoder.BERTEncoder(
    max_length=80,
    pretrain_path='BERT pretrain model path'
)
# Define model
model = opennre.model.SoftmaxNN(sentence_encoder, len(rel2id), rel2id)
# Define framework
framework = opennre.framework.SentenceRE(
    train_path='path of training data',
    val_path='path of validation data',
    test_path='path of test data',
    model=model,
    ckpt='path of checkpoint',
    batch_size=64,
    max_epoch=3,
    lr=2e-5,
    opt='adamw'
)
# Train
framework.train_model()
# Test
framework.load_state_dict(torch.load(ckpt)['state_dict'])
result = framework.eval_model(framework.test_loader)
```

Wang *et al.* [72] proposed an improvement for modeling relations between multiple entities within a document. Current baselines for this task, including the DocRED (Document-level Relation Extraction Dataset) [45] dataset, use Bidirectional LSTM (BiLSTM) models to encode the whole document and are trained from scratch. The authors argue that such simple baselines are not strong enough to model complex interactions between entities. That is why they apply a pre-trained language model (BERT) to provide a stronger baseline for this task. Moreover, they demonstrate that solving this task in phases can further improve the performance. In the first phase, the goal is to predict whether or not two entities have a

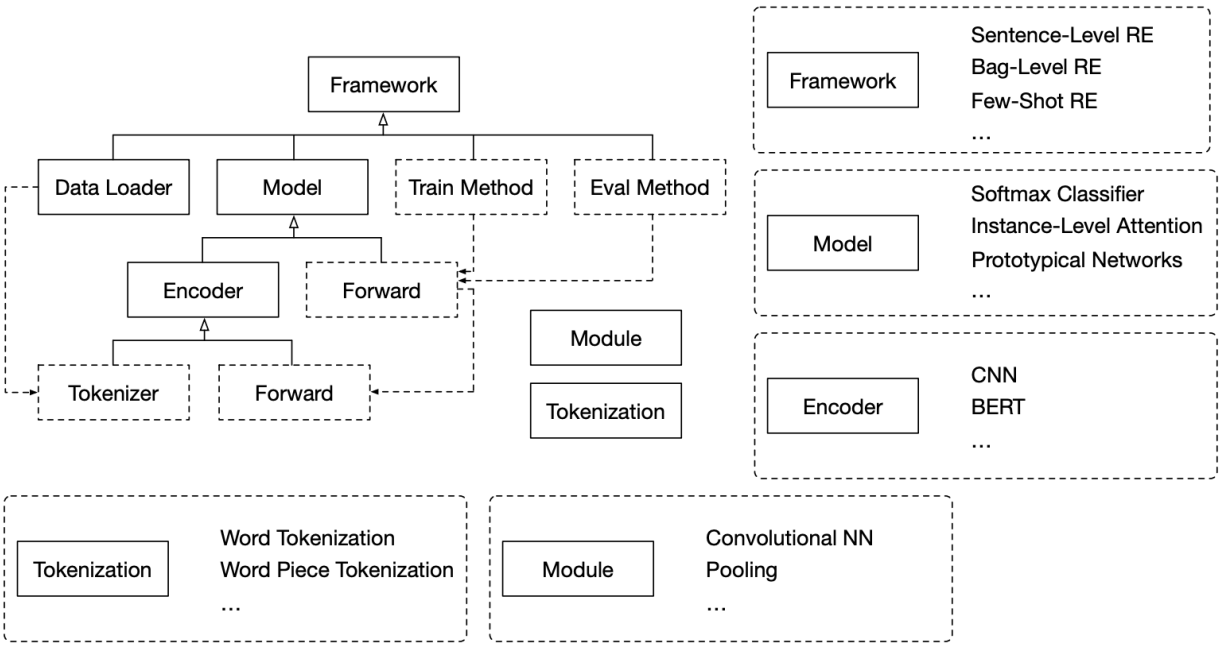

Figure 2.18: Architecture of OpenNRE (Source: [42]).

relation, whereas the second phase's goal is to predict the specific relation. However, due to the lack of accessible document-level RE datasets, the authors experiments were limited to DocRED, which is, in turn, only available for the English language.

Lately, different models have achieved state-of-the-art performance on relation classification benchmarks. In particular, the work of [73] introduces an improved baseline for relation classification by adopting new ways of representing entities. More specifically, the authors show that the Typed Entity Marker (TEM) entity representation technique yields better results when compared with the Entity Marker (EM) technique. This approach outperforms another strong baseline proposed by [49], that introduced a new training setup called *matching the blanks* (MTB), which relies on entity resolution annotations.

A new paradigm for relation classification is proposed by the authors of [74], who argue that the existing methods regard all relations as the candidate relations for a pair of entities, neglecting the restrictions on candidate relations by entity types. Thus, they propose a new model, called RECENT (RElation Classification with ENtity Type restriction), which exploits entity types to restrict candidate relations. This is the current state-of-the-art RE model evaluated on the TACRED benchmark [75], with a F-Score of 75.2%.

2.4.4 NLP with BERT

The era of social networks and big data made room for several NLP applications in the last years. More specifically, automatic text summarization became a fundamental step in order to extract significant information from different sources. In [76], the authors created a topic-aware extractive summarization model based on BERT, which is able to gather contextual representations to explore topic inference, generating consistent topics with state-of-the-art results.

In [77] it is presented a BERT-based model that aims to identify speech acts as means to comprehend the communicative intention of a speaker. The proposed model achieved an accuracy of 77.52%, outperforming other baselines approaches. Nevertheless, this work focused on developing a speech act for Twitter,

assessing the content and intent of tweets, and not extending their analysis for other domains.

In [78], the authors trained a BERT language understanding model for the Italian language (AlBERTo), focused on the language used in social networks, specifically on Twitter. The model is able to evaluate three sentiment analysis sub-tasks for the Italian language:

- **Subjectivity Classification:** To decide whether a given message is subjective or objective;
- **Polarity Classification:** To decide whether a given message is of positive, negative, neutral or mixed sentiment;
- **Irony Detection:** To decide whether a given message is ironic or not.

Despite achieving good results, AlBERTo was designed with a very specific choice: tweets in the Italian language. Therefore, this model does not perform well on other NLP tasks, nor on other languages.

The work of [32] introduces BioBERT, which is a domain-specific language representation model pre-trained on large scale biomedical corpora. With almost the same architecture across tasks, the authors show that BioBERT outperforms BERT in a variety of biomedical text mining tasks, including named entity recognition (0.62% F1 score improvement), biomedical relation extraction (2.80% F1 score improvement) and biomedical question answering (12.24% Mean Reciprocal Rank improvement.) Similarly to other domain-specific NLP models, BioBERT is restricted to some applications, evidently, in this case, to the biomedical field and related areas.

The work of [79] demonstrates the application of BERT to coreference resolution, achieving strong improvements on this task for the English language. According to their experiments, the authors state that BERT-large is particularly better than BERT-base at distinguishing between related but distinct entities, even though there is still room for improvement in modeling document-level context to deal with spread-out clusters. Another improvement for the coreference resolution task is proposed by [80], via an "Entity Equalization"mechanism. The key element of their approach is to capture properties of entity clusters and use those in the resolution process using BERT embeddings. Future work for their approach also includes the plan to further enrich these representations by considering information from across the document.

Another language representation model, ERNIE (Enhanced Representation through kNowledge IntEgration), was proposed by [81]. It incorporates Knowledge Graphs (KGs) to enhance language representation with external knowledge, establishing a good baseline for knowledge-driven tasks. However, it still underperforms LUKE and other state-of-the-art models. The work of [82] presented the first public large-scale pre-trained language model for English Tweets, named BERTweet. It achieves state-of-the-art performance on Tweet NLP tasks, including NER and text classification.

3 NEURAL INFORMATION EXTRACTION METHODOLOGY

In this work, it is proposed a pipeline for information extraction consisting of several components. The architecture of the proposed pipeline is shown in Figure 3.1. Each one of the components is described as follows.

3.1 INFORMATION EXTRACTION PIPELINE

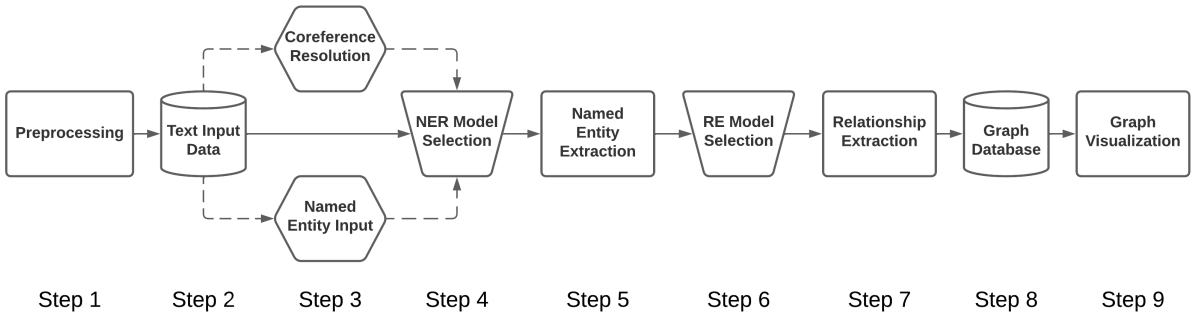

Figure 3.1: Information Extraction Pipeline Architecture.

3.1.1 Step 1: Preprocessing

The first step of the information extraction system is called *preprocessing* and it refers to the tasks made prior to processing the input data through the pipeline. In this work, this step corresponds to *data collection* and *model training and fine-tuning* (Figure 3.2).

The data collection task refers to the acquisition of the text data that will be processed by the information extraction pipeline in the next steps. The data may come from a variety of sources, including databases, websites, chats or documents, to name a few. A forensics approach can be used here to collect this data, and it is recommended to clean the data as much as possible, removing unnecessary information and reviewing the texts in order to generate good results by the end of the pipeline IE process.

The model training and fine-tuning task refers to the training and fine-tuning of the NLP models used in the IE pipeline. The main NLP models proposed in the next steps are task-specific, including models for Coreference Resolution, Named Entity Recognition and Relation Extraction. It is recommended to train and fine-tune NLP models for a specific domain or application to achieve the best possible results. The models' language is also a pertinent factor that should be considered during this task.

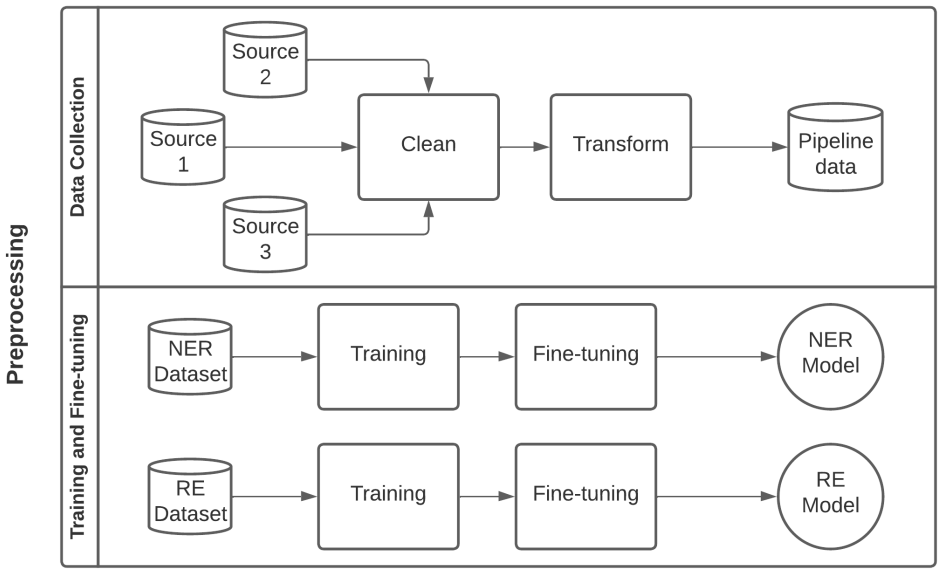

Figure 3.2: Pipeline's preprocessing step.

3.1.2 Step 2: Text Input Data

The second component of the IE system is characterized by text input data (Figure 3.1). This data can be anything, from news articles to social media chats, for example. There is not a limitation for the language type present in the input texts. However, some NER and RE models were trained on specific languages only, and therefore may not perform as well in other languages. Some multilingual models can be used to overcome this limitation, as we will discuss later. Ideally, the input data must be free of typographical and grammatical errors in order to output better results in the subsequent steps.

3.1.3 Step 3: Named Entity Input and Coreference Resolution

Step 2 of the processing pipeline as shown in Figure 3.1 consists of two optional components: Named Entity Input (NEI) and Coreference Resolution (CR). NEI can help improve the model output by forcing it to take into account some entities considered relevant for the task at hand. Since the NER models have limited and frequently different labels for the entities they recognize, it may be important to include some entities manually or by using regular expressions. Moreover, CR may help improve the accuracy of the RE model in Step 5, since the identification of antecedent chains can output cleaner relation pairs and less ambiguous text [83]. The decision about whether to use NEI and CR or not is left to the user. However, some general criteria such as the number of different entity classes the NER models are able to identify, as well as the sources of the data being used, and the time required to process all this data, can be used to decide whether the steps of NEI and CR are worth it or not.

3.1.4 Step 4: Named Entity Recognition Model Selection

In Step 4 (Figure 3.1), it is selected one of the trained NER models to be used for Entity Extraction in Step 5. The models differ from each other in the language they were trained on, in the types of entities they are able to recognize and in their performance. All NER models were fine-tuned on BERT and its

derivatives, like ELECTRA [23] and RoBERTa [22], for example. The model selection is strictly related to the data's input language from Step 1. If the input language is known beforehand, it is recommended to select a NER model fine-tuned for that particular language, given the fact that non-multilingual models may not perform well on different languages. However, if the input language is unknown, or if it is composed of multilingual documents, a multilingual model may perform well for most languages.

3.1.5 Step 5: Named Entity Extraction

Once the underlying NER model is selected, entities in the input data are recognized and extracted in Step 5 (Figure 3.1), generating a list of all entities that will later be used to feed the RE models. Usually some cleaning process is applied in this step with the intent of removing duplicated entity mentions from the final entities list. This is desirable in order to generate a more concise and efficient graph visualization. However, it also may be desirable to keep a record of how many times the same entity was mentioned in a document or across documents, for example.

3.1.6 Step 6: Relation Extraction Model Selection

Step 6 (Figure 3.1) consists of choosing an appropriate Relation Extraction model to be applied over the named entities extracted from the previous step. Three RE models were used, including two for English texts and one for Portuguese texts. Similarly to NER models, they differ from each other in the types of relations they can recognize. Therefore, it is crucial to select a RE model for the target language, since for this step, no multilingual RE models were conceived. Besides, different RE models have different sets of predefined relations. This means that applying different RE models over the same input text may output different relations between entities in the text, some of which may be detected by one of them and not by the other, or they both may detect the same relation but with different names or confidence scores. It also means that this is an interactive process and most of the time it may be desirable to apply more than one RE model over the same text input in order to combine outputs and extract more relevant information.

3.1.7 Step 7: Relationship Extraction

After choosing an appropriate RE model, the relations between entity pairs are extracted in Step 7 (Figure 3.1), generating a list of entities and their corresponding relations, if any. The RE output for a pair of entities is usually represented in the form of a triplet: `(e1, rel, e2)`, where `e1` is the source entity, `e2` is the target entity and `rel` is the predicted relation between them. This output format facilitates the construction of a directed graph, however, it is necessary to note that some relations only exist in one direction. For example, in the sentence "John is the father of Daniel", there are two possible outcomes: `(John, father, Daniel)` and `(Daniel, parent, John)`. A RE model may have these two predefined relations, whereas others may have only one of them. Therefore, it is important to consider both directions (`source entity <-> target entity`) and decide whether the output is relevant for both of them, only one of them or none of them. Some RE models are able to detect when there is no relation between a pair of entities at all, while others will output some predefined relation with a low confidence score. This characteristic is related to how the model was trained and with which data and predefined relations set.

3.1.8 Step 8: Graph Database

Step 8 (Figure 3.1) corresponds to the storage process for the entities and relations extracted from the previous steps. This data is usually stored in a graph database, but it can be as easily stored in traditional relational and non-relational databases. The data format is flexible and it may be defined by the interested user. A JSON (JavaScript Object Notation) is usually a good option format since it can be exported for several different applications and it is extensively used in web applications.

3.1.9 Step 9: Graph Visualization

Step 9 (Figure 3.1) is the output of the proposed system, in the form of an interactive graph, in which nodes represent named entities and edges represent the relations between them. A graph visualization helps analysts and investigators with an overview of all relevant insights obtained from the input data, making it easy to create filters, plots and detailed reports. For consistency purposes, it is important to have a set of predefined relations between named entity types. In this way, it is easier to search similar relationships across different documents mentioning related entities and plot them together in the same graph, for instance.

3.2 PREPROCESSING

In order to achieve the output shown in Figure 3.1 as a graph visualization, several NLP steps had to be implemented. Figure 3.3 shows an example of how the coreference model works with a text input.

3.2.1 Coreference Resolution

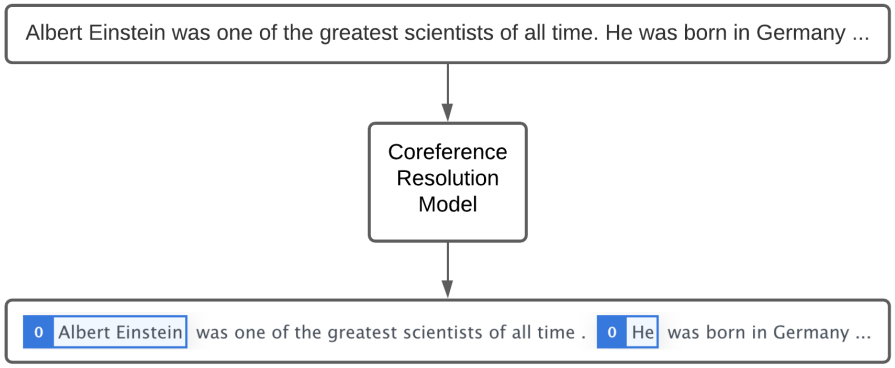

Figure 3.3: Coreference Resolution example.

The model essentially searches the text based on parsing and morphological dependencies to replace proper nouns and possessive pronouns with their respective terms. In the example of the Figure 3.3, “He”, in the second phrase, was replaced by its original reference, “Einstein”. For this part, it was used Neural-coref [84], which uses the corpora of the CoNLL-2012 shared task [85] and a neural net scoring model

described in [86]. Since the original repository only includes production-ready support for the English language, applications of CR were not considered in this work for other languages.

However, it is possible to train a CR model based on [84] for other languages, provided an annotated dataset in the target language and some modifications on the loading and parsing scripts. For Portuguese, corpora [87] and [88] are good options to train a new coreference model. Further experiments to expand the CR model to support Portuguese were not considered in this work due to the fact that it would require a significant understanding of the language parse trees to reach an acceptable identification of mentions, which is out of the scope of this research.

3.2.2 Named Entity Input

NEI can help improve the model output by forcing it to take into account some entities considered relevant for the task at hand. Since the NER models have limited and frequently different labels for the entities they recognize, it may be important to include some entities manually. SpaCy, for example, enables the use of a custom Entity Ruler based on a token-level or document-level matcher to identify and label entities in the input text. Traditional regular expressions matching is also supported, making it easy and often useful to highlight words of interest within the text.

3.2.3 Named Entity Recognition Corpora

Regarding the NER model training, a couple of pre-trained BERT derivative transformers were used, in order to fine-tune them with custom datasets for this matter. Table 3.1 shows some information about the datasets used for NER fine-tuning [89] [90].

Table 3.1: Datasets used for NER.

Dataset	Language	Domain	Sentences	Tokens	Reference
First HAREM	Portuguese	General	5000	80000	[91]
Mini HAREM	Portuguese	General	1000	1000	[91]
Second HAREM	Portuguese	General	3500	100000	[92]
Paramopama	Portuguese	Wikipedia	12500	310000	[5]
LeNER-Br	Portuguese	Legal	10392	318073	[63]
WikiNER	Portuguese	Wikipedia	125821	2830000	[93]
CoNLL03	English	News	22137	301418	[66]
WNUT17	English	Social Media	5690	101857	[94]

As shown in Table 3.1, the datasets used for NER were in two languages: Portuguese [89] and English [90]. These languages were prioritized because there are not many production level solutions for Portuguese NER yet, and because English is widely used worldwide, also due to its relevance in texts over the Internet and social media.

For First HAREM (*Avaliação de Reconhecimento de Entidades Mencionadas*), Mini Harem, Second HAREM and Paramopama corpora, the training, development and test splits were 60%, 20% and 20%, respectively. For LeNER-Br, CoNLL03 and WNUT17 (Workshop on Noisy User-generated Text) it was adopted the splits from their original sources.

The tagging scheme used for NER corpora was the IOB (Inside-Outside-Beginning) scheme [95], in which all data files contain one word per line with empty lines representing sentence boundaries. At the end of the line there is a tag that expresses whether the current word is the beginning of an entity (B), inside an entity (I) or not an entity at all (O). Here it is an example sentence:

Albert	B-PER
Einstein	I-PER
was	O
in	O
Germany	B-LOC
.	O

3.2.4 Relation Extraction Corpora

For the Relation Extraction stage, it was used three different models, as shown in Table 3.2.

Table 3.2: Relation Extraction Models.

Name	Language	Sentences	Entities	Relations	Reference
DBPedia	Portuguese	91914	3	9	[96]
Wiki	English	56000	4	25	[70]
TACRED	English	106264	5	42	[75]

The framework behind RE operates over four different possible approaches: Sentence-Level RE, Bag-Level RE, Document-Level RE and Few-Shot RE. In [42], the authors explain the fundamental differences between each approach. In this work, it was prioritized Sentence-Level RE since the training dataset was based on sentences, besides this being the primary approach adopted by [42]. However, it is simple to select another option based on the characteristics of the data being analysed.

For the DBPedia corpus [97], the training, development and test split was 60%, 20% and 20%, respectively. For the Wiki corpus [70], it was used the splits provided by [98], and the splits for TACRED are described in [75]. Table 3.3 shows the data splits for each dataset.

The data format required to train a RE model with [98] is a text file in which each line is a JSON (JavaScript Object Notation) object containing a list of tokens, two entities with their respective position indexes in the sentence and the relation label between them.

Wiki and TACRED models were trained on English data, whereas DBPedia model was developed over a Portuguese corpus. Figures 3.5, 3.7 and 3.9 show the possible relationships between entity types for DBPedia, Wiki and TACRED, respectively.

3.3 RELATIONS SCHEMAS

This section presents the relations schemas considered in this work in order to generate the entity graphs as the output of the information extraction pipeline proposed. Three schemas were assembled: one

for Portuguese texts (DBPedia) and two for English texts (Wiki and TACRED). Each relation schema is composed of nodes representing named entities classes and edges representing predefined relationship types. For convention, it was attributed a different color for each entity class shown throughout this work: green for Person (PER), blue for Organization (ORG), orange for Location (LOC), yellow for Miscellaneous (MISC) and red for Date (DATE).

The relations schemas presented here do not take into consideration new customized entity classes, like the ones introduced into the IE pipeline in Step 2 (NEI), for example. Thus, if an entity class is created through regular expressions or other methods, it will be present among the other entities, nevertheless, since that new entity class is not present in any predefined relation schema, there will be no relationships between it and any other extracted entity. It is possible, however, to extend the relations schemas in order to include new entity classes and relationship types. To do so, it would be required to include observations of these new relationships in the training data and retrain the model to support it.

Table 3.3: Data splits for each RE dataset.

Dataset	Split	Number of examples
DBPedia	Train	14,572
	Dev	4,859
	Test	4,860
Wiki	Train	15,750
	Dev	1,750
	Test	1,750
TACRED	Train	68,124
	Dev	22,631
	Test	15,509

3.3.1 DBPedia Relations Schema

Table 3.4: Relationship instances gathered for the DBPedia schema.

Subject	Object	Relationship	Number of examples
Person	Person	parent	271
Person	Person	successor	519
Person	Person	partner	140
Organization, Location	Person	keyPerson	383
Person, Organization	Organization	influencedBy	153
Person, Organization	Organization	partOf	5313
Person	Location	origin	23880
Person	Location	deathOrBurialPlace	6832
Location, Organization	Location	locatedInArea	44291

For the DBPedia schema, as shown in Figure 3.5, there are 9 different types of predefined relationships (including a *other* class) between 3 node classes: Person, Organization and Location. Table 3.4 shows the

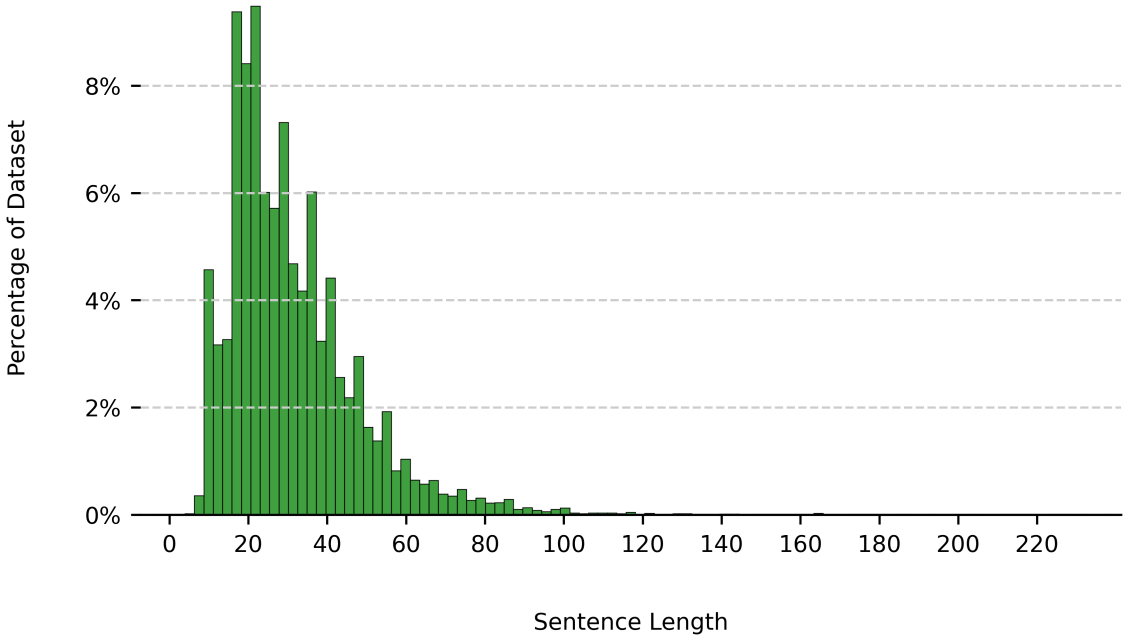

Figure 3.4: DBPedia sentence length (tokens count) distribution: The lowest sentence length is 4, the highest sentence length is 230 and the average sentence length is 31.29.

subject and object types plus the number of examples for each relationship class, whereas Figure 3.4 shows the sentence length distribution of the DBPedia corpus.

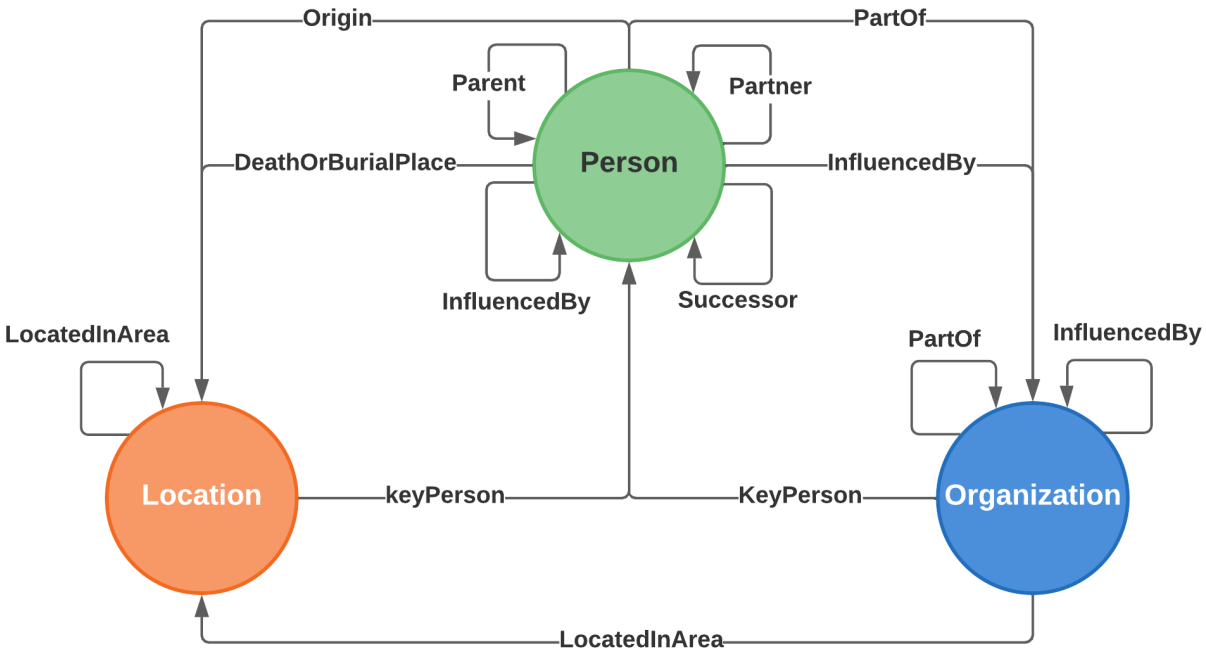

Figure 3.5: DBPedia Relations Schema (adapted from [96]).

Table 3.5: Relationship instances gathered for the Wiki schema.

Subject	Object	Relationship	Number of examples
Person	Miscellaneous	religion	700
Person	Location	head of government	700
Person	Location	country of citizenship	700
Person	Miscellaneous	participant of	700
Person	Miscellaneous	position held	700
Organization	Location	location of formation	700
Person, Organization	Location	country of origin	700
Person	Person	father	700
Miscellaneous	Organization	developer	700
Miscellaneous	Organization	manufacturer	700
Person	Organization	member of political party	700
Organization	Location	headquarters location	700
Person	Person	sibling	700
Person	Miscellaneous	instrument	700
Person	Miscellaneous	occupation	700
Person	Location	residence	700
Person	Location	work location	700
Organization	Organization	subsidiary	700
Person	Miscellaneous	participant	700
Organization	Organization	owned by	700
Person	Miscellaneous	field of work	700
Person	Person	spouse	700
Person	Person	mother	700
Person, Location	Organization, Location	member of	700
Person	Person	child	700

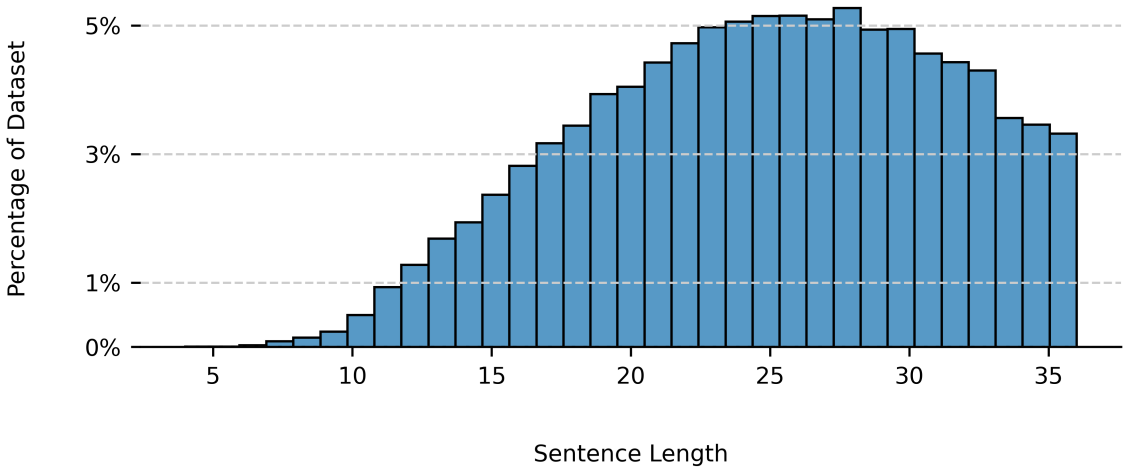

Figure 3.6: Wiki sentence length (tokens count) distribution: The lowest sentence length is 5, the highest sentence length is 36 and the average sentence length is 25.03.

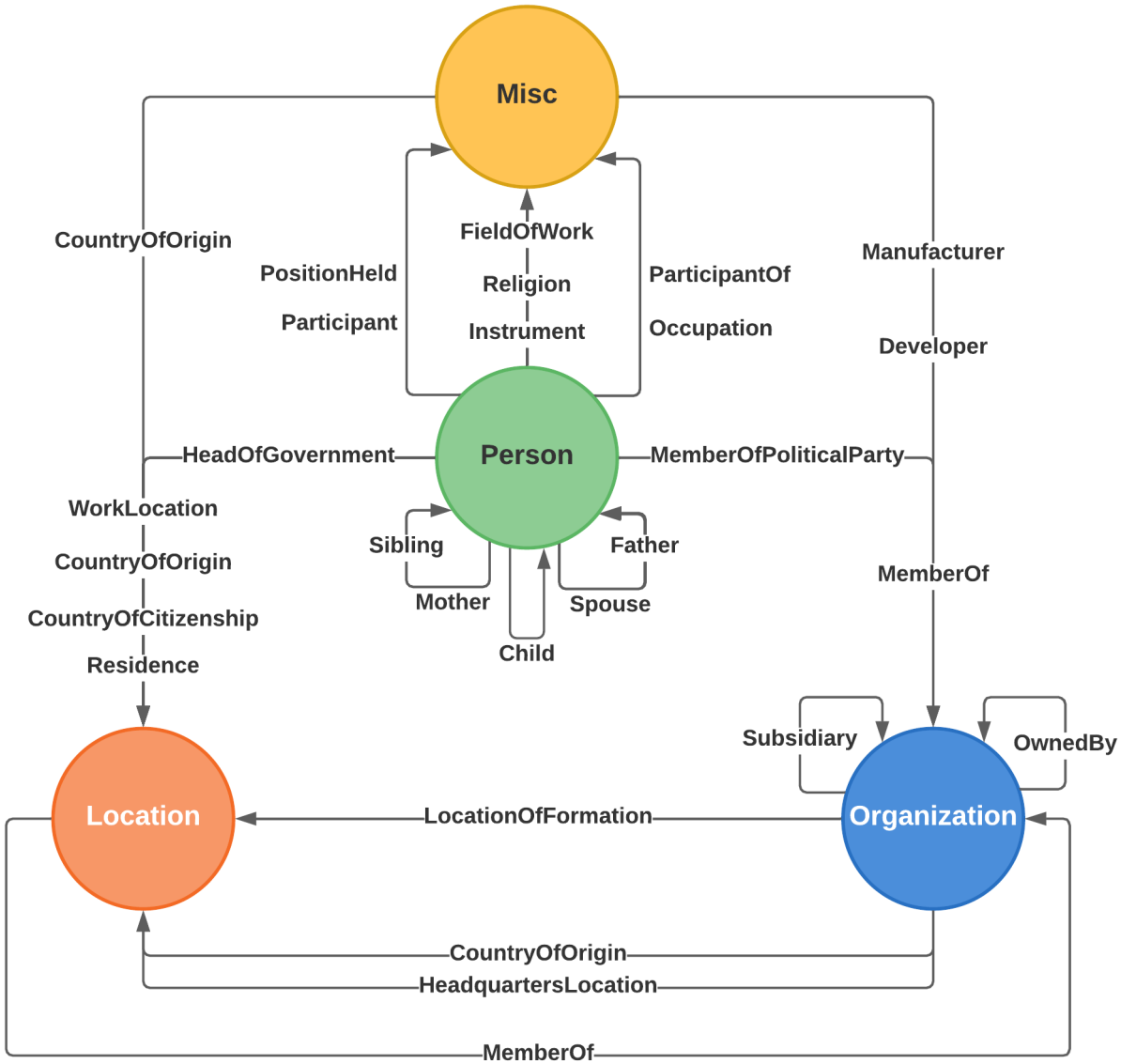

Figure 3.7: Wiki Relations Schema (adapted from [70]).

3.3.2 Wiki Relations Schema

The Wiki schema (Figure 3.7) presents 25 relationship classes and 4 node classes (Person, Organization, Location and Miscellaneous). Table 3.5 shows the subject and object types plus the number of examples for each relationship class, whereas Figure 3.6 shows the sentence length distribution of the Wiki corpus.

3.3.3 TACRED Relations Schema

The TACRED schema consists of 42 relation classes (including a *no_relation*) class and 5 entity classes (Person, Organization, Location, Date and Miscellaneous), as shown in Figure 3.9. The original dataset [75] contains other entity classes (`CITY`, `COUNTRY` and `NUMBER`, for example) that were converted to one of the proposed entity classes for practical applications. Table 3.6 shows the subject and object types

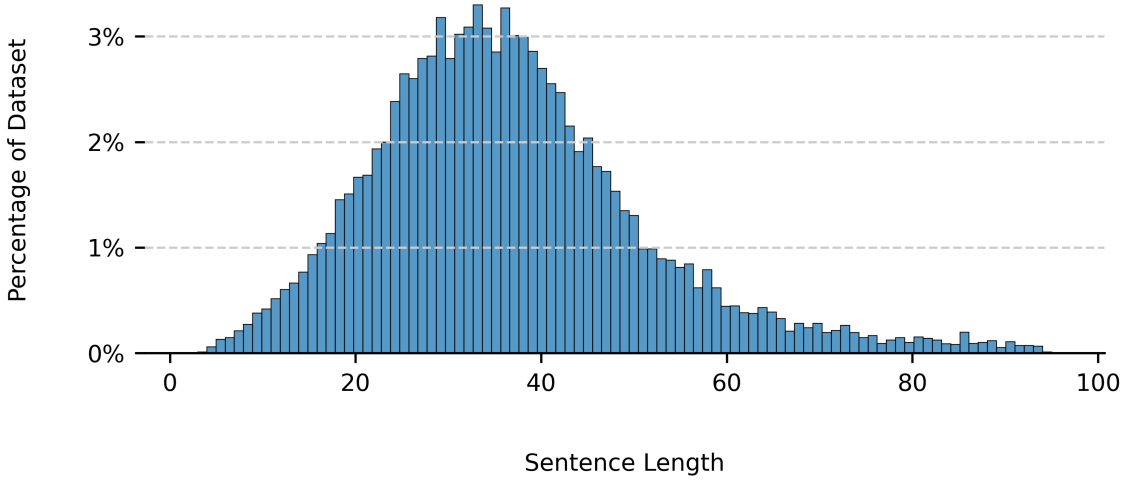

Figure 3.8: TACRED sentence length (tokens count) distribution: The lowest sentence length is 2, the highest sentence length is 96 and the average sentence length is 36.38.

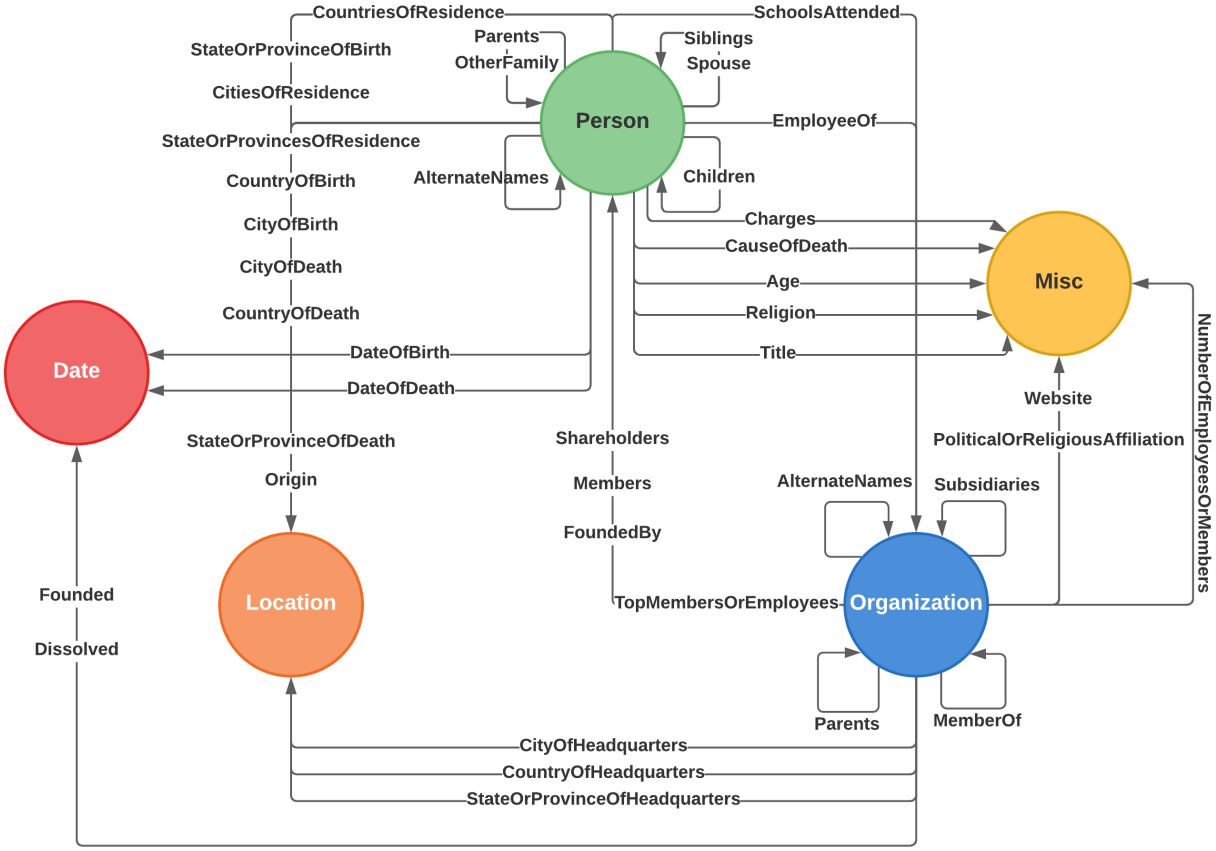

Figure 3.9: TACRED Relations Schema (adapted from [75]).

plus the number of examples for each relationship class, whereas Figure 3.8 shows the sentence length distribution of the TACRED corpus.

Table 3.6: Relationship instances gathered for the TACRED schema.

Subject	Object	Relationship	Number of examples
Organization	Date	org:founded	166
Organization	Organization	org:subsidiaries	453
Person	Date	per:date_of_birth	103
Person	Miscellaneous	per:cause_of_death	337
Person	Miscellaneous	per:age	833
Person	Miscellaneous	per:charges	280
Person	Location	per:stateorprovince_of_birth	72
Person	Location	per:countries_of_residence	819
Person	Location	per:country_of_birth	53
Person	Location	per:stateorprovinces_of_residence	484
Organization	Miscellaneous	org:website	223
Person	Location	per:cities_of_residence	742
Person	Person	per:parents	296
Person	Organization	per:employee_of	2163
Person	Location	per:city_of_birth	103
Organization	Miscellaneous	org:political/religious_affiliation	125
Organization	Miscellaneous	org:number_of_employees/members	121
Person	Organization	per:schools_attended	229
Person	Location	per:country_of_death	61
Person	Person	per:children	347
Organization	Person	org:top_members/employees	2770
Person	Date	per:date_of_death	394
Organization	Person	org:members	286
Organization	Organization	org:parents	444
Organization	Organization	org:alternate_names	1359
Person	Miscellaneous	per:religion	153
Organization	Organization	org:member_of	171
Organization	Location	org:city_of_headquarters	573
Person	Location	per:origin	667
Organization	Person	org:shareholders	144
Person	Miscellaneous	per:title	3862
Organization	Location	org:country_of_headquarters	753
Person	Person	per:alternate_names	153
Person	Person	per:siblings	250
Organization	Location	org:stateorprovince_of_headquarters	350
Organization	Date	org:dissolved	33
Person	Person	per:spouse	483
Person	Person	per:other_family	319
Person	Location	per:city_of_death	227
Person	Location	per:stateorprovince_of_death	104
Organization	Person	org:founded_by	268

4 EXPERIMENTS AND RESULTS

This Chapter presents the experiments and main results of this thesis, including NER models hyper-parameters tuning that proved to enhance performance through higher F1 scores, besides six examples of the full information extraction pipeline application for Portuguese and English texts, culminating in the presentation of interactive graphs. Table 4.1 shows the tools and versions used for each NLP task, whereas Table 4.2 shows the average time invested in each discussed step, including training and validation times for Named Entity Recognition, Hyper-parameters Tuning and supervised Relation Extraction for a single language (based on Colab's cloud environment [99]).

Table 4.1: Tools used for each NLP task.

Task	Tool	Version
Data acquisition	FTK Imager [100]	4.5.0.3
Digital evidence processing	IPED [16]	3.18.9
Coreference resolution	AllenNLP [101]	2.1.0
Named entity recognition	SpaCy [4]	3.2.0
Relation extraction	OpenNRE [98]	0.1
Graph database and graph visualization	Neo4j Desktop [102]	1.4.8
Models training and fine-tuning	Google Colab [99]	Free [1]
Pipeline application	Avell computer [2]	Ubuntu 20.04.2 LTS

[1] Intel Xeon CPU @2.30GHz, 13GB DDR4 RAM and a 12GB GDDR5 NVIDIA Tesla K80 GPU.
[2] Intel Core i7-7700HQ CPU @2.80GHz, 32GB DDR4 RAM and a 4GB GDDR5 NVIDIA GeForce GTX 1050 Ti GPU.

Table 4.2: Average time invested in each experiment.

Preprocessing	NER	Hyper-parameters Tuning	RE	Total
2 hours	2 hours	4 hours	2 hours	10 hours

The following equations describe the evaluation metrics used throughout this work for NER and RE, including Micro Precision (P), Micro Recall (R) and Micro F-Score (F_1):

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.3}$$

Where TP (True Positives) mean correctly extracted mentions and FP (False Positives) and FN (False Negatives) mean incorrectly extracted mentions.

4.1 NAMED ENTITY RECOGNITION SETUP

Training is an iterative process in which the model's predictions are compared against the reference annotations in order to estimate the gradient of the loss. The gradient of the loss is then used to calculate the gradient of the weights through backpropagation. The gradients indicate how the weight values should be changed so that the model's predictions become more similar to the reference labels over time (Figure 4.1).

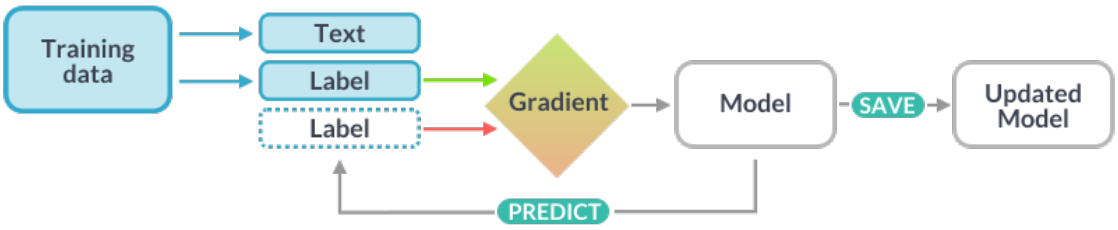

Figure 4.1: SpaCy's training flow (source: SpaCy documentation [103]).

When training a model, the goal is not just to make it memorize the examples but, rather, make it come up with a theory that can be generalized across unseen data. After all, it is not desirable to instruct the model to learn that a particular instance of "Amazon" is a company, but instead it should learn that "Amazon", in contexts like this, is most likely a company. That is why the training data should always be representative of the data that will be processed in real-world applications. A model trained on Wikipedia, where sentences in the first person are extremely rare, will likely perform badly on Twitter. Similarly, a model trained on romantic novels will likely perform badly on legal texts.

This also means that in order to know how the model is performing, and whether it is learning the right things through the right parameters, not only training data is required, but also evaluation data. If the performance of the model is only tested with the data it was trained on, there will be no indications of how well it is generalizing. Thus, in order to train a model from scratch, usually at least a few hundred examples for both training and evaluation are required.

The NER task itself can be represented by its own processing pipeline (Figure 4.2). In SpaCy, The recommended way to train pipelines is via the `spacy train` command (CMD) on the Command Line Interface (CLI). It only needs a single `config.cfg` configuration file that includes all settings and hyper-parameters. Optionally, it is possible to overwrite settings on the command line, and load in a Python file to register custom functions and architectures. Appendix I contains more details about setting up a NER

pipeline with SpaCy and its configuration parameters.

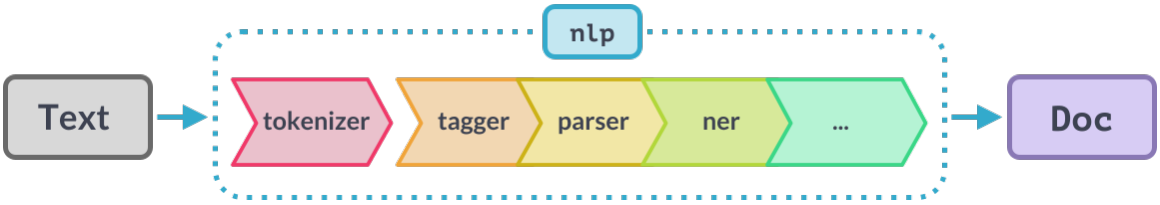

Figure 4.2: SpaCy's NLP pipeline (source: SpaCy training [103]).

During processing, SpaCy first tokenizes the text (segments it into words, punctuation and so on). This is done by applying rules specific to each language. For example, punctuation at the end of a sentence should be split off, whereas "N.Y." should remain one token (Figure 4.3). Each `Doc` consists of individual tokens, and it is possible to iterate over them:

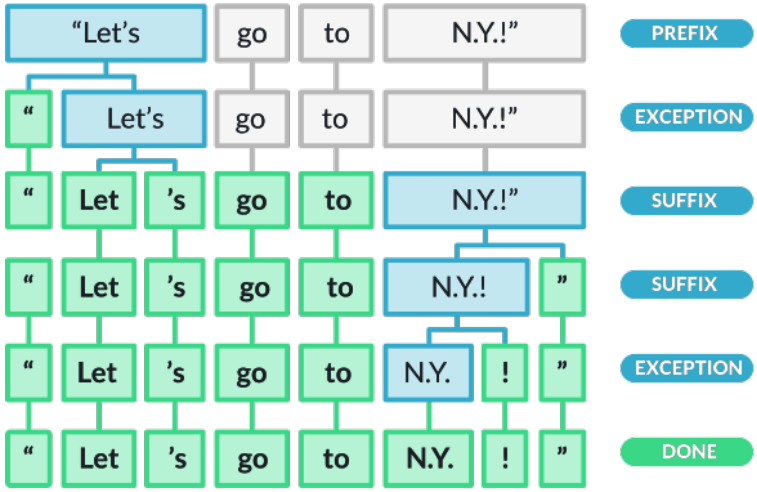

Figure 4.3: Tokenization example (source: SpaCy documentation [103]).

First, the raw text is split on whitespace characters, similar to Python's `text.split(' ')` method. Then, the tokenizer processes the text from left to right. On each substring, it performs two checks:

1. Does the substring match a tokenizer exception rule? For example, "don't" does not contain whitespace, but should be split into two tokens, "do" and "n't", while "U.K." should always remain one token;

2. Can a prefix, suffix or infix be split off? For example punctuation like commas, periods, hyphens or quotes.

If there's a match, the rule is applied and the tokenizer continues its loop, starting with the newly split substrings. This way, SpaCy can split complex, nested tokens like combinations of abbreviations and multiple punctuation marks.

While punctuation rules are usually pretty general, tokenizer exceptions strongly depend on the specifics of the individual language. This is why each available language has its own subclass, like English or Portuguese, that loads in lists of hard-coded data and exception rules.

In order to establish a performance comparison between the datasets used for NER fine-tuning and different transformer models, an experiment was made with some common hyper-parameters. Table 4.3 shows the hyper-parameters used for training the models, Table 4.4 shows the overall results for Portuguese models and Table 4.5 shows the overall results for English models, whereas Figure 4.4 compares Precision, Recall and F-Score values for Portuguese NER systems and Figure 4.5 compares Precision, Recall and F-Score values for English NER systems, respectively.

Table 4.3: Hyper-parameters Values used to Train the NER Models.

Hyper-parameter	**Value**
Number of epochs	20
Early stop patience	1600
Scheduler	Linear with warm-up
Dropout rate	0.1
Batch size	128
Optimizer	AdamW with bias correction
AdamW ϵ	1×10^{-8}
Learning rate	2×10^{-5}
Warmup steps	250
Total steps	20000
Training evaluation frequency	200
Clipping gradient norm	1.0
Sequence size	128

Table 4.4: Overall Results for NER Systems for Portuguese.

System	**Corpus**	**Transformer**	**Precision**	**Recall**	**F-Score**
FHBPT	First HAREM	BERT-PT[1]	77.9	82.0	79.9
MHBPT	Mini HAREM	BERT-PT[1]	80.8	78.1	79.4
MHBM	Mini HAREM	BERT-M[1]	75.6	74.9	75.2
MHDBM	Mini HAREM	DistilBERT-M[1]	73.3	69.6	71.4
SHBPT	Second HAREM	BERT-PT[1]	84.5	86.9	85.7
PBPT	Paramopama	BERT-PT[1]	88.9	89.2	89.0
LBPT	LeNER-Br	BERT-PT[1]	90.3	88.2	89.2
WBPT	WikiNER	BERT-PT[1]	90.5	90.9	90.7

[1] BERT-PT = BERT Portuguese [21]; BERT-M = BERT Multilingual [3]; DistilBERT-M = DistilBERT Multilingual [19].

As we can see from the results shown in Tables 4.4 and 4.5, the best models for Portuguese were the ones trained on WikiNER and Paramopama datasets, with F-Scores of 90% and 89%, respectively. For English, the best ones were based on RoBERTa and DistilRoBERTa transformers over the CoNLL03 corpus, with F-Scores of 92% and 91%, respectively.

The results may vary depending mainly on the corpus used for training and the base transformer adopted. For example, the WNUT17 [94] corpus was chosen since it is composed of social media texts, containing many words and expressions that are not present on the other datasets. This allows to obtain a

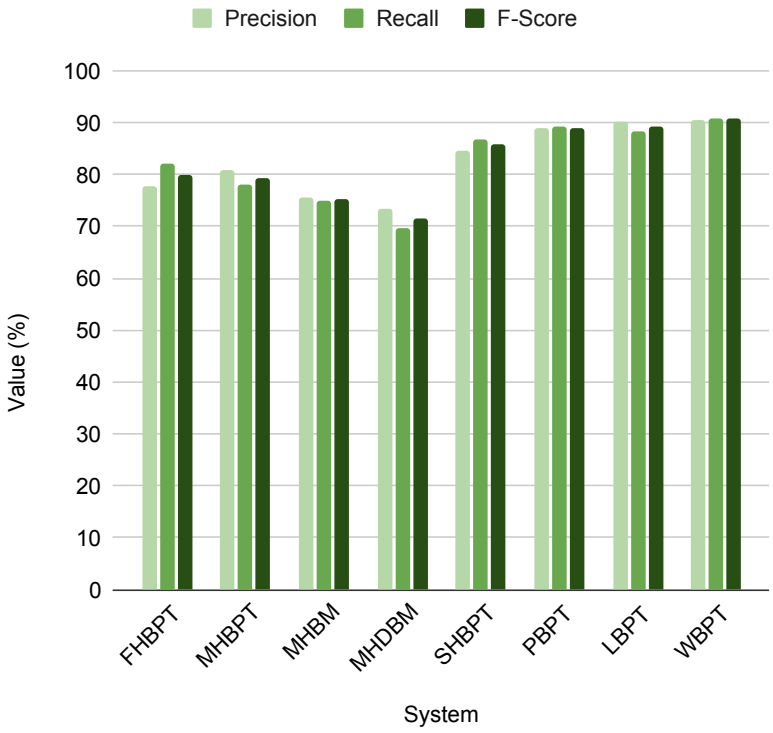

Figure 4.4: Precision, Recall and F-Score output for Portuguese NER systems (from Table 4.4).

Table 4.5: Overall Results for NER Systems for English.

System	Corpus	Transformer	Precision	Recall	F-Score
CB	CoNLL03	BERT [3]	90.9	90.7	90.8
CDB	CoNLL03	DistilBERT [19]	89.0	89.5	89.3
CR	CoNLL03	RoBERTa [22]	91.8	92.4	92.1
CDR	CoNLL03	DistilRoBERTa [19]	91.2	91.6	91.4
CE	CoNLL03	ELECTRA [23]	90.5	91.4	91.0
WB	WNUT17	BERT [3]	58.7	32.9	42.0
WR	WNUT17	RoBERTa [22]	56.8	42.0	48.3
WE	WNUT17	ELECTRA [23]	57.6	38.0	45.8

model that performs better on NER for social media texts, even though it may not perform as well in other domains. The same occurs with the LeNER-Br corpus, which is based on Brazilian legal documents, containing entities that only make sense in such texts.

In practice, apart from the final scores, choosing the best model requires knowledge about the nature of the input text. Some corpus allow flexibility for several domains, while others achieve state-of-the-art results for a specific domain.

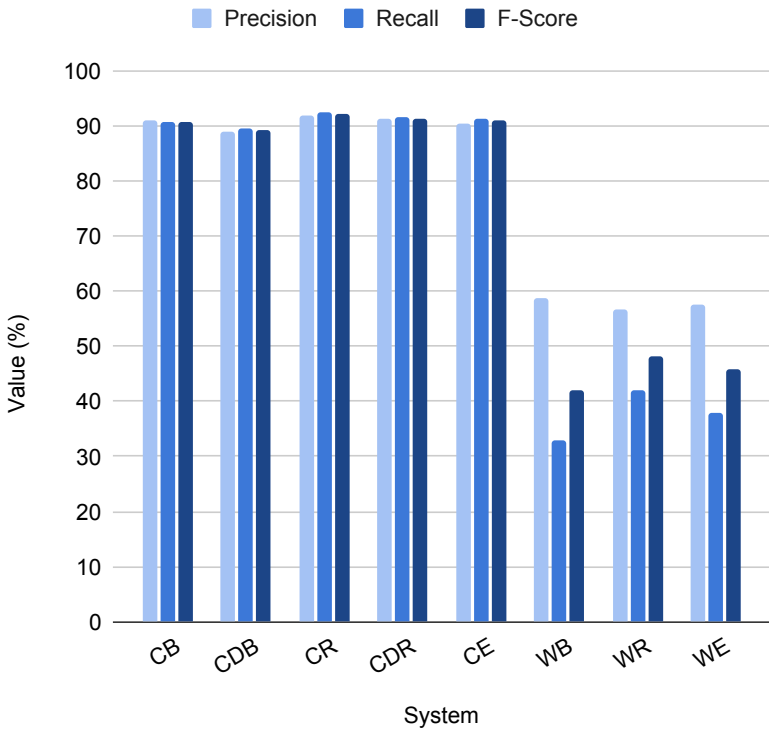

Figure 4.5: Precision, Recall and F-Score output for English NER systems (from Table 4.5).

4.1.1 Hyper-parameters Tuning

In order to achieve better results and an optimal model for NER in Portuguese, some tests were made with different hyper-parameters. English models were not considered for this task since they already show state-of-the-art results for many transformer architectures and different annotated corpora. In this sense, our focus was to choose a Portuguese corpus that allows us to run multiple tests and yet achieve good performance with a reasonable amount of time required for training.

The Paramopama corpus was chosen as the input data for training, since it was one of the best performing corpus for Portuguese in the tests, and it is significantly smaller than the original WikiNER corpus. Listing 4.1 shows fragments of a script used to automate the process of finding better hyper-parameters through a Bayesian Search [104]. Table 4.6 shows the tuned hyper-parameters for this corpus and Table 4.7 shows the NER model training output using these hyper-parameters for Portuguese, whereas Table 4.8 shows the evaluation output for the English NER model based on CoNLL03 with common hyper-parameters.

Listing 4.1: Hyper-parameters optimization with Bayes Search.

```
method: bayes
metric:
  goal: maximize
  name: ents_f
command:
  - ${env}
```

```
  - ${interpreter}
  - scripts/sweeps_using_config.py
  - ./configs/default_config.cfg
  - ./training
parameters:
  training.optimizer.learn_rate.warmup_steps:
    values:
    - 100
    - 150
    - 200
    - 250
  training.dropout:
    distribution: uniform
    max: 0.6
    min: 0.1
```

Table 4.6: Tuned Hyper-parameters for NER in Portuguese.

Hyper-parameter	**Value**
Transformer	BERT-Base-Portuguese
Number of epochs	30
Training max steps	30000
Dropout rate	0.5
Batch size	128
Learning rate	2×10^{-5}
Warmup steps	150
Total steps	20000
Training Eval frequency	200

Table 4.7: Hyper-parameters Tuning Results for NER in Paramopama.

Entity	**Precision**	**Recall**	**F-Score**
Location	94.8	94.2	94.5
Organization	68.5	80.5	74.0
Time	86.8	90.6	88.7
Person	88.0	96.1	91.9
Overall	**90.6**	**92.5**	**91.5**

It was possible to achieve an improvement of about 2% for the F-Score metric by just analyzing the best hyper-parameters for the model. This setup is not guaranteed to be the best, since not all possible combinations of parameters for training were tested. However, it is a considerable improvement for a state-of-the-art NER model for the Portuguese language. Figure 4.6 shows the confusion matrix for the fine-tuned Portuguese NER model, in which the X axis represents the predicted label for each entity class, and the Y axis represents the true label for each entity class. By analysing this matrix, it is possible to observe that the model obtained good performances in the identification of `Person` and `Location`

entity classes, although it had some difficulty in distinguishing between `Organization` and `Location` and missed a considerable amount of `Time` entities.

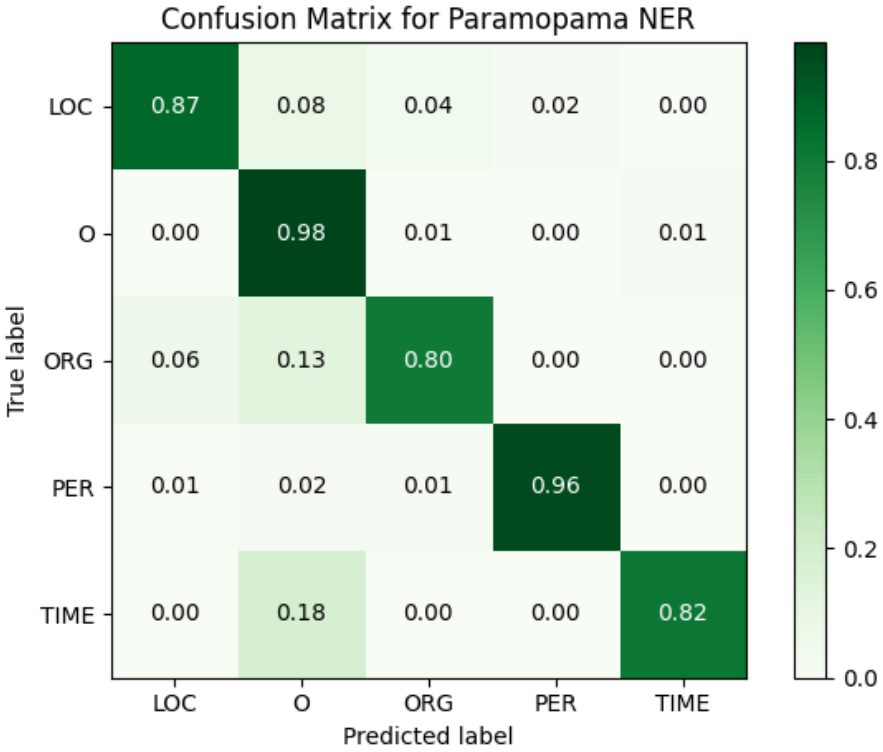

Figure 4.6: Confusion Matrix for Paramopama NER.

Table 4.8: Results for NER in CoNLL03 with Common Hyper-parameters.

Entity	Precision	Recall	F-Score
Organization	90.2	92.3	91.3
Location	93.7	93.6	93.7
Miscellaneous	80.6	82.3	81.4
Person	96.7	95.5	96.1
Overall	**91.8**	**92.4**	**92.1**

In order to create a confusion matrix for English NER, since the CoNLL's test set was already tokenized, it was necessary to reverse engineer the tokenization process to form sentences and input them into SpaCy's NLP pipeline for Named Entity Recognition. During the experiments, it was clear that the tokenization process used prior to the train-dev-test set splits had some differences in comparison with SpaCy's built-in tokenizer. As an example, words with an hyphen between them, such as `weather-related`, were split into two different tokens (`weather` and `related`, in this example). The solution was to modify the pipeline's tokenizer to address these particular cases and make the tokenization process equal, so that it would be possible to compare the NER model's output token by token and generate the necessary statistics for the confusion matrix.

Listing 4.2 shows the modifications made to the tokenizer to address the tokenizations differences.

Listing 4.2: Custom tokenizer definition for CoNLL NER.

```
1  import spacy
2  from sklearn.metrics import confusion_matrix
3  from matplotlib import pyplot
4  import numpy
5  import re
6  import spacy
7  from spacy.tokenizer import Tokenizer
8  from spacy.util import compile_prefix_regex, compile_infix_regex,
9  compile_suffix_regex
10
11 spacy.require_gpu()
12
13 # For English CoNLL
14 def custom_tokenizer(nlp):
15
16     infix_re = re.compile(r'''[~]''')
17     prefix_re = compile_prefix_regex(nlp.Defaults.prefixes)
18     suffix_re = re.compile(r'''[\]\)]$''')
19
20     special_cases = {"'s": [{"ORTH": "'s"}], "'S": [{"ORTH": "'S"}],
21                      "'m": [{"ORTH": "'m"}], "'M": [{"ORTH": "'M"}],
22                      "'re": [{"ORTH": "'re"}], "'RE": [{"ORTH": "'RE"}],
23                      "'ll": [{"ORTH": "'ll"}], "'LL": [{"ORTH": "'LL"}],
24                      "'ve": [{"ORTH": "'ve"}], "'VE": [{"ORTH": "'VE"}],
25                      "'d": [{"ORTH": "'d"}], "'D": [{"ORTH": "'D"}]}
26
27     return Tokenizer(nlp.vocab, rules=special_cases,
28                                 prefix_search=prefix_re.search,
29                                 suffix_search=suffix_re.search,
30                                 infix_finditer=infix_re.finditer,
31                                 token_match=None)
32
33
34 nlp = spacy.load('conll-roberta-base/model-best')
35 nlp.tokenizer = custom_tokenizer(nlp)
```

Firstly, the `infix` definition, which is responsible for intra-tokens split rules, was redefined to avoid tokenization of words with hyphen and punctuation. Later, the `suffix` definition was also edited, specially to address cases in which numbers followed by a dot were split into two separate tokens. Finally, some special cases for the English language had to be manually provided to guarantee contractions would remain as a single token. Once the new tokenizer was defined, the trained NER model was loaded (line 34) and the tokenizer applied (line 35).

Listing 4.3, in turn, shows the gold-standard (line 15) and prediction (line 16) vectors used to plot the confusion matrix. The classes (NER labels) were defined, as well as a function used to remove the IOB tags from the input annotated file and retain only the label. This is done because even though the IOB tags are necessary for training the model, it is not mandatory for evaluation nor for creating a confusion matrix. Therefore, a vector input of `["B-PER", "I-PER", "O", "B-ORG"]`, for example, would

be converted into `["PER", "PER", "O", "ORG"]`. After populating both vectors, they can then be forwarded to a confusion matrix plot function.

Listing 4.3: Prediction for CoNLL NER.

```
inputFile = open("input/test_conll_clean.tsv", "r", encoding='UTF-8')
exportFile = open("output/output.tsv", "a", encoding='UTF-8')

classes = ["LOC", "MISC", "O", "ORG", "PER"]

def remove_iob_tag(tag):
    clean_tag = tag.split('\n')[0]
    clean_tag = clean_tag.split('-')
    if len(clean_tag) > 1:
        return clean_tag[1]
    return clean_tag[0]

inputFileLines = inputFile.readlines()

y_true = []
y_pred = []
docs = []
sent = ""

# build sentences from CoNLL format
for index in range(0, len(inputFileLines)):

    current_line_token = inputFileLines[index].rpartition('\t')[0]
    current_line_tag = inputFileLines[index].rpartition('\t')[2]

    if current_line_tag == '\n':
        docs.append(sent)
        sent = ""

    elif index == len(inputFileLines)-1:
        sent += current_line_token
        docs.append(sent)
        sent = ""
        y_true.append(remove_iob_tag(current_line_tag))

    else:
        sent += current_line_token + " "
        y_true.append(remove_iob_tag(current_line_tag))

print(f'\nNumber of sentences: {len(docs)}')

# perform NER in the sentences
for sentence in docs:
    doc = nlp(sentence)
    for word in doc:
        if word.ent_iob_ == 'O':
```

```
            obj = f'{word}\t{word.ent_iob_}\n'
            y_pred.append(f'{word.ent_iob_}')
        else:
            obj = f'{word}\t{word.ent_iob_}-{word.ent_type_}\n'
            y_pred.append(f'{word.ent_type_}')
        exportFile.write(obj)
    exportFile.write('\n')

print(f'\n\ny_true length: {len(y_true)} \ny_pred length: {len(y_pred)} \n\n')

cm, ax, pyplot = plot_confusion_matrix(y_true, y_pred, classes, normalize=True)
pyplot.show()
```

Figure 4.7 shows the confusion matrix for the English NER model using a RoBERTa transformer with CoNLL03's test set, which achieved the best results during training among other transformers used for English. It is possible to observe the classes in which the model miss-classified, as well as the classes with the best classification rate. For instance, in 5% of cases, the model classified a word as "LOCATION" when it was, in fact, an "ORGANIZATION". The highest miss-classification rate occurred for "ORG-MISC" tokens, achieving a value of 6%. However, the model still performed very well, with "PERSON" and "LOCATION" reaching an accuracy of 96% and 94%, respectively.

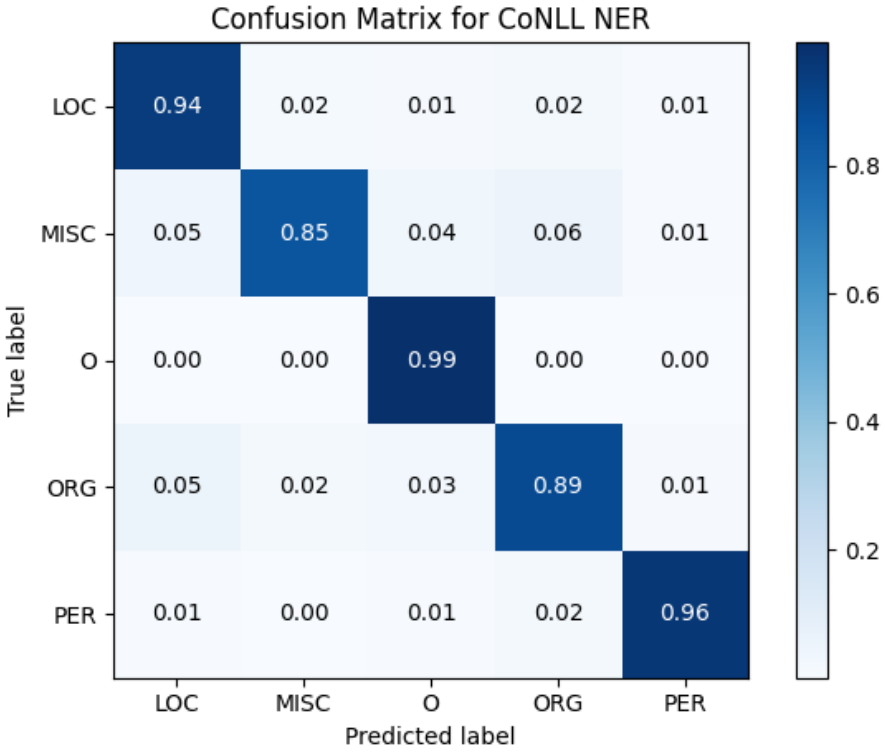

Figure 4.7: Confusion Matrix for CoNLL NER.

4.2 RELATION EXTRACTION SETUP

The hyper-parameters used for training the DBPedia, Wiki and TACRED RE models, presented in Table 4.9, yielded the results shown in Table 4.10, with a value of 84% for Micro F1 score on the test set for DBPedia RE model, 88% for Micro F1 score for Wiki RE model and 79% for the TACRED RE model, respectively. All models were trained based on Sentence-Level Relation Extraction, with a BERT-based transformer for the target language. Appendix II contains further details about the fine-tuning process for these models, including some application examples and codes.

Table 4.9: Hyper-parameters Values used to Train the RE Models.

Hyper-parameter	**Value**
Number of epochs	3
Batch size	64
Max sentence length	128
Optimizer	AdamW with bias correction
AdamW ϵ	1×10^{-8}
Learning rate	2×10^{-5}
Metric	Micro F-Score
Entity representation	Entity Marker

Table 4.10: Overall results for RE for DBPedia, Wiki and TACRED.

Dataset	**Transformer**	**Precision**	**Recall**	**F-Score**
DBPedia	BERT-Base-Portuguese	87.3	82.1	84.6
Wiki	BERT-Base	91.6	84.8	88.1
TACRED	BERT-Base	81.5	76.6	79.0

Figure 4.8 shows the F-Score and validation loss values per epoch during training. Preliminary analysis based on these outputs suggest that increasing the number of epochs for training may yield even better F-Score values for all three models, in particular for the TACRED model, which still has a high validation loss value in the final epoch. The Wiki model presented the lowest validation loss in the last epoch (below 0.1), suggesting that adding one or two epochs may be enough, while adding more than that may result in over-fitting.

According to recent works, the following entity representation techniques are used to represent entities in a sentence for relation classification:

- **Entity mask:** This technique introduces new special tokens to mask the subject and object entities in the original text. Zhang *et al.* [75] claim that this technique prevents the relation classifier from over-fitting specific entity names, leading to more generalizable inference;
- **Typed entity mask:** It extends the entity masking by also including the entities' NER type in the sentence representation, therefore increasing the number of special tokens required to represent both entity spans and entity classes.

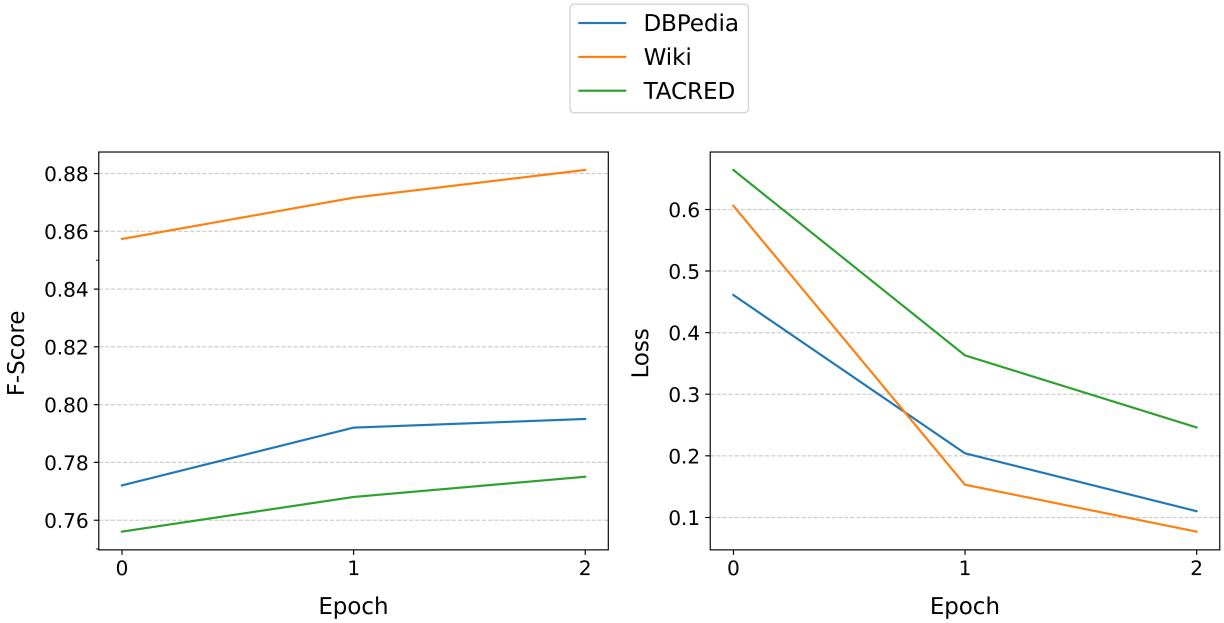

Figure 4.8: F-Score and Loss results per epoch for DBPedia, Wiki and TACRED models.

- **Entity marker:** This technique introduces special token pairs `[E1]`, `[/E1]` and `[E2]`, `[/E2]` to enclose the subject and object entities [49];
- **Entity marker (punct):** This technique is a variant of entity marker that encloses entity spans using punctuation. The main difference from the previous technique is that this one does not introduce new special tokens into the model's reserved vocabulary [105];
- **Typed entity marker:** This technique further incorporates the NER classes into entity markers. It introduces new special tokens, where the NER type is given by a named entity tagger, modifying the input [106];
- **Typed entity marker (punct):** A variant of the typed entity marker technique was proposed by [73], which marks the entity span and entity types without introducing new special tokens, using some characters to this end, including "@" and "#", for example.

For this baseline RE setup, it was used an Entity Marker (EM) entity representation technique, which marks both subject and object entities' boundaries in a sentence using special tokens that are added to the pre-trained language model's vocabulary. Table 4.11 compares each entity representation technique given the same input example ("Bob was born in Brazil").

The work of [107] explores and discusses the application of entity masking based on different RE benchmarks, arguing that this technique is useful to avoid memorization of entities that are frequently mentioned in the datasets. However, preliminary experiments with the benchmarks used in this thesis did not show relevant impacts in performance when masking entity mentions during training. Besides, the use of the punctuation-based techniques was also not explored due to the fact that although it does not add new special tokens to the model's reserved vocabulary, further modifications in the source code are required to correctly use punctuation to delimit entity spans and classes, which is not a simple process for practical and complex applications. Hence, it was chosen to use the entity marker technique in the baseline RE setup

for the supervised experiments, whose results are compared later with the results obtained with the typed entity marker technique in Section 4.7.

Table 4.11: Entity representation techniques.

Method	Input Example
Entity mask	`[SUBJ]` was born in `[OBJ]`.
Typed entity mask	`[SUBJ-PER]` was born in `[OBJ-LOC]`.
Entity marker	`[E1]` Bob `[/E1]` was born in `[E2]` Brazil `[/E2]`.
Entity marker (punct)	@ Bob @ was born in # Brazil #.
Typed entity marker	<S:PER> Bob </S:PER> was born in <O:LOC> Brazil </O:LOC>.
Typed entity marker (punct)	@ * PER * Bob @ was born in # ∧ LOC ∧ Brazil #.

4.2.1 Two-step Relation Extraction process

In order to enhance the performance of RC pipelines, some works proposed a two-step process for relation extraction. This process consists of deciding whether there exists a relation between a pair of entities prior to identifying the specific relation in question. The idea behind this solution is to avoid the output of a non-existing relation between entities. If a well-trained model is able to predict that there is no relation at all, the next step (relation identification) is skipped, reducing the amount of noise and miss-classified relations in the results. In order to experiment with this approach and evaluate whether it is relevant, based on the work of Wang *et al.* [72], new models were developed for each step, considering the specific tasks of detecting the existence of a relation between a pair of entities and classifying this particular relation, in that order.

In the first step, the relation label for all relational instances was set to 1, while the label for all N/A relations was set to 0. Preserving the `train/dev/test` splits, a new English model (based on the TACRED dataset) was trained for the binary classification task of relation detection. In the second step, new models were trained using both relational and non-relational instances, keeping the specific relation label for this step. Table 4.12 shows the results of the binary relation classification models plus the evaluation results for relation extraction models with and without a *no_relation* class.

Table 4.12: Evaluation of Binary and Multi-label classifiers for Relation Classification.

Classifier	Transformer	Precision	Recall	F-Score
Binary	BERT-Base	75.3	71.0	73.1
Multi-label + *no_relation*	BERT-Base	81.5	76.6	79.0
Multi-label	BERT-Base	93.7	82.3	87.6

From the results obtained with the two-step process, the difficulty of this process lies in the first step, which is to detect whether there exists a relation between a pair of entities, while identifying a specific relation seems to be less challenging. Figure 4.9 shows an example architecture that can be used to

face the relation extraction task as a two-step process. A similar approach was also recently proposed in [74], which showed that implementing different classifiers for different relation classes may improve the overall performance of the NLP pipeline. However, in practice, it is a costly architecture both in terms of time and computational resources consumption. Ultimately, the main objective to tackle in order to avoid miss-classifications is to include examples of non-related entities among the training and validation data whenever possible, so that the RC model will be able to differentiate between valid and non-existing relations for entities in a given text.

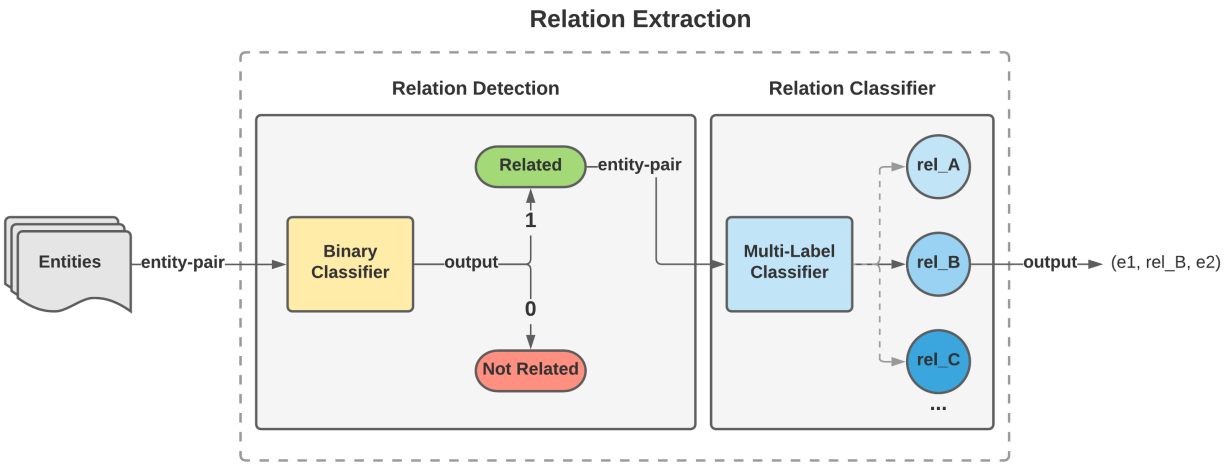

Figure 4.9: Two-Step Relation Extraction architecture.

4.2.2 Zero-Shot Relation Extraction

Although the advent of pre-trained language models have significantly decreased the need for greater amounts of labeled data for supervised learning NLP models to perform well, labeled data can still be scarce in a lot of different application scenarios. With that in mind, a new paradigm for RC emerged based on the Transfer Learning (TL) capability of transformer-based models like BERT called Zero-Shot learning. In this new paradigm, a RE model is able to extract relationships between pairs of entities in a text without any training data at all. In order to do this, these models usually make use of NLI or QA techniques.

Some experiments were made based on the work of [50], whose Zero-Shot RE approach is composed of three main components (Listing 4.4):

- **Relations:** a list of all possible relationships between entities;
- **Template:** a simple template example that represents the relationship between a pair of entities for each relation class; and
- **Valid conditions:** a list of relation classes and the corresponding possible entity classes for each relation class.

Listing 4.4: Zero-Shot NLI RE settings example.

```
{
    "relations": {
        "per:date_of_death",
        "per:schools_attended",
        "org:subsidiaries"
    },
    "template_mapping": {
        "per:date_of_death": ["{subj} died in {obj}"],
        "per:schools_attended": [
            "{subj} studied in {obj}",
            "{subj} graduated from {obj}"
        ],
        "org:subsidiaries": [
            "{obj} is a subsidiary of {subj}",
            "{obj} is a branch of {subj}"
        ],
        ...
    },
    "valid_conditions": {
        "per:date_of_death": ["PERSON:DATE"],
        "per:schools_attended": ["PERSON:ORGANIZATION"],
        "org:subsidiaries": [
            "ORGANIZATION:ORGANIZATION"
        ]
        ...
    }
}
```

In order to evaluate the performance of this approach, experiments were made based on each relations schema. Table 4.13 shows the results obtained in terms of Precision, Recall and F-Score using transformer models (RoBERTa) previously fine-tuned on a NLI task. Unlike the experiments made for supervised RE, the results presented here used a large version of transformers (with more layers and parameters) since there is no training step and, thus, the power of a larger model is leveraged to yield better results.

Since there is not a pre-trained NLI model currently available for the Portuguese language, a multilingual transformer was used to experiment with the DBPedia corpus, and a baseline English NLI model was used in Wiki and TACRED experiments. Regarding DBPedia, two experiments were made: in the first experiment, the multilingual transformer was used without any modification and the results for this scenario are shown on the first line (Table 4.13) and, for the second experiment, the same multilingual transformer was previously fine-tuned on a Portuguese NLI corpus (SICK-BR [54]) and the results for this scenario are shown on the second line (Table 4.13). It is possible to notice that fine-tuning the multilingual transformer on Portuguese substantially improved the performance on DBPedia, with an increase of over 9% in terms of F-Score.

For the experiment based on TACRED, the results are also satisfactory, especially considering the fact that the TACRED schema contains a greater number of node and relationship classes, although falling behind the supervised approach by about 6% in terms of F-Score. The Wiki experiment reveled a limitation of this approach, which is the fact that if a relation class cannot be modeled by a simple template or

Table 4.13: Results for Zero-Shot Relation Extraction with Natural Language Inference.

Schema	Model	Size	Precision	Recall	F-Score
DBPedia	$NLI_{RoBERTa}$	Large	83.1	83.0	83.0
DBPedia	$NLI_{RoBERTa\text{-}FT}$	Large	92.4	92.4	92.4
Wiki	$NLI_{RoBERTa}$	Large	52.6	52.6	52.6
TACRED	$NLI_{RoBERTa}$	Large	73.7	73.6	73.7

two or more relations share similar templates, the results are significantly affected and the performance of the model is evidently worse than that of a supervised model. For example, the Wiki schema contains the relations `participant` and `participant of`, which are very similar and also difficult to differentiate in terms of a simple template mapping. This scenario was purposely addressed here in order to state that Zero-Shot learning approaches have limitations and should not be applied to each and every context. Instead, they should be carefully studied based on each schema and application.

4.3 PIPELINE APPLICATION

Once the models were properly fine-tuned, it was possible to apply the full pipeline architecture to process text data and store them in a graph database. To better understand the processing flow, six examples will be presented, three for Portuguese (described in Section 4.5) and three for English (described in Section 4.6).

Figure 4.10 shows the complete flow of NLP for these examples. Firstly, the data is collected and it goes through a forensics process to extract the text data that will be used in the Neural Information Extraction pipeline. The input text data is analysed to determine in which language it was written. If English is the language of the data, a coreference resolution model may be applied to enhance entity recognition. The next step is to input named entities manually or by following a pattern or rule in order to obtain more entities, although being an optional step. Then, a specific NER model is selected for better performance based on the language determined in the previous step. Once the named entities are recognized and extracted, a relationship model is applied between them, generating an output in the form of a JSON file that can then be easily imported into a graph knowledge application or database.

For better post-processing information retrieval, each node (entity) and edge (relationship) contain some metadata. Regarding the nodes, their entity types and the documents or texts they originated from are stored. For edges, the relationship type and its confidence level is stored for further analysis. Hence, with the aid of a query language, it is possible to filter relationships and extract information in a more granular way. Further details about the implementation of the information extraction pipeline in the form of an API are discussed in Appendix III, and the graph DB setup, including importing, filtering and visualization of nodes and links, is discussed in Appendix IV, respectively.

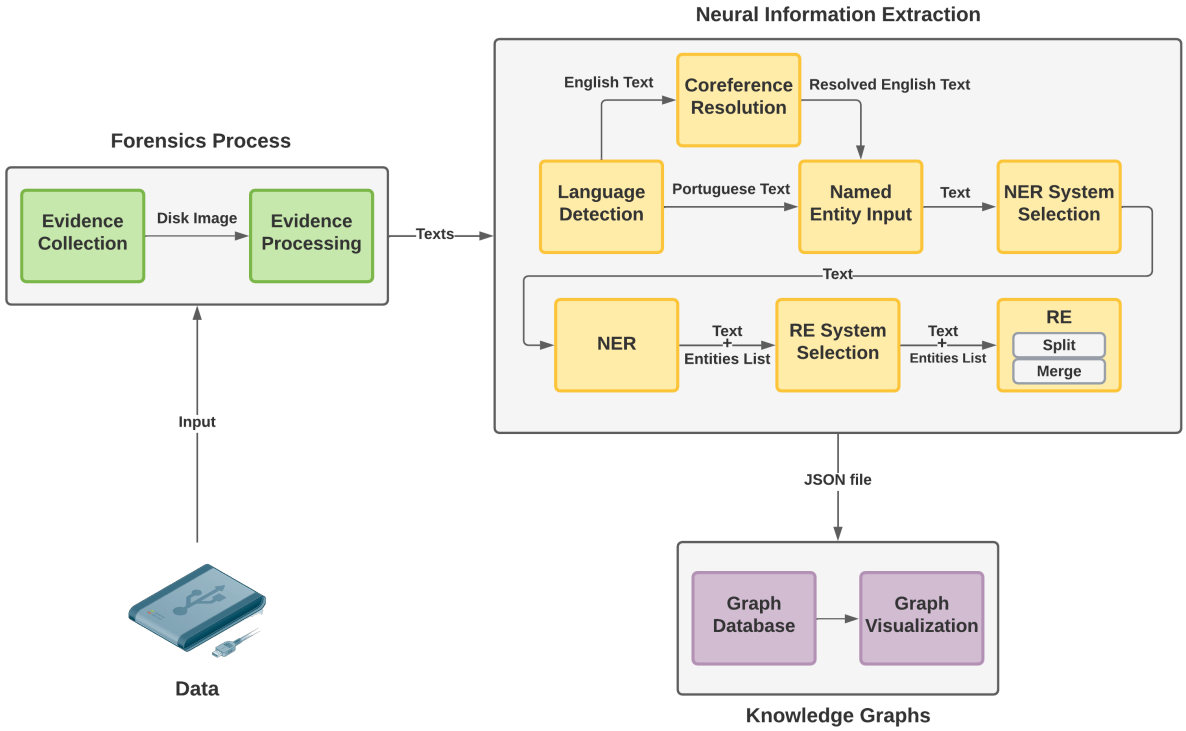

Figure 4.10: Pipeline's application flow.

4.4 DATA ACQUISITION

For data acquisition (disk image creation), it was used FTK (Forensic Toolkit) Imager. The target device was a USB (Universal Serial Bus) drive containing a single folder. Figure 4.11 shows the output of the disk image creation process for this particular device, including the computed and reported hashes using MD5 (Message-Digest algorithm 5) and SHA1 (Secure Hash Algorithm 1) for the output file (*image.ad1*). It is possible to notice that both computed and reported hashes match, corroborating the requirement that the digital evidence should not be modified during this process.

Name	image.ad1
MD5 Hash	
Computed hash	1d5c51d7a2691e9e1d9a33107a8dd081
Report Hash	1d5c51d7a2691e9e1d9a33107a8dd081
Verify result	Match
SHA1 Hash	
Computed hash	6bd03a8da863d4dd417ce08f009631b1c52b5223
Report Hash	6bd03a8da863d4dd417ce08f009631b1c52b5223
Verify result	Match

Figure 4.11: Disk image creation output after processing with FTK Imager.

IPED was used for digital evidence processing. Listing 4.5 shows a command used to start a new case with the default processing options.

Listing 4.5: IPED's command to process digital evidence.

```
1 $ iped.exe -d image.dd -o output
```

In Listing 4.5, the `-d` parameter (`image.dd`) is the full path to the forensic image that will be processed and the `-o` parameter (`output`) is the folder where the case will be created (it must not exist or must be an empty folder). After the case is processed, it is possible to open the `IPEDSearchApp.exe` program located inside the output folder to start the case analysis, which will open an user interface application, as shown in Figure 4.12.

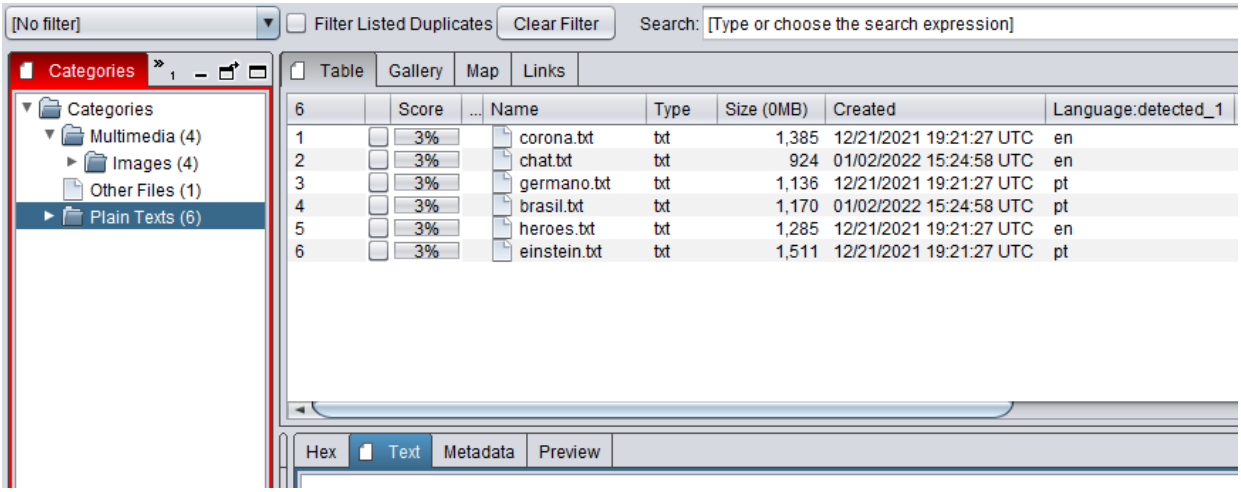

Figure 4.12: IPED's user interface with processed files.

4.5 SCENARIO 1: INFORMATION EXTRACTION FOR PORTUGUESE

For this example, suppose a forensics analysis was made over a suspect's computer. Thanks to IPED, all digital assets found were categorized and labelled according to its content type. Consequently, the text files (which are the target of this NLP work) could be easily accessed and processed to begin the information extraction. Three files were considered for this task: *einstein.txt*, which contains an overview of Albert Einstein's life, *germano.txt*, containing descriptions about the suspect's rivals in the crime world, and *brasil.txt*, which contains information about Brazil's characteristics and history.

4.5.1 Language Detection

IPED's language detection module stated that the language of the files' contents was Portuguese, therefore, in order to extract the named entities from the text, a Portuguese NER system was chosen, in this case, a NER system obtained from the Paramopama corpus using the BERTimbau [21] transformer.

4.5.2 Named Entity Input

Since it is suspected that there may be mentions of drugs in the text, especially because drug dealing is a relevant concern for law enforcement agencies, it was used a list containing the names of several drugs for the Named Entity Input step. Besides, IPED is also able to detect mentions of money quantities and cryptocurrencies' addresses in the text, so these entity types were also included in the analysis.

4.5.3 Coreference Resolution

As shown in Figure 4.10, this step was skipped for the Portuguese scenario for two reasons: it is an optional step and according to recent research, there are no publicly available tools for coreference resolution for Portuguese. That part is left for future work, as discussed in Chapter 5.

4.5.4 Named Entity Recognition and Extraction

Figures 4.13, 4.14 and 4.15 show the output of the NER system for all files (*einstein.txt*, *germano.txt* and *brasil.txt*). It is possible to see that the NER system was able to correctly identify several entities in the text, including the names of people, organizations, locations, drugs, cryptocurrencies' addresses, money values and dates. The system attributed different colors based on the entities' types, making it easier to differentiate them.

Figure 4.13: Named Entity Recognition for Scenario 1 (*einstein.txt*).

4.5.5 Relationship Model Selection and RE

After the entities were extracted, it is time to configure a RE model and apply it. Since it is a Portuguese text, the DBPedia model was used, following the schema presented in Figure 3.5, to detect relationships between entity pairs. Because the DBPedia RE model is based on Sentence-Level, the document was split into sentences for better performance while deciding which pairs of entities should yield better relationships.

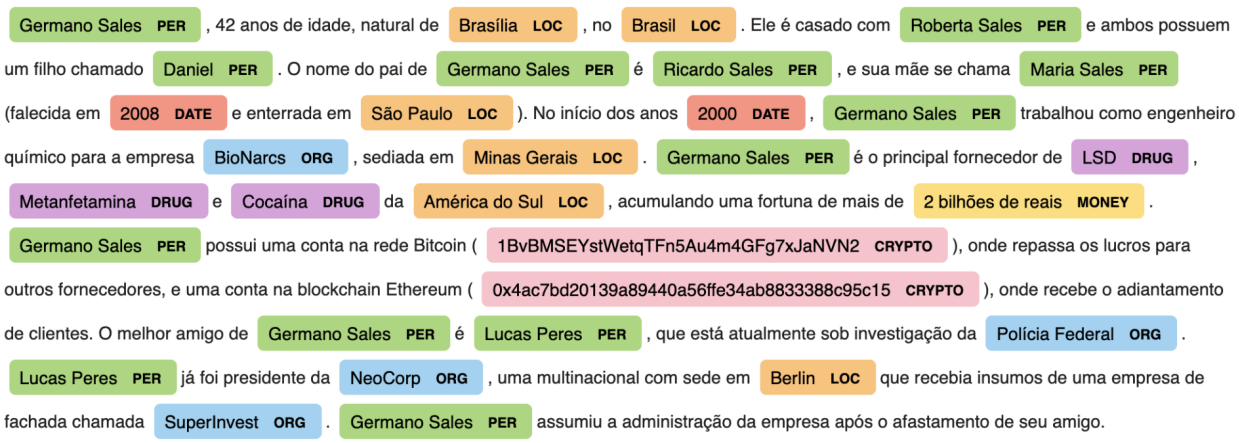

Figure 4.14: Named Entity Recognition for Scenario 1 (*germano.txt*).

Figure 4.15: Named Entity Recognition for Scenario 1 (*brasil.txt*).

4.5.6 Graph Visualization

Once the valid relationships are extracted, a JSON file is generated containing all entities and relationships from the text, for each example. This file can then be imported in a graph visualization application, like Neo4j [102]. Figures 4.16, 4.17 and 4.18 show a graph visualization based on the output of the information extraction system. Since the DBPedia schema only contains relationships for 3 entity types (Person, Organization and Location), the results were filtered to show only nodes with relationships between them, leaving other entities (like Drug, for example) out.

Despite the fact that the DBPedia schema is relatively small when compared with the others, the outputs of the proposed IE pipeline for the Portuguese texts were very informative, and the graphs generated have the potential to grasp hidden insights that may not be directly present in the text, but are still captured by the NLP models. Nevertheless, some limitations still exist when dealing with complex application scenarios. For instance, while calculating the relations to build the KG showed in Figure 4.18, the RE models sometimes confuse the direction of the relationship for the `locatedInArea` class, which may

cause wrong assumptions. These exceptions must be analysed carefully before submitting a final report for the case, but it is rather a limitation of the relation classes used for training and the relations schema adopted, rather than a performance issue with the RE model itself.

Figure 4.16: Scenario 1 (*einstein.txt*) Graph Output for DBPedia RE Model.

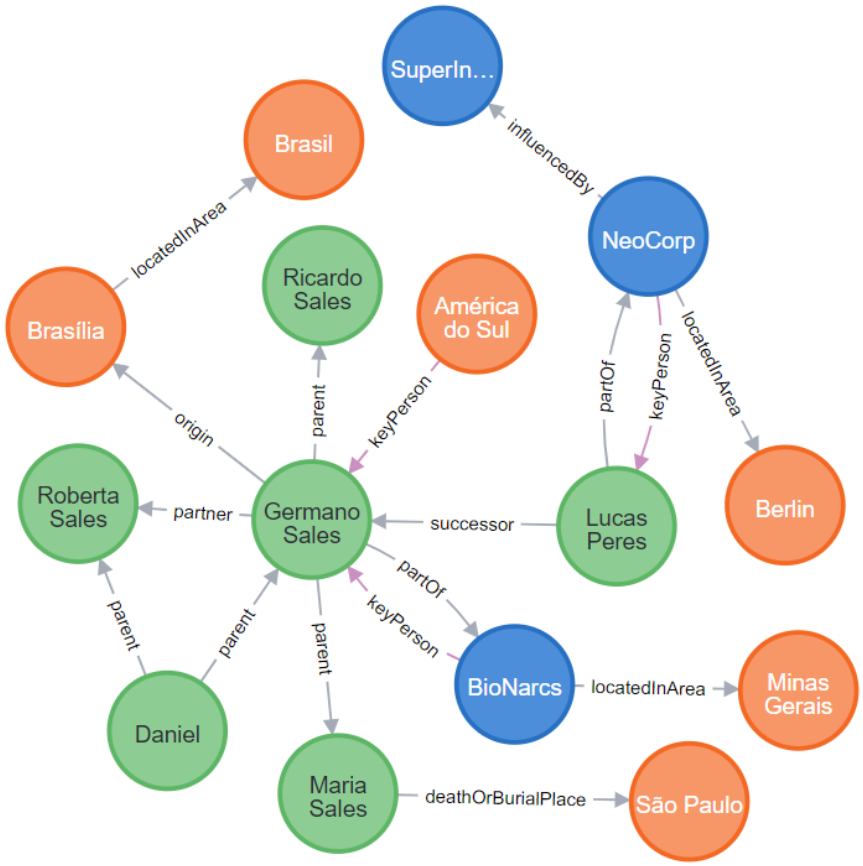

Figure 4.17: Scenario 1 (*germano.txt*) Graph Output for DBPedia RE Model.

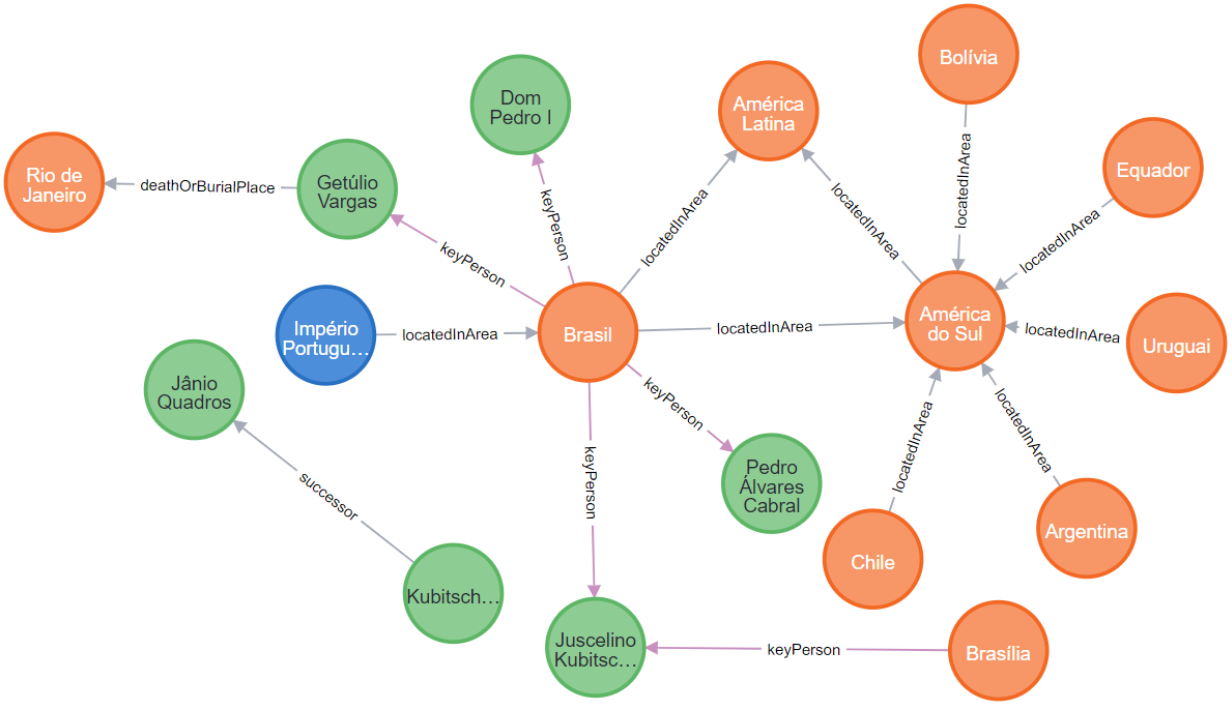

Figure 4.18: Scenario 1 (*brasil.txt*) Graph Output for DBPedia RE Model.

4.6 SCENARIO 2: INFORMATION EXTRACTION FOR ENGLISH

This second example is similar to the previous one, with two peculiarities: the addition of a coreference resolution step and two RE models for English. Suppose, now, that three other key files for the investigation were found on the suspect's computer, named *corona.txt*, *heroes.txt* and *chat.txt*.

4.6.1 Language Detection

IPED has identified that, unlike the previous files, these ones were written in English.

4.6.2 Named Entity Input

The first file's content (*corona.txt*) is about the Coronavirus, so a regular expression was used in the NEI step to detect all mentions of words that contain the term "virus", creating, thus, a new entity category for these words. For the second (*heroes.txt*) and third (*chat.txt*) files, there were no named entity inputs.

4.6.3 Coreference Resolution

Before feeding the NER system, the texts were submitted to a CR model, which intends to remove duplicate entity mentions and improve the overall results. Figure 4.19 shows the coreference chains for *corona.txt* identified by the CR system: chain 0, which refers to the word "Coronavirus", chain 1, which refers to David Tyrrel, and chain 2, which refers to the virologist June Almeida. Since chain 0 refers to the term and not the virus itself, it was decided not to resolve it in this example. The other chains, however, were resolved to the root term. For *heroes.txt* a bigger number of coreference chains were identified, as shown in Figure 4.20. The main references that were resolved for this file include mentions of "Wayne" and "Bruce" (resolved to "Bruce Wayne"), as well as "Clark" (resolved to "Clark Kent"), "Barry" (resolved to "Barry Allen"), "Oliver" (resolved to "Oliver Queen") and "Iris" (resolved to "Iris West"). The CR step was skipped for the *chat.txt* example because it is an example of an informal conversation with a complex text structure.

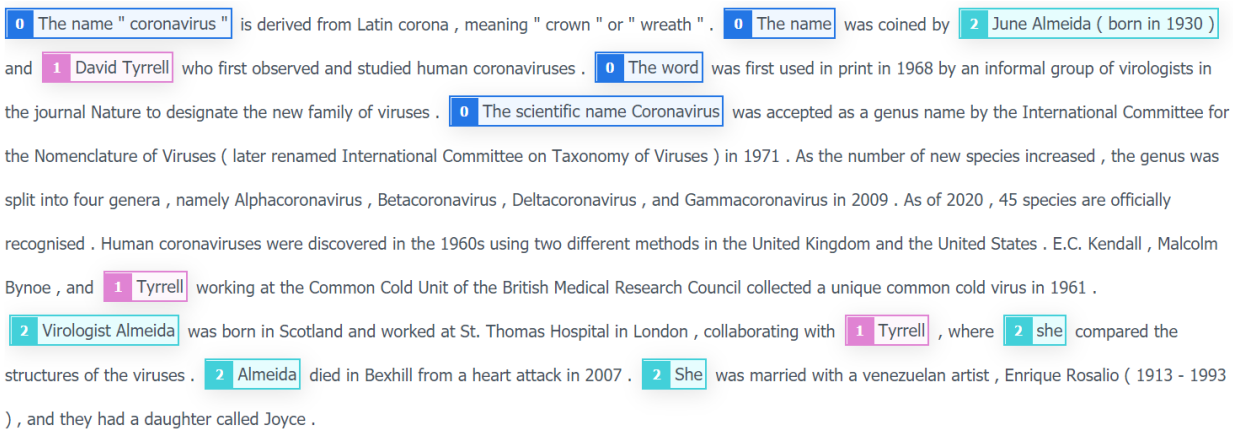

Figure 4.19: Scenario 2 (*corona.txt*) Coreference Resolution.

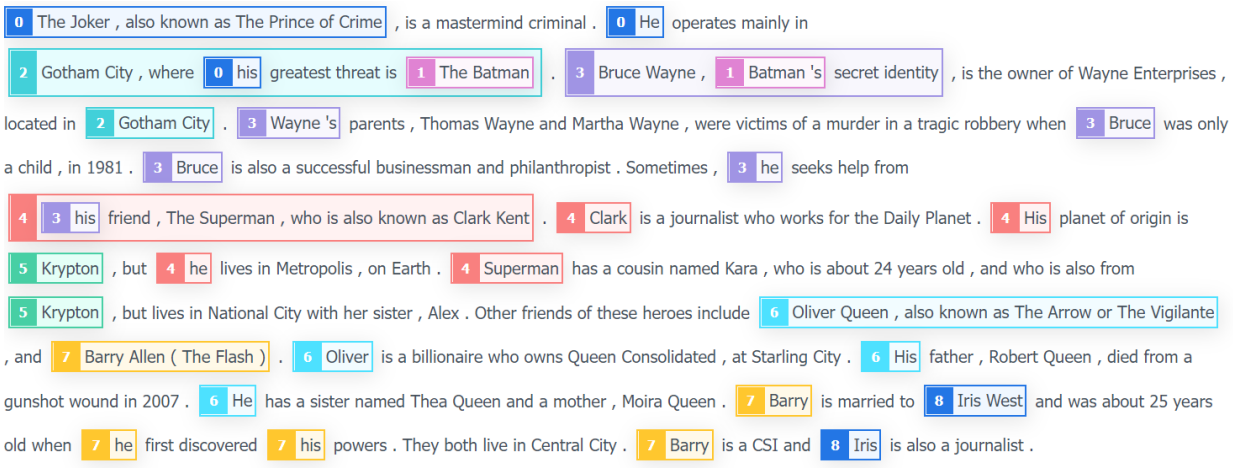

Figure 4.20: Scenario 2 (*heroes.txt*) Coreference Resolution.

4.6.4 NER Model Selection and Named Entity Extraction

The next step in our NLP flow is to select an appropriate NER model for the input text. For these examples, it was used System 1 from Table 4.5. Figures 4.21, 4.22 and 4.23 show the output of the NER system for the English examples. Again, it is possible to see that all relevant entities were correctly identified and classified according to their type, including the new Virus category.

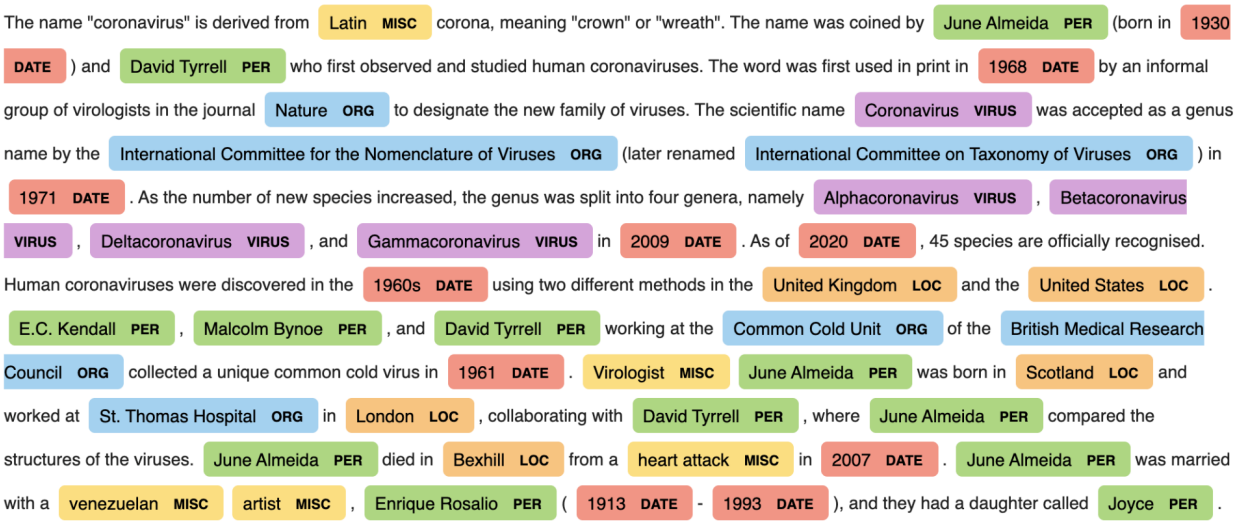

Figure 4.21: Named Entity Recognition for Scenario 2 (*corona.txt*).

4.6.5 Relationship Model Selection and RE

Now it is time to extract the relationships between the entities found. For this part, it was used two different models for English (Wiki and TACRED) and outputted their results separately, for each example (*corona.txt*, *heroes.txt* and *chat.txt*).

Figure 4.22: Named Entity Recognition for Scenario 2 (*heroes.txt*).

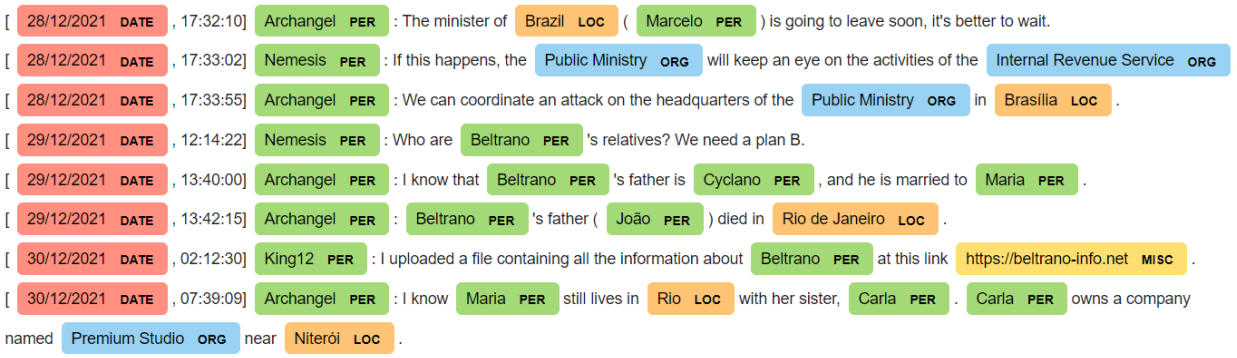

Figure 4.23: Named Entity Recognition for Scenario 2 (*chat.txt*).

4.6.6 Graph Visualization

For the *corona.txt* file, Figure 4.24 shows the graph visualization for the Wiki RE model, and Figure 4.25 shows the graph visualization for the TACRED RE model, respectively. It is possible to notice that the Wiki RE model was able to detect several relevant relationships, like the relatives of June Almeida, her field of work and work location, for example. The TACRED model also achieved good results, with the benefit of being able to detect relationships for Date entity types, like June's date of birth and date of death. This can be a key information when dealing with forensics and when trying to understand the chronology of facts under investigation, for example. For the *heroes.txt* file, Figure 4.26 shows the graph visualization for the Wiki RE model, and Figure 4.27 shows the graph visualization for the TACRED RE model. Both outputs achieved good results, with the detection of relevant relations, like the occupation, residence, work location and family members of the persons in the file's content. The TACRED RE model was able to extract the ages mentioned in the text for Barry Allen and Kara, the cause of death and date of death for Robert Queen and dates of death for Bruce's parents (not detected by the Wiki model). However, the Wiki model correctly detected Kara's residence in Krypton and Barry Allen's occupation as a CSI (both missed by the TACRED model). For the *chat.txt* file, Figures 4.28 and 4.29 show the graph visualization generated based on Wiki and TACRED schemas, respectively.

Often, in practice, it may be relevant to switch between different NER and RE systems in order to

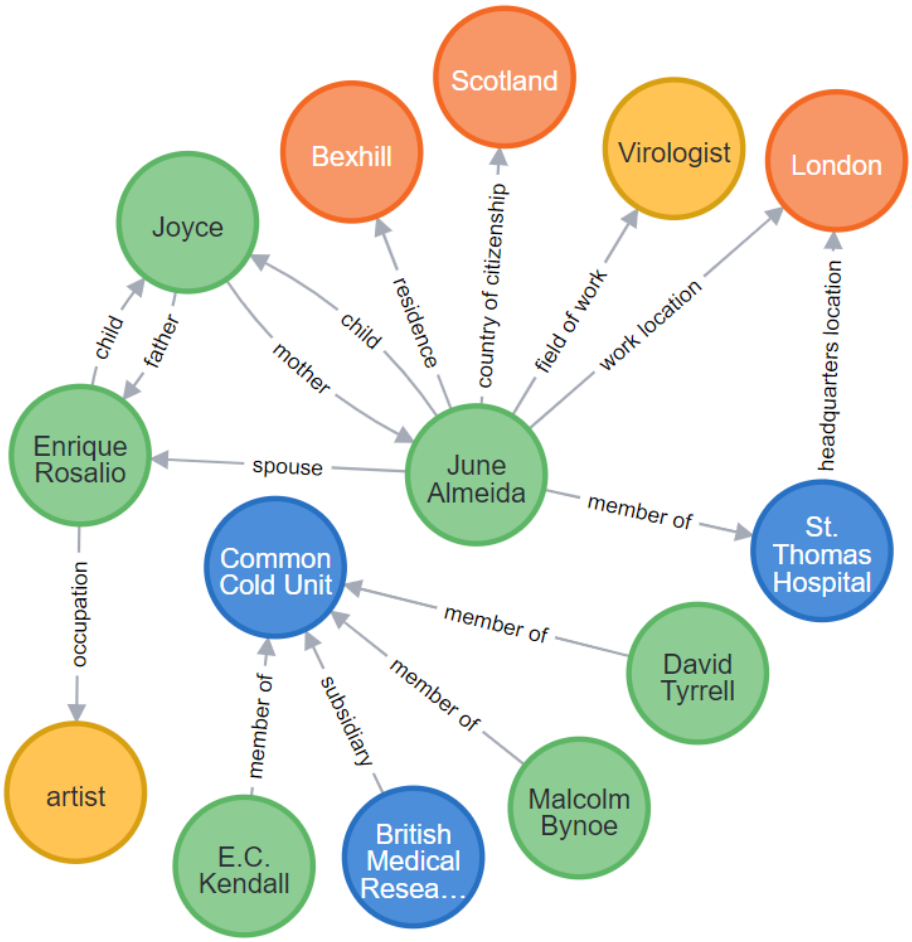

Figure 4.24: Scenario 2 (*corona.txt*) Graph Output for Wiki RE Model.

Figure 4.25: Scenario 2 (*corona.txt*) Graph Output for TACRED RE Model.

Figure 4.26: Scenario 2 (*heroes.txt*) Graph Output for Wiki RE Model.

Figure 4.27: Scenario 2 (*heroes.txt*) Graph Output for TACRED RE Model.

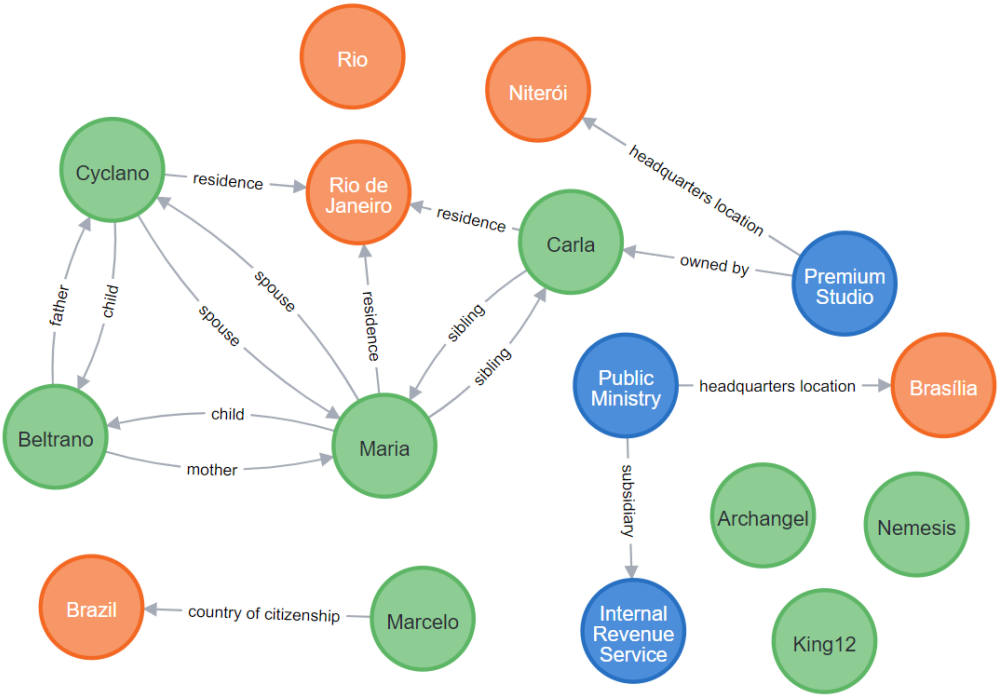

Figure 4.28: Scenario 2 (*chat.txt*) Graph Output for Wiki RE Model.

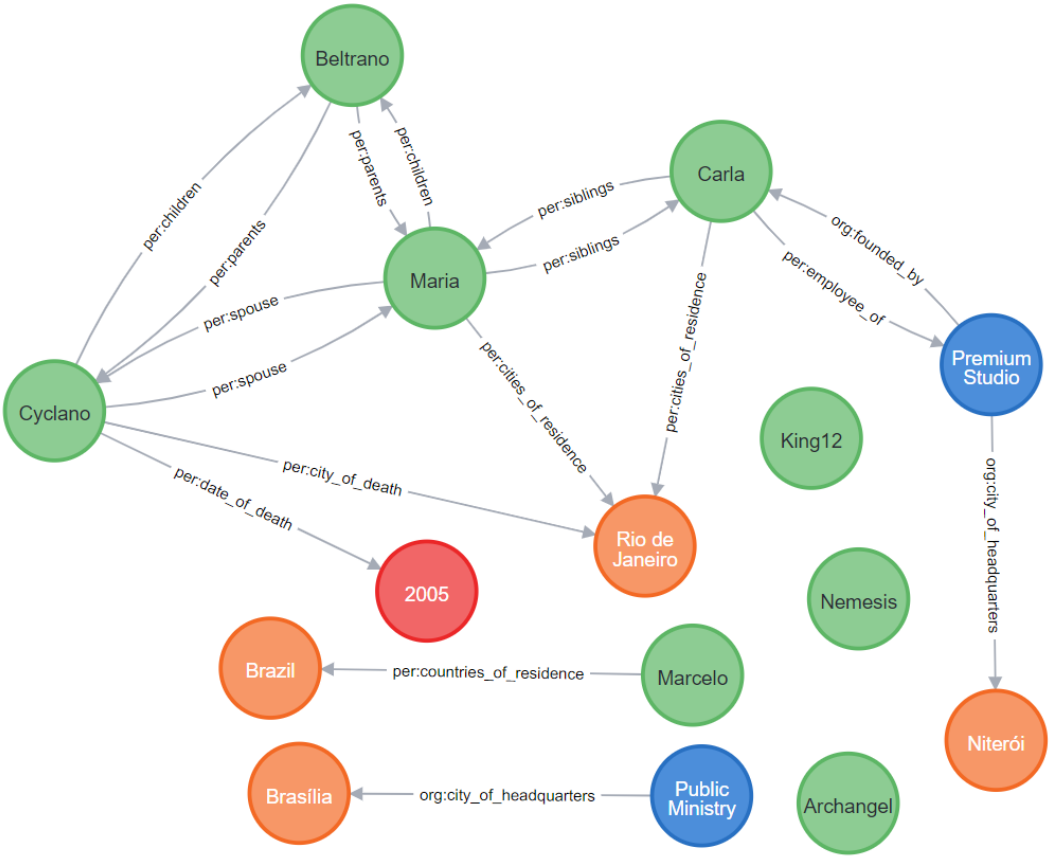

Figure 4.29: Scenario 2 (*chat.txt*) Graph Output for TACRED RE Model.

detect and analyse as many relations as possible, specially because longer texts tend to contain complex relations between entities. Even though the outputs showed in Figures 4.24 and 4.25 and in Figures 4.26 and 4.27 are similar, they can complement each other and provide better results.

4.7 COMPARISON WITH THE STATE-OF-THE-ART

In this section, the implementations of NER and RE models presented in this research are compared with some state-of-the-art models from the literature. Table 4.14 and Figure 4.30 show the comparisons of NER models, whereas Table 4.15 and Figure 4.31 show the comparisons of RE models.

Table 4.14: Comparison of Results for Named Entity Recognition with the state-of-the-art.

Corpus	Model	Size	F-Score	Reference
Mini HAREM	BERT-PT (*)	Base	**79.4**	–
	BERT-PT	Large	83.7	[21]
Paramopama	BERT-PT (*)	Base	**89.0**	–
	LSTM-CRF	–	90.5	[63]
	BERT-PT$_{HT}$ (*)	Base	**91.2**	–
LeNER-Br	LSTM-CRF	–	86.6	[63]
	BERT-PT (*)	Base	**89.2**	–
CoNLL03	RoBERTa (*)	Base	**92.1**	–
	BERT	Large	92.8	[3]
	FLERT	Large	94.0	[64]
	LUKE	Large	94.3	[24]
	ACE	Large	94.6	[65]
WNUT17	RoBERTa (*)	Base	**48.3**	–
	BERTweet	Base	56.5	[82]

(*) NER models developed in this work.

For the NER task, the best models developed based on Mini HAREM, Paramopama, LeNER-Br, CoNLL03 and WNUT17 are compared with the current state-of-the-art. No current relevant results for the First and Second HAREM collections were found, nor for the Portuguese WikiNER corpus, therefore these collections were not included in the comparisons. For the Paramopama dataset, the previous state-of-the-art (LSTM-CRF model) is compared with the developed model before hyper-parameters tuning (BERT-PT) and after hyper-parameters tuning (BERT-PT$_{HT}$). The fine-tuned model outperforms the previous state-of-the-art, achieving a new standard for Portuguese NER in this dataset.

For the CoNLL03 dataset, the developed model is compared with three others. The ACE model [65] is the current state-of-the-art for named entity recognition on CoNLL03. Nevertheless, it is important to point out that these models were fine-tuned for a longer period of time if compared with the implementations presented in this dissertation, since the number the number of epochs was limited to 20 in the experiments for practical reasons, while most of the state-of-the-art NER models usually adopt 50 epochs for training. In addition, the codes for FLERT [64], LUKE [24] and ACE [65] were released to the community, which

means that their results can be easily replicated and applied to different datasets with some modifications. Regarding the WNUT17 benchmark, the BERTweet model significantly outperforms the one presented in this work, achieving a F-Score of 56.5%. Nevertheless, BERTweet was pre-trained on a large domain-specific corpus, whereas in this research it was used a general-purpose model for the same task.

Table 4.15: Comparison of Results for Relation Extraction with the state-of-the-art.

Corpus	Model	Size	F-Score	Reference
DBPedia	KNN	–	55.6	[96]
	BERT-PT$_{EM}$ (*)	Base	**84.6**	–
	BERT-PT$_{TEM}$ (*)	Base	**86.6**	–
TACRED	BERT	Base	66.0	[81]
	ERNIE	Base	67.9	[81]
	BERT$_{EM}$ (*)	Base	**69.9**	–
	BERT$_{EM}$	Large	70.1	[49]
	SpanBERT	Large	70.8	[18]
	BERT$_{EM}$+MTB	Large	71.5	[49]
	BERT$_{TEM}$ (*)	Base	**71.8**	–
	LUKE	Large	72.7	[24]
	RoBERTa$_{TEM}$	Large	74.6	[73]
	RECENT	Large	75.2	[74]
M-TACRED	BERT$_{EM}$ (*)	Base	**79.0**	–
	BERT$_{TEM}$ (*)	Base	**80.5**	–

(*) RE models developed in this work.

For the task of relation extraction, since the Wiki dataset used in this work is not an official benchmark, it was not adopted for comparison. In addition, because the implementation of the TACRED dataset required some modifications in the original dataset for practical purposes, as explained in Section 3.3, two new models were trained with the original dataset in order to establish a fair comparison with the current state-of-the-art. The first TACRED model, BERT$_{EM}$, uses an Entity Marker (EM) described by [49], which is the same method used previously with all the supervised RE models implemented in this work. The second TACRED model, BERT$_{TEM}$, uses a Typed Entity Marker described in [73]. The results on the modified version of TACRED (M-TACRED) were also included for comparison at the bottom of Table 4.15. For M-TACRED, it was included the result of the BERT model with Entity Marker from Section 4.2 and the result for the same base model trained with Typed Entity Marker, which achieved an improvement of 1.5% in the F-Measure.

For the DBPedia RE models comparison, no other works that used the same dataset and the same group of relation classes were found during this research. Because of that, only the original implementation [96] was used for comparison. The implemented BERT-PT$_{TEM}$ model significantly outperforms the previous state-of-the-art for the original DBPedia model, which was based on a K-Nearest-Neighbors (KNN) model. Both of the implemented TACRED RE models also achieved good performances, outperforming other approaches, but still a little behind the current state-of-the-art. However, the models implemented in this work are based on BERT-Base, which has significantly less parameters than BERT-Large and other large transformers.

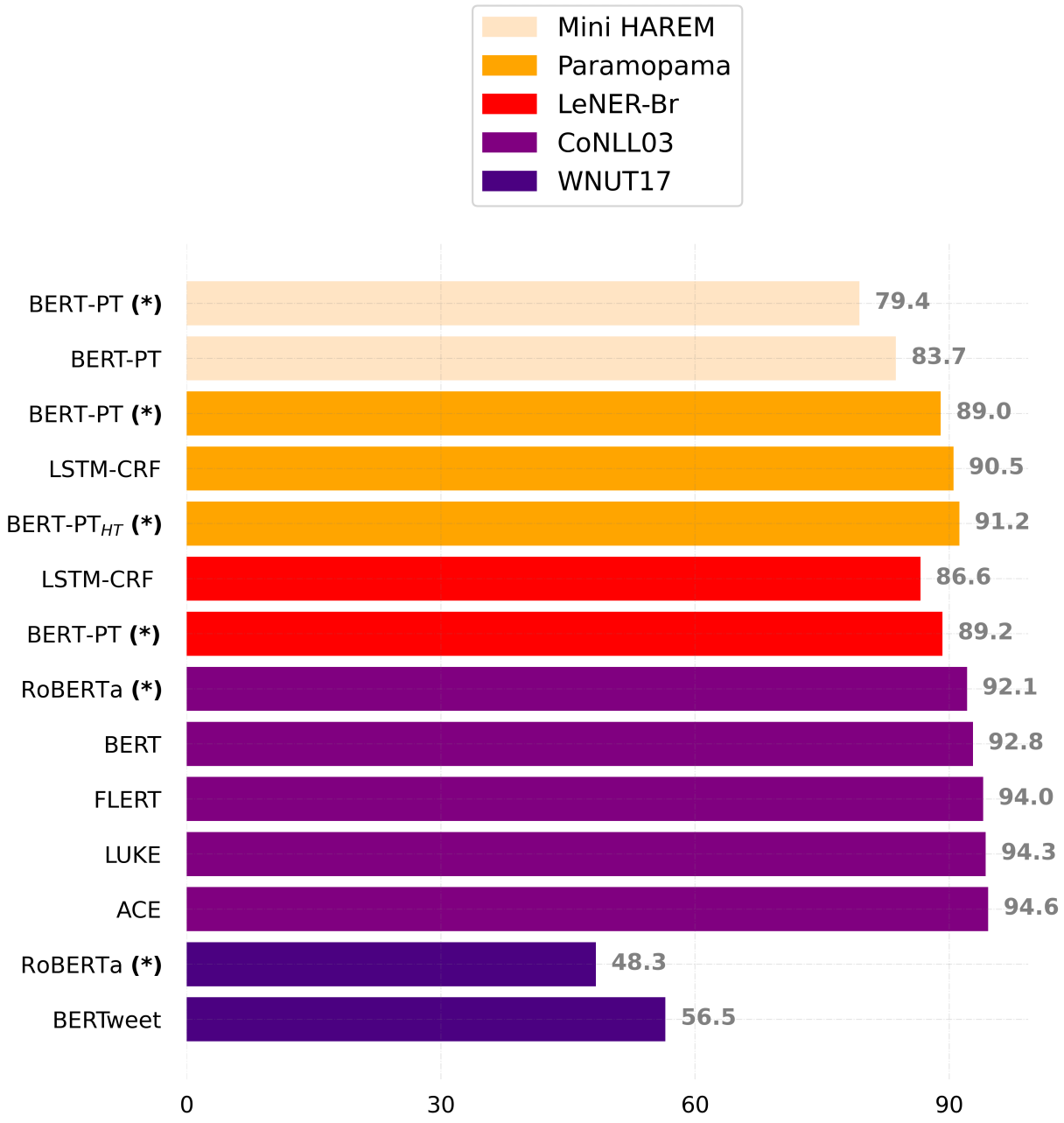

Figure 4.30: Comparison between the NER models developed, represented with an asterisk symbol (*), and the state-of-the-art. The Y axis show different state-of-the-art models and their corresponding F-Scores are shown in the X axis. Different datasets are represented by different colors.

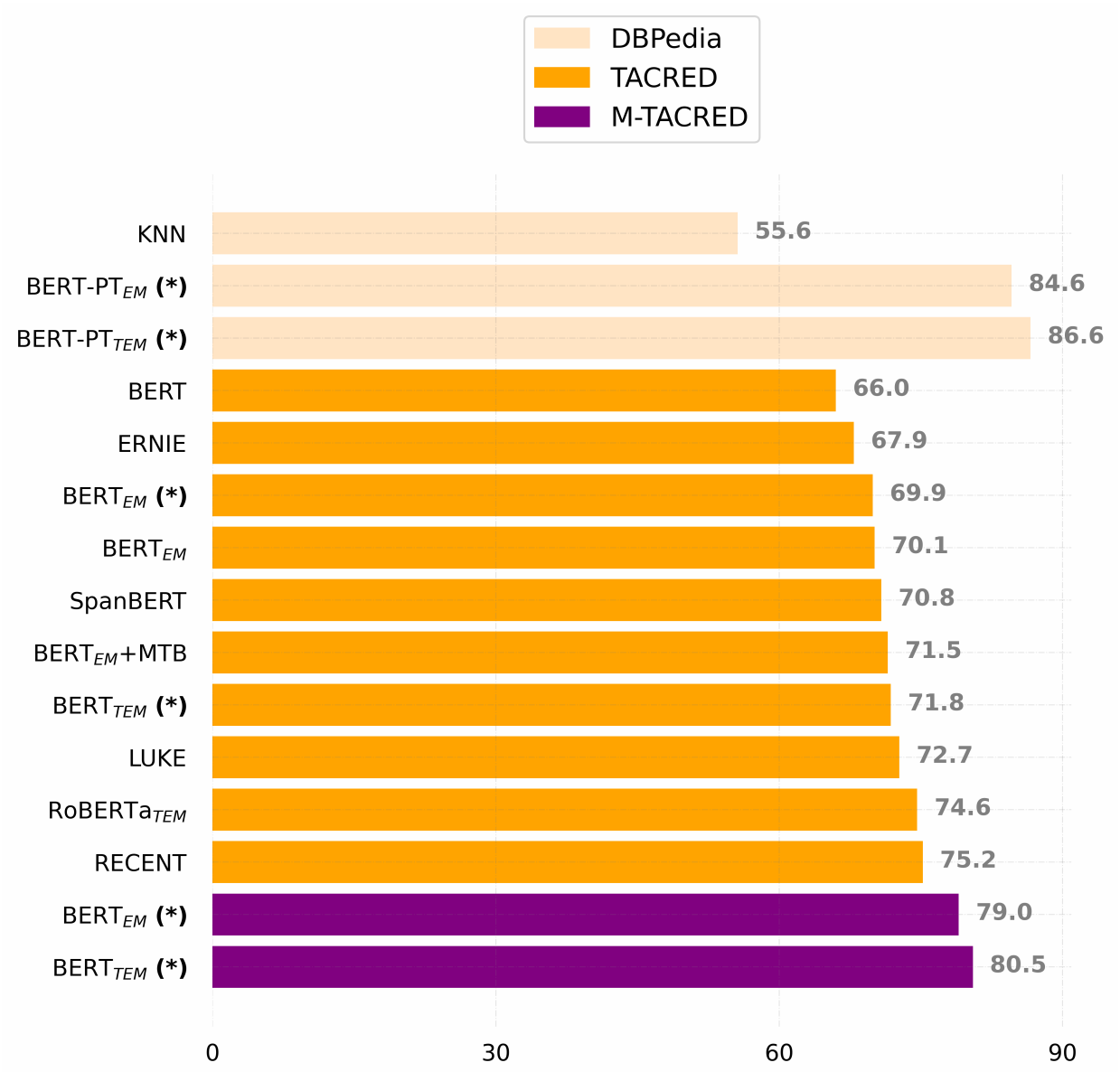

Figure 4.31: Comparison between the RE models developed, represented with an asterisk symbol (*), and the state-of-the-art. The Y axis show different state-of-the-art models and their corresponding F-Scores are shown in the X axis. Different datasets are represented by different colors.

5 CONCLUSION

A forensics investigation process is filled with details and often manual or repetitive operations. Technological solutions applied to this area help investigators and analysts with the automation of tasks, some of which leverage the power of machine learning approaches to solve specific problems or speed-up their conclusion. Nevertheless, most of these solutions did not keep up with the recent advances in ML and NLP, using algorithms that are either outdated, slow or unsuitable for the current dynamics of the cyber world.

This work proposed a systematic solution for information extraction through natural language processing of text data. It was showed that Named Entity Recognition (NER) is a common task to gather specific token types like persons, organizations and locations. The advent of transformers made it easier to create state-of-the-art performing models fine-tuned for multiple tasks, including NER and Relation Extraction (RE).

In spite of that, these cutting-edge models are not yet implemented for some applications, including those of some forensic tools. Furthermore, according to recent research, there are not many solutions available for Portuguese so far, with most of the efforts being concentrated in the English language. It was also demonstrated, however, that the availability of new transformer models for Portuguese, like BERTimbau, with the appropriate corpora, made it possible to fine-tune new models and achieve state-of-the-art performance in any language.

In this sense, several NER models for both Portuguese and English were trained with great performances, with the two best ones being the Portuguese model trained on Paramopama (F-Score of 91%) and the English model trained on CoNLL03 (F-Score of 92%). For the Relation Extraction part, it was trained a new model for Portuguese based on the DBPedia corpus, achieving a F-Score of 86% on the supervised RE task. Regarding the English language, several experiments were made based on the TACRED benchmark and a corpus derived from Wikipedia (Wiki), which also achieved good performance. It was also explored and discussed a new paradigm for RE that uses Natural Language Inference and does not require training data. Such approach may be useful when dealing with domain-specific scenarios in which there is a limited number of examples per relationship class or none of them at all.

Finally, six application examples were presented with different text inputs for Portuguese and English in order to demonstrate the implementation of the neural information extraction pipeline from a forensics perspective. The results (outputs) were organized in the form of knowledge graphs, stored in a graph database, in which nodes represent named entities and links represent semantic relations between the entities found in the texts. This structure makes it possible to standardize the representation of relevant information from unstructured data (texts), facilitating the analysis process as a whole.

It is important to keep in mind that, unlike some of the experiments presented in this work, whose texts were carefully checked for grammar and typing errors, for a lot of applications and scenarios making these previous checks before forwarding the text to an NLP model may be either not possible or undesirable, which can lead to wrong or ambiguous classifications. The usage of a two-step process for Relation Extraction, as discussed in Section 4.2.1, alongside a more granular schema for named entities can help mitigate this effect on complex application scenarios.

5.1 FUTURE WORK

Future work for further improvements with information retrieval using NLP systems is still in progress. In [108], it is shown how to create a massive NER corpus for Portuguese using open source datasets. Low amounts of labeled data is usually the bottleneck of many NLP downstream tasks, thus, a massive corpus could help improve the results. Feature engineering, alongside with hyper-parameters tuning, can also provide better models and scores, and there are other approaches for relation extraction that could be considered as well, like open information extraction, to extend the relations detected and build a more comprehensive graph database.

More efforts are also desirable to fine-tune NER and RE models for other languages. The CoNLL 2003 [66] task also provided a NER dataset for German, and CoNLL 2012 shared task [85] provided datasets for Chinese and Arabic that could be used to develop and test NER systems for these languages. For Coreference Resolution, according to most recent research, there are no available solutions for Portuguese, and it can be inferred that other Latin-derived languages, like Spanish and Italian, are equally challenging for this task.

Some ideas on future works regarding this thesis are presented below:

- Combine different NER and RE corpora in order to provide more training data;
- Integrate other relation extraction approaches into the information extraction pipeline, like open information extraction and document-level relation extraction;
- Train and develop NER and RE models for other languages;
- Integrate a functional coreference resolution model for Portuguese.

BIBLIOGRAPHY

1 KOHN, M. D.; ELOFF, M. M.; ELOFF, J. H. Integrated digital forensic process model. *Computers & Security*, Elsevier, v. 38, p. 103–115, 2013. ISSN 0167-4048. Doi: <https://doi.org/10.1016/j.cose.2013.05.001>.

2 ALLES, V. J.; GIOZZA, W. F.; ALBURQUERQUE, R. de O. Natural language processing to classify named entities of the Brazilian Union Official Diary. In: IEEE. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2018. p. 1–6. Doi: <https://doi.org/10.23919/CISTI.2018.8399215>.

3 DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. Available: <http://arxiv.org/abs/1810.04805>.

4 SPACY. 2021. <https://spacy.io>. Accessed: 2021-09-20.

5 JÚNIOR, C. M.; MACEDO, H.; BISPO, T.; SANTOS, F.; SILVA, N.; BARBOSA, L. Paramopama: a Brazilian-Portuguese corpus for named entity recognition. *Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, 2015.

6 RODRIGUES, F. B.; GIOZZA, W. F.; ALBUQUERQUE, R. de O.; VILLALBA, L. J. G. Natural language processing applied to forensics information extraction with transformers and graph visualization. *IEEE Transactions on Computational Social Systems*, p. 1–17, 2022. Available: <https://doi.org/10.1109/TCSS.2022.3159677>.

7 PALMER, G. et al. A road map for digital forensic research. In: *First digital forensic research workshop, utica, new york*. [S.l.: s.n.], 2001. p. 27–30.

8 REITH, M.; CARR, C.; GUNSCH, G. An examination of digital forensic models. *International Journal of Digital Evidence*, v. 1, n. 3, p. 1–12, 2002.

9 BREZINSKI, D.; KILLALEA, T. *RFC3227: Guidelines for Evidence Collection and Archiving*. [S.l.]: RFC Editor, 2002.

10 LYLE, J. R. et al. Nist cftt: testing disk imaging tools. In: *Proceedings of Second Digital Forensic Research Workshop*. [S.l.: s.n.], 2002.

11 KUMAR, K.; SOFAT, S.; JAIN, S.; AGGARWAL, N. Significance of hash value generation in digital forensic: A case study. *International Journal of Engineering Research and Development*, Citeseer, v. 2, n. 5, p. 64–70, 2012.

12 CARRIER, B.; SPAFFORD, E. An event-based digital forensic investigation framework. *Digital Investigation*, Elsevier, 2004.

13 KARTTUNEN, L.; CHANOD, J.-P.; GREFENSTETTE, G.; SCHILLE, A. Regular expressions for language engineering. *Natural Language Engineering*, Cambridge University Press, v. 2, n. 4, p. 305–328, 1996.

14 WU, S.; MANBER, U. Agrep–a fast approximate pattern-matching tool. In: *Usenix Winter 1992 Technical Conference*. [S.l.: s.n.], 1992. p. 153–162.

15 THUEN, C. Understanding counter-forensics to ensure a successful investigation. *Department of Computer Science, University of Idaho, Moscow, Idaho (pdfs. semanticscholar. org/d5b6/b658d9178dbcdf33e095a53c45b4f7a43fc8. pdf)*, Citeseer, 2007.

16 IPED Tool. 2021. <https://github.com/sepinf-inc/IPED>. Accessed: 2021-09-20.

17 IPED Interface. 2021. <https://servicos.dpf.gov.br/ferramentas/IPED/3.14.5/IPED-Manual_pt-BR.pdf>. Accessed: 2021-09-20.

18 JOSHI, M.; CHEN, D.; LIU, Y.; WELD, D. S.; ZETTLEMOYER, L.; LEVY, O. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, MIT Press, v. 8, p. 64–77, 2020.

19 SANH, V.; DEBUT, L.; CHAUMOND, J.; WOLF, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019. Available: <https://arxiv.org/abs/1910.01108>.

20 LAN, Z.; CHEN, M.; GOODMAN, S.; GIMPEL, K.; SHARMA, P.; SORICUT, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019. Available: <https://arxiv.org/abs/1909.11942>.

21 SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. BERTimbau: pretrained BERT models for Brazilian Portuguese. In: SPRINGER. *Brazilian Conference on Intelligent Systems*. [S.l.], 2020. p. 403–417. Doi: <https://doi.org/10.1007/978-3-030-61377-8_28>.

22 LIU, Y.; OTT, M.; GOYAL, N.; DU, J.; JOSHI, M.; CHEN, D.; LEVY, O.; LEWIS, M.; ZETTLEMOYER, L.; STOYANOV, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. Available: <https://arxiv.org/abs/1907.11692>.

23 CLARK, K.; LUONG, M.-T.; LE, Q. V.; MANNING, C. D. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020. Available: <https://arxiv.org/abs/2003.10555>.

24 YAMADA, I.; ASAI, A.; SHINDO, H.; TAKEDA, H.; MATSUMOTO, Y. Luke: deep contextualized entity representations with entity-aware self-attention. *arXiv preprint arXiv:2010.01057*, 2020. Available: <https://arxiv.org/abs/2010.01057>.

25 VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008.

26 PARIKH, A. P.; TÄCKSTRÖM, O.; DAS, D.; USZKOREIT, J. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016. Available: <https://arxiv.org/abs/1606.01933>.

27 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

28 SHIV, V.; QUIRK, C. Novel positional encodings to enable tree-based transformers. *Advances in Neural Information Processing Systems*, v. 32, 2019. Available: <https://proceedings.neurips.cc/paper/2019/file/6e0917469214d8fbd8c517dcdc6b8dcf-Paper.pdf>.

29 COLLOBERT, R.; WESTON, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In: *Proceedings of the 25th international conference on Machine learning*. Association for Computing Machinery, 2008. p. 160–167. ISBN 9781605582054. Available: <https://doi.org/10.1145/1390156.1390177>.

30 BELTAGY, I.; LO, K.; COHAN, A. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019. Available: <https://arxiv.org/abs/1903.10676>.

31 HUANG, K.; ALTOSAAR, J.; RANGANATH, R. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019. Available: <https://arxiv.org/abs/1904.05342>.

32 LEE, J.; YOON, W.; KIM, S.; KIM, D.; KIM, S.; SO, C. H.; KANG, J. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, Oxford University Press, v. 36, n. 4, p. 1234–1240, 2020. Doi: <https://doi.org/10.1093/bioinformatics/btz682>.

33 WANG, A.; SINGH, A.; MICHAEL, J.; HILL, F.; LEVY, O.; BOWMAN, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018. Available: <https://arxiv.org/abs/1804.07461>.

34 BUCILUă, C.; CARUANA, R.; NICULESCU-MIZIL, A. Model compression. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2006. p. 535–541.

35 HINTON, G.; VINYALS, O.; DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. Available: <https://arxiv.org/abs/1503.02531>.

36 ZHU, Y.; KIROS, R.; ZEMEL, R.; SALAKHUTDINOV, R.; URTASUN, R.; TORRALBA, A.; FIDLER, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 19–27.

37 YANG, Z.; DAI, Z.; YANG, Y.; CARBONELL, J.; SALAKHUTDINOV, R. R.; LE, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, v. 32, 2019. Available: <https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf>.

38 RAU, L. F. Extracting company names from text. In: IEEE COMPUTER SOCIETY. *Proceedings the Seventh IEEE Conference on Artificial Intelligence Application*. 1991. p. 29–30. Available: <https://doi.org/10.1109/CAIA.1991.120841>.

39 DING, N.; XU, G.; CHEN, Y.; WANG, X.; HAN, X.; XIE, P.; ZHENG, H.-T.; LIU, Z. Few-nerd: A few-shot named entity recognition dataset. *arXiv preprint arXiv:2105.07464*, 2021. Available: <https://arxiv.org/abs/2105.07464>.

40 BATISTA, D. S. *Large-Scale Semantic Relationship Extraction for Information Discovery*. Tese (Doutorado) — INSTITUTO SUPERIOR TÉCNICO, 2016.

41 ETZIONI, O.; BANKO, M.; SODERLAND, S.; WELD, D. S. Open information extraction from the web. *Communications of the ACM*, ACM New York, NY, USA, v. 51, n. 12, p. 68–74, 2008.

42 HAN, X.; GAO, T.; YAO, Y.; YE, D.; LIU, Z.; SUN, M. OpenNRE: An Open and Extensible Toolkit for Neural Relation Extraction. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*. [S.l.]: Association for Computational Linguistics, 2019. p. 169–174. Doi: <https://doi.org/10.18653/v1/D19-3029>.

43 RIEDEL, S.; YAO, L.; MCCALLUM, A. Modeling relations and their mentions without labeled text. In: SPRINGER. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. [S.l.], 2010. p. 148–163.

44 HOFFMANN, R.; ZHANG, C.; LING, X.; ZETTLEMOYER, L.; WELD, D. S. Knowledge-based weak supervision for information extraction of overlapping relations. In: *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*. [S.l.: s.n.], 2011. p. 541–550.

45 YAO, Y.; YE, D.; LI, P.; HAN, X.; LIN, Y.; LIU, Z.; LIU, Z.; HUANG, L.; ZHOU, J.; SUN, M. Docred: A large-scale document-level relation extraction dataset. *arXiv preprint arXiv:1906.06127*, 2019.

46 MUNKHDALAI, T.; YU, H. Meta networks. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2017. p. 2554–2563.

47 RAVI, S.; LAROCHELLE, H. Optimization as a model for few-shot learning. 2016.

48 FINN, C.; ABBEEL, P.; LEVINE, S. Model-agnostic meta-learning for fast adaptation of deep networks. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2017. p. 1126–1135.

49 SOARES, L. B.; FITZGERALD, N.; LING, J.; KWIATKOWSKI, T. Matching the blanks: Distributional similarity for relation learning. *arXiv preprint arXiv:1906.03158*, 2019.

50 SAINZ, O.; LACALLE, O. L. de; LABAKA, G.; BARRENA, A.; AGIRRE, E. Label verbalization and entailment for effective zero-and few-shot relation extraction. *arXiv preprint arXiv:2109.03659*, 2021. Available: <https://arxiv.org/abs/2109.03659>.

51 MACCARTNEY, B.; MANNING, C. D. Modeling semantic containment and exclusion in natural language inference. In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, 2008. p. 521–528. Available: <https://aclanthology.org/C08-1066>.

52 BOWMAN, S. R.; ANGELI, G.; POTTS, C.; MANNING, C. D. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015. Available: <https://arxiv.org/abs/1508.05326>.

53 WILLIAMS, A.; NANGIA, N.; BOWMAN, S. R. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017. Available: <https://arxiv.org/abs/1704.05426>.

54 REAL, L.; RODRIGUES, A.; SILVA, A. Vieira e; ALBIERO, B.; THALENBERG, B.; GUIDE, B.; SILVA, C.; LIMA, G. d. O.; CÂMARA, I.; STANOJEVIĆ, M. et al. Sick-br: a portuguese corpus for inference. In: SPRINGER. *International Conference on Computational Processing of the Portuguese Language*. Springer International Publishing, 2018. p. 303–312. Available: <https://doi.org/10.1007/978-3-319-99722-3_31>.

55 CONNEAU, A.; LAMPLE, G.; RINOTT, R.; WILLIAMS, A.; BOWMAN, S. R.; SCHWENK, H.; STOYANOV, V. Xnli: Evaluating cross-lingual sentence representations. *arXiv preprint arXiv:1809.05053*, 2018. Available: <https://arxiv.org/abs/1809.05053>.

56 CAVIGLIONE, L.; WENDZEL, S.; MAZURCZYK, W. The future of digital forensics: Challenges and the road ahead. *IEEE Security & Privacy*, IEEE, v. 15, n. 6, p. 12–17, 2017. Doi: <https://doi.org/10.1109/MSP.2017.4251117>.

57 UKWEN, D. O.; KARABATAK, M. Review of NLP-based Systems in Digital Forensics and Cybersecurity. In: IEEE. *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*. [S.l.], 2021. p. 1–9. Doi: <https://doi.org/10.1109/ISDFS52919.2021.9486354>.

58 BAAR, R. V.; BEEK, H. M. van; EIJK, E. V. Digital Forensics as a Service: A game changer. *Digital Investigation*, Elsevier, v. 11, p. S54–S62, 2014. Doi: <https://doi.org/10.1016/j.diin.2014.03.007>.

59 BEEK, H. M. van; EIJK, E. V.; BAAR, R. V.; UGEN, M.; BODDE, J.; SIEMELINK, A. Digital forensics as a service: Game on. *Digital Investigation*, Elsevier, v. 15, p. 20–38, 2015. Doi: <https://doi.org/10.1016/j.diin.2015.07.004>.

60 BEEK, H. M. van; BOS, J. van den; BOZTAS, A.; EIJK, E. van; SCHRAMP, R.; UGEN, M. Digital forensics as a service: Stepping up the game. *Forensic Science International: Digital Investigation*, Elsevier, v. 35, p. 301021, 2020. ISSN 2666-2817. Doi: <https://doi.org/10.1016/j.fsidi.2020.301021>.

61 SCHMITT, X.; KUBLER, S.; ROBERT, J.; PAPADAKIS, M.; LETRAON, Y. A replicable comparison study of NER software: StanfordNLP, NLTK, OpenNLP, SpaCy, Gate. In: IEEE. *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. [S.l.], 2019. p. 338–343. Doi: <https://doi.org/10.1109/SNAMS.2019.8931850>.

62 CABRERA-DIEGO, L. A.; MORENO, J. G.; DOUCET, A. Simple Ways to Improve NER in Every Language using Markup. In: *CLEOPATRA@ WWW*. [S.l.: s.n.], 2021. p. 17–31.

63 ARAUJO, P. H. L. de; CAMPOS, T. E. de; OLIVEIRA, R. R. de; STAUFFER, M.; COUTO, S.; BERMEJO, P. Lener-br: a dataset for named entity recognition in brazilian legal text. In: SPRINGER. *International Conference on Computational Processing of the Portuguese Language*. [S.l.], 2018. p. 313–323. Doi: <https://doi.org/10.1007/978-3-319-99722-3_32>.

64 SCHWETER, S.; AKBIK, A. Flert: Document-level features for named entity recognition. *arXiv preprint arXiv:2011.06993*, 2020. Available: <https://arxiv.org/abs/2011.06993>.

65 WANG, X.; JIANG, Y.; BACH, N.; WANG, T.; HUANG, Z.; HUANG, F.; TU, K. Automated concatenation of embeddings for structured prediction. *arXiv preprint arXiv:2010.05006*, 2020. Available: <https://arxiv.org/abs/2010.05006>.

66 SANG, E. F.; MEULDER, F. D. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003. Available: <https://arxiv.org/abs/cs/0306050>.

67 LEE, J.; SEO, S.; CHOI, Y. S. Semantic relation classification via bidirectional lstm networks with entity-aware attention using latent entity typing. *Symmetry*, Multidisciplinary Digital Publishing Institute, v. 11, n. 6, p. 785, 2019. Doi: <https://doi.org/10.3390/sym11060785>.

68 NAN, G.; GUO, Z.; SEKULIĆ, I.; LU, W. Reasoning with latent structure refinement for document-level relation extraction. *arXiv preprint arXiv:2005.06312*, 2020. Available: <https://arxiv.org/abs/2005.06312>.

69 SHI, P.; LIN, J. Simple bert models for relation extraction and semantic role labeling. *arXiv preprint arXiv:1904.05255*, 2019. Available: <https://arxiv.org/abs/1904.05255>.

70 HAN, X.; ZHU, H.; YU, P.; WANG, Z.; YAO, Y.; LIU, Z.; SUN, M. Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. *arXiv preprint arXiv:1810.10147*, 2018. Available: <https://arxiv.org/abs/1810.10147>.

71 GAO, T.; HAN, X.; ZHU, H.; LIU, Z.; LI, P.; SUN, M.; ZHOU, J. Fewrel 2.0: Towards more challenging few-shot relation classification. *arXiv preprint arXiv:1910.07124*, 2019. Available: <https://arxiv.org/abs/1910.07124>.

72 WANG, H.; FOCKE, C.; SYLVESTER, R.; MISHRA, N.; WANG, W. Fine-tune bert for docred with two-step process. *arXiv preprint arXiv:1909.11898*, 2019. Available: <https://arxiv.org/abs/1909.11898>.

73 ZHOU, W.; CHEN, M. An improved baseline for sentence-level relation extraction. *arXiv preprint arXiv:2102.01373*, 2021. Available: <https://arxiv.org/abs/2102.01373>.

74 LYU, S.; CHEN, H. Relation classification with entity type restriction. *arXiv preprint arXiv:2105.08393*, 2021. Available: <https://arxiv.org/abs/2105.08393>.

75 ZHANG, Y.; ZHONG, V.; CHEN, D.; ANGELI, G.; MANNING, C. D. Position-aware attention and supervised data improve slot filling. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. [S.l.: s.n.], 2017. p. 35–45.

76 MA, T.; PAN, Q.; RONG, H.; QIAN, Y.; TIAN, Y.; AL-NABHAN, N. T-BERTSum: Topic-Aware Text Summarization Based on BERT. *IEEE Transactions on Computational Social Systems*, IEEE, 2021. Doi: <https://doi.org/10.1109/TCSS.2021.3088506>.

77 SAHA, T.; JAYASHREE, S. R.; SAHA, S.; BHATTACHARYYA, P. BERT-caps: A transformer-based capsule network for tweet act classification. *IEEE Transactions on Computational Social Systems*, IEEE, v. 7, n. 5, p. 1168–1179, 2020. Doi: <https://doi.org/10.1109/TCSS.2020.3014128>.

78 POLIGNANO, M.; BASILE, P.; GEMMIS, M. D.; SEMERARO, G.; BASILE, V. Alberto: Italian BERT language understanding model for NLP challenging tasks based on tweets. In: CEUR. *6th Italian Conference on Computational Linguistics, CLiC-it 2019*. [S.l.], 2019. v. 2481, p. 1–6.

79 JOSHI, M.; LEVY, O.; WELD, D. S.; ZETTLEMOYER, L. BERT for coreference resolution: Baselines and analysis. *arXiv preprint arXiv:1908.09091*, 2019. Available: <https://arxiv.org/abs/1908.09091>.

80 KANTOR, B.; GLOBERSON, A. Coreference resolution with entity equalization. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. [S.l.: s.n.], 2019. p. 673–677.

81 ZHANG, Z.; HAN, X.; LIU, Z.; JIANG, X.; SUN, M.; LIU, Q. Ernie: Enhanced language representation with informative entities. *arXiv preprint arXiv:1905.07129*, 2019. Available: <https://arxiv.org/abs/1905.07129>.

82 NGUYEN, D. Q.; VU, T.; NGUYEN, A. T. Bertweet: A pre-trained language model for english tweets. *arXiv preprint arXiv:2005.10200*, 2020. Available: <https://arxiv.org/abs/2005.10200>.

83 GABBARD, R.; FREEDMAN, M.; WEISCHEDEL, R. Coreference for learning to extract relations: Yes virginia, coreference matters. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. [S.l.: s.n.], 2011. p. 288–293.

84 NEURALCOREF. 2021. <https://github.com/huggingface/neuralcoref>. Accessed: 2021-09-20.

85 PRADHAN, S.; MOSCHITTI, A.; XUE, N.; URYUPINA, O.; ZHANG, Y. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In: *Joint Conference on EMNLP and CoNLL-Shared Task*. [S.l.: s.n.], 2012. p. 1–40.

86 CLARK, K.; MANNING, C. D. Deep reinforcement learning for mention-ranking coreference models. *arXiv preprint arXiv:1609.08667*, 2016. Available: <https://arxiv.org/abs/1609.08667>.

87 ANTONITISCH, A.; FIGUEIRA, A.; AMARAL, D.; FONSECA, E. B.; ABREU, S. C. de; VIEIRA, R. Summ-it++: an enriched version of the summ-it corpus. *Proceedings of LREC 2016, 2016, Eslovênia.*, 2016.

88 VIEIRA, R.; MENDES, A.; QUARESMA, P.; FONSECA, E.; COLLOVINI, S.; ANTUNES, S. Corref-PT: A Semi-Automatic Annotated Portuguese Coreference Corpus. *Computación y Sistemas*, Centro de Investigación en computación, IPN, v. 22, n. 4, p. 1259–1267, 2018.

89 NER datasets for Portuguese. 2021. <https://github.com/davidsbatista/NER-datasets/blob/master/Portuguese/README.MD>. Accessed: 2021-09-20.

90 NER datasets for English. 2021. <https://github.com/juand-r/entity-recognition-datasets>. Accessed: 2021-09-20.

91 SANTOS, D.; SECO, N.; CARDOSO, N.; VILELA, R. Harem: An advanced ner evaluation contest for portuguese. In: *quot; In Nicoletta Calzolari; Khalid Choukri; Aldo Gangemi; Bente Maegaard; Joseph Mariani; Jan Odjik; Daniel Tapias (ed) Proceedings of the 5 th International Conference on Language Resources and Evaluation (LREC'2006)(Genoa Italy 22-28 May 2006)*. [S.l.: s.n.], 2006.

92 FREITAS, C.; CARVALHO, P.; OLIVEIRA, H. G.; MOTA, C.; SANTOS, D. Second harem: advancing the state of the art of named entity recognition in portuguese. In: EUROPEAN LANGUAGE RESOURCES ASSOCIATION. *quot; In Nicoletta Calzolari; Khalid Choukri; Bente Maegaard; Joseph Mariani; Jan Odijk; Stelios Piperidis; Mike Rosner; Daniel Tapias (ed) Proceedings of the International Conference on Language Resources and Evaluation (LREC 2010)(Valletta 17-23 May de 2010) European Language Resources Association*. [S.l.], 2010.

93 NOTHMAN, J.; RINGLAND, N.; RADFORD, W.; MURPHY, T.; CURRAN, J. R. Learning multilingual named entity recognition from wikipedia. *Artificial Intelligence*, Elsevier, v. 194, p. 151–175, 2013. Doi: <https://doi.org/10.1016/j.artint.2012.03.006>.

94 DERCZYNSKI, L.; NICHOLS, E.; ERP, M. van; LIMSOPATHAM, N. Results of the WNUT2017 shared task on novel and emerging entity recognition. In: *Proceedings of the 3rd Workshop on Noisy User-generated Text*. [S.l.: s.n.], 2017. p. 140–147.

95 RAMSHAW, L. A.; MARCUS, M. P. Text chunking using transformation-based learning. In: *Natural language processing using very large corpora*. [S.l.]: Springer, 1999. p. 157–176. Doi: <https://doi.org/10.1007/978-94-017-2390-9_10>.

96 BATISTA, D. S.; FORTE, D.; SILVA, R.; MARTINS, B.; SILVA, M. Exploring DBpedia and Wikipedia for Portuguese Semantic Relationship Extraction. *linguamatica*, v. 5, n. 1, p. 41–57, 2013.

97 DBPEDIA semantic relationship dataset for Portuguese. 2021. <https://github.com/davidsbatista/Annotated-Semantic-Relationships-Datasets/blob/master/datasets/DBpediaRelations-PT-0.2.txt.bz2>. Accessed: 2021-09-20.

98 OPENNRE Framework. 2021. <https://github.com/thunlp/OpenNRE>. Accessed: 2021-09-20.

99 GOOGLE Colab. 2021. <https://colab.research.google.com/>. Accessed: 2021-09-20.

100 FTK Imager. 2022. <https://www.exterro.com/ftk-imager>. Accessed: 2022-01-11.

101 ALLENNLP Natural Language Processing Platform. 2021. <https://allennlp.org/>. Accessed: 2021-09-20.

102 NEO4J Graph Database Platform. 2021. <https://neo4j.com/>. Accessed: 2021-09-20.

103 SPACY Training. 2021. <https://spacy.io/usage/training>. Accessed: 2021-09-20.

104 HYPER-PARAMETERS tuning script. 2021. <https://github.com/explosion/projects/tree/v3/integrations/wandb/scripts>. Accessed: 2021-09-20.

105 WANG, R.; TANG, D.; DUAN, N.; WEI, Z.; HUANG, X.; CAO, G.; JIANG, D.; ZHOU, M. et al. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*, 2020. Available: <https://arxiv.org/abs/2002.01808>.

106 ZHONG, Z.; CHEN, D. A frustratingly easy approach for joint entity and relation extraction. *arXiv e-prints*, 2020. Available: <https://arxiv.org/abs/2010.12812>.

107 PENG, H.; GAO, T.; HAN, X.; LIN, Y.; LI, P.; LIU, Z.; SUN, M.; ZHOU, J. Learning from context or names? an empirical study on neural relation extraction. *arXiv preprint arXiv:2010.01923*, 2020. Available: <https://arxiv.org/abs/2010.01923>.

108 MENEZES, D.; MILIDIU, R.; SAVARESE, P. Building a massive corpus for named entity recognition using free open data sources. In: IEEE. *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2019. p. 6–11. Doi: <https://doi.org/10.1109/BRACIS.2019.00011>.

109 NEO4J Python Driver. 2021. <https://github.com/neo4j/neo4j-python-driver>. Accessed: 2021-09-20.

APPENDIX

I. NAMED ENTITY RECOGNITION FINE-TUNING PROCESS

This appendix describes the main steps used for fine-tuning Named Entity Recognition models based on the SpaCy NLP library.

I.1 DATA PREPARATION

The first step in order to develop a good model is to prepare the data. This process usually involves data collection, cleansing and transformation. For NER models, the usual format for the data is the IOB representation in the form of a Tab Separated Values (TSV) file. Listing I.1 shows an example command to convert a TSV file with test data with the IOB representation into SpaCy's binary file, which are optimized for training and evaluation of different NLP models.

Listing I.1: NER convert command.

```
$ python -m spacy convert ./test.tsv . -t json -n 1 -c ner
```

Listing I.2 shows an example of how to inspect both the configuration file containing all the hyper-parameters for the model and the data itself through a Command Line Interface (CLI), which helps identifying missing parameters or problems with the data format before initializing the training process.

Listing I.2: NER debug command.

```
$ python -m spacy debug config ./config.cfg && \
  python -m spacy debug data ./config.cfg --verbose
```

I.2 TRAINING

For the training step, it is required to set different hyper-parameters for the model. Conveniently, these parameters can be set within a special configuration file that uses markups to identify components and their parameters. Listing I.3 presents the contents of a configuration file containing the hyper-parameters required to train a NER model based on the RoBERTa model.

Listing I.3: NER training configuration file contents.

```
[paths]
train = "splits/train.spacy"
```

```
dev = "splits/dev.spacy"
vectors = null
init_tok2vec = null

[system]
gpu_allocator = "pytorch"
seed = 0

[nlp]
lang = "en"
pipeline = ["transformer","ner"]
batch_size = 128
disabled = []
before_creation = null
after_creation = null
after_pipeline_creation = null
tokenizer = {"@tokenizers":"spacy.Tokenizer.v1"}

[components]

[components.ner]
factory = "ner"
moves = null
update_with_oracle_cut_size = 100

[components.ner.model]
@architectures = "spacy.TransitionBasedParser.v2"
state_type = "ner"
extra_state_tokens = false
hidden_width = 64
maxout_pieces = 2
use_upper = false
nO = null

[components.ner.model.tok2vec]
@architectures = "spacy-transformers.TransformerListener.v1"
grad_factor = 1.0
pooling = {"@layers":"reduce_mean.v1"}
upstream = "*"

[components.transformer]
factory = "transformer"
max_batch_items = 4096
set_extra_annotations =
{"@annotation_setters":"spacy-transformers.null_annotation_setter.v1"}

[components.transformer.model]
@architectures = "spacy-transformers.TransformerModel.v1"
name = "roberta-base"

[components.transformer.model.get_spans]
@span_getters = "spacy-transformers.strided_spans.v1"
```

```
window = 128
stride = 96

[components.transformer.model.tokenizer_config]
use_fast = true

[corpora]

[corpora.dev]
@readers = "spacy.Corpus.v1"
path = ${paths.dev}
max_length = 0
gold_preproc = false
limit = 0
augmenter = null

[corpora.train]
@readers = "spacy.Corpus.v1"
path = ${paths.train}
max_length = 500
gold_preproc = false
limit = 0
augmenter = null

[training]
accumulate_gradient = 3
dev_corpus = "corpora.dev"
train_corpus = "corpora.train"
seed = ${system.seed}
gpu_allocator = ${system.gpu_allocator}
dropout = 0.1
patience = 1600
max_epochs = 0
max_steps = 20000
eval_frequency = 200
frozen_components = []
before_to_disk = null

[training.batcher]
@batchers = "spacy.batch_by_padded.v1"
discard_oversize = true
size = 2000
buffer = 256
get_length = null

[training.logger]
@loggers = "spacy.ConsoleLogger.v1"
progress_bar = false

[training.optimizer]
@optimizers = "Adam.v1"
beta1 = 0.9
```

```
beta2 = 0.999
L2_is_weight_decay = true
L2 = 0.01
grad_clip = 1.0
use_averages = false
eps = 0.00000001

[training.optimizer.learn_rate]
@schedules = "warmup_linear.v1"
warmup_steps = 250
total_steps = 20000
initial_rate = 0.00005

[training.score_weights]
ents_per_type = null
ents_f = 1.0
ents_p = 0.0
ents_r = 0.0
```

Once all the hyper-parameters for the model are properly defined, it is possible to use the configuration file to start the training process. Listing I.4 shows an example command to start this process.

Listing I.4: NER training command.

```
$ python -m spacy train -g 0 ./config.cfg --output .
```

I.3 EVALUATION

After training, it is important to evaluate the model's performance and further fine-tune its hyper-parameters to improve results. Listing I.5 shows an example command to evaluate the model's performance given a file containing the test data, whereas Listing I.6 shows the evaluation output, including Precision, Recall and F-Score overall values as well as the same metrics for each entity type in the training data.

Listing I.5: NER evaluation command.

```
$ python -g 0 -m spacy evaluate ./model-best ./test.spacy \
--output ./model-best-evaluation.txt
```

Listing I.6: NER evaluation results.

```
{
  "token_acc":1.0,
  "ents_p":0.9189998239,
```

```
  "ents_r":0.9240439093,
  "ents_f":0.9215149642,
  "speed":4904.6220125346,
  "ents_per_type":{
    "LOC":{
      "p":0.93757503,
      "r":0.9364508393,
      "f":0.9370125975
    },
    "PER":{
      "p":0.967438948,
      "r":0.9554730983,
      "f":0.9614187928
    },
    "MISC":{
      "p":0.8061366806,
      "r":0.8233618234,
      "f":0.81465821
    },
    "ORG":{
      "p":0.9028840494,
      "r":0.9235400361,
      "f":0.9130952381
    }
  }
}
```


Listing I.7 shows an example of how to use a trained NER model and apply it to arbitrary texts to extract named entities. First of all, a method is defined to extract the entities for a given text (`get_entities()`) based on the entity classes used during training (`ENTITY_TYPES`). Then, this method can be applied to any new text or sentence, which will return a list of objects containing both the text and class for each detected entity in the input.

Listing I.7: NER model application example.

```python
import spacy

def get_entities(text, distinct=True):

  ENTITY_TYPES = ["PER", "ORG", "LOC", "MISC", "DATE"]

  entities = [{'label': ent.label_, 'title': ent.text} for ent in
              nlp(text, disable=["tagger", "parser"]).ents if ent.label_ in
              ENTITY_TYPES]

  if distinct:
    # Remove duplicates from entities
    entities = [dict(t) for t in {tuple(d.items()) for d in entities}]

```

```
    return entities

nlp = spacy.load("conll-roberta-base/model-best")

text = """Albert Einstein was a German scientist born in Germany in the
          19th century. He worked at NATO in World War II.
          Albert Einstein had a son named Daniel."""

entities = get_entities(text)
print('Named entities: ' + entities)

>> [{'label': 'PER', 'title': 'Albert Einstein'},
>> {'label': 'MISC', 'title': 'German'},
>> {'label': 'MISC', 'title': 'scientist'},
>> {'label': 'LOC', 'title': 'Germany'},
>> {'label': 'ORG', 'title': 'NATO'},
>> {'label': 'MISC', 'title': 'World War II'},
>> {'label': 'PER', 'title': 'Daniel'}]
```

II. RELATION EXTRACTION FINE-TUNING PROCESS

This appendix describes the main steps used for fine-tuning supervised Relation Extraction models based on PyTorch and OpenNRE libraries.

II.1 DATA PREPARATION

The data format required to train and evaluate a RE model is usually in the form of a JSON file. In particular, OpenNRE uses a JSON lines format, in which each line of a file is a JSON object containing properties about the data observation, including the tokenized sentence, the entity text and position for both subject and object and the relation class between them. Listing II.1 shows an example of how to use a transformer-based NER model developed with the SpaCy library (in this example, a RoBERTa model for English) to classify the entities for each sentence in the dataset. The generated output is in the same format as before, with the addition of two properties: `subj_type` and `obj_type`. The process of classifying the entities in the sentence is required for Typed Entity Marker (TEM) entity representation schemes, but it is not required for Entity Marker (EM) schemes, since EM schemes do not use entity classes.

Listing II.1: RE dataset entity classification example.

```
# Objective: classify subject and object type for entity relations
# Import dataset
inputFile = open("./dataset.txt", "r", encoding='UTF-8')
exportFile = open("./dataset.txt", "w", encoding='UTF-8')

spacy.require_gpu()

nlp = spacy.load('D:/spacy/conll-roberta-base/model-best')

inputFileLines = inputFile.readlines()

# loop through the the corpus
for line in inputFileLines:

    line_dict = json.loads(line)
    subj = line_dict["h"]["name"]
    obj = line_dict["t"]["name"]
    relation = line_dict["relation"]

    # Subject classification
    doc = nlp(subj)

    if len(doc.ents) > 0:
        subj_type = doc.ents[0].label_
    else:
        subj_type = "OTHER"
```

```
27
28      # Object classification
29      doc = nlp(obj)
30
31      if len(doc.ents) > 0:
32          obj_type = doc.ents[0].label_
33      else:
34          obj_type = "OTHER"
35
36      line_dict["subj_type"] = subj_type
37      line_dict["obj_type"] = obj_type
38
39      exportFile.write(json.dumps(line_dict, ensure_ascii=False)+'\n')
40
41  inputFile.close()
42  exportFile.close()
```

Listing II.2 shows an example Python script for splitting the data into train, validation (development) and test sets for each relation class. In this example, a file containing all relations of the class *parent* is imported and divided into the corresponding splits following a proportion of 60%, 20% and 20%, respectively. The same process should be applied to all other relation classes to concatenate them into the final respective files. There are several libraries and tools to achieve the same objective but this example's goal is to provide a simple intuition about this process.

Listing II.2: RE Dataset split example.

```
1  # Save the splits to new files
2  import random
3
4  inputFile = open("relation_parent.txt", "r", encoding='UTF-8')
5  exportFileTrain = open("train.txt", "w", encoding='UTF-8')
6  exportFileDev = open("dev.txt", "w", encoding='UTF-8')
7  exportFileTest = open("test.txt", "w", encoding='UTF-8')
8
9  inputFileLines = inputFile.readlines()
10 random.shuffle(inputFileLines)
11
12 print(f"Total corpus size: {len(inputFileLines)}")
13
14 train = inputFileLines[:int(len(inputFileLines)*0.6)]
15 dev = inputFileLines[int(len(inputFileLines)*0.6):int(len(inputFileLines)*0.8)]
16 test = inputFileLines[int(len(inputFileLines)*0.8):]
17
18 print(f"No. of train examples: {len(train)}")
19 print(f"No. of dev examples: {len(dev)}")
20 print(f"No. of test examples: {len(test)}")
21
22 for line in train:
23   exportFileTrain.write(line)
```

```
24
25 for line in dev:
26   exportFileDev.write(line)
27
28 for line in test:
29   exportFileTest.write(line)
30
31 inputFile.close()
32 exportFileTrain.close()
33 exportFileDev.close()
34 exportFileTest.close()
35
36 # Total corpus size: 1000
37 # No. of train examples: 600
38 # No. of dev examples: 200
39 # No. of test examples: 200
```

II.2 TRAINING

Once the training, validation and test data are properly formatted, the training process can be initiated. For this step, some hyper-parameters are required, which cannot be altered during training. Listing II.3 shows an example command to start the training process based on the OpenNRE framework, given some basic hyper-parameters for the model.

Listing II.3: Training command for Relation Extraction.

```
1  $ python OpenNRE/example/train_supervised_bert.py \
2  --pretrain_path neuralmind/bert-base-portuguese-cased \
3  --train_file train.txt \
4  --val_file dev.txt \
5  --rel2id_file dbpedia_rel2id.json \
6  --batch_size 64 \
7  --max_epoch 3 \
8  --lr 2e-5 \
9  --seed 6 \
10 --ckpt checkpoint
```

II.3 EVALUATION

In order the evaluate the model's performance and tune its hyper-parameters, it is recommended to have a separated test set. Listing II.4 shows an example command for this purpose, whereas Listing II.5 shows an example of an output of the evaluation process with values for Precision, Recall and F-Score.

Listing II.4: Evaluation command for Relation Extraction.

```
$ python OpenNRE/example/train_supervised_bert.py \
--pretrain_path neuralmind/bert-base-portuguese-cased \
--train_file train.txt \
--val_file dev.txt \
--test_file test.txt \
--rel2id_file dbpedia_rel2id.json \
--batch_size 64 \
--only_test
```

Listing II.5: Relation Extraction model evaluation results.

```
root - INFO - Test set results:
root - INFO - Accuracy: 0.9702880658436214
root - INFO - Micro precision: 0.9082819986310746
root - INFO - Micro recall: 0.8285119667013527
root - INFO - Micro F1: 0.866565084893339
```

In order to apply the trained RE model to arbitrary texts for practical applications, it is necessary to properly load the model's state dictionary with all the parameters obtained after training. Listing II.6 shows an example for this process using Python, PyTorch and the OpenNRE framework.

Listing II.6: Relation Extraction model initialization.

```
def get_model(model_name, root_path=default_root_path):
    ckpt = os.path.join(root_path, 'pretrain/nre/' + model_name + '.pth.tar')
    if model_name in ['wiki', 'tacred']:
        rel2id = json.load(open(os.path.join(root_path,
        '{}_rel2id.json'.format(model_name))))
        sentence_encoder = encoder.BERTEntityEncoder(
            max_length=80,
            pretrain_path=os.path.join(root_path, 'pretrain/bert-base-uncased'))
        m = model.SoftmaxNN(sentence_encoder, len(rel2id), rel2id)
        m.load_state_dict(torch.load(ckpt, map_location='cpu')['state_dict'])
        return m
    elif model_name in ['dbpedia']:
        rel2id = json.load(open(os.path.join(root_path, 'dbpedia_rel2id.json')))
        sentence_encoder = encoder.BERTEntityEncoder(
            max_length=80,
            pretrain_path=os.path.join(root_path,
            'pretrain/bert-base-portuguese-cased'))
        m = model.SoftmaxNN(sentence_encoder, len(rel2id), rel2id)
        m.load_state_dict(torch.load(ckpt, map_location='cpu')['state_dict'])
        return m
    else:
        raise NotImplementedError
```

After implementing the loading methods for the RE models, their application on arbitrary texts can be relatively simple. Listing II.7 shows the application of a RE model based on the Wiki schema for an example sentence (stored in the `text` variable). Prior to calling the model's predefined infer method (which outputs the best candidate relation for a given pair or entities in a sentence), it is required to format the input sentence into a JSON object, informing the positions for the head entity (`h`, also known as the subject) and the tail entity (`t`, also known as the object entity).

Listing II.7: Relation Extraction model application example.

```
import opennre

model = opennre.get_model('wiki')

text = 'Jack has a father, Derek, who is the primary suspect of this crime.'

ents = [('Jack', 'Derek')]

for ent in ents:
  head_start = text.find(ent[0])
  head_end = head_start + len(ent[0])
  tail_start = text.find(ent[1])
  tail_end = tail_start + len(ent[1])
  head = (head_start, head_end)
  tail = (tail_start, tail_end)
  print(model.infer({ 'text': text, 'h': { 'pos': head }, 't': { 'pos': tail } }))

>> (father, 0.988765)
```

III. INFORMATION EXTRACTION API SUMMARY

This appendix describes some of the development steps required to implement the Information Extraction Pipeline, including the creation of a Docker container to implement a platform-independent solution in the form of a Python-based API. Listing III.1 shows the contents of the `Dockerfile` for the proposed API.

Listing III.1: Dockerfile contents for the Information Extraction system.

```
FROM python:3.6-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
COPY models models
COPY images images
COPY OpenNRE OpenNRE
COPY gpr_pub gpr_pub
RUN pip3 install -r requirements.txt
RUN python3 -m spacy download en
RUN python3 -m spacy download pt
RUN cd OpenNRE && python3 setup.py install

COPY . .

CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0", "--port=3000"]
```

As showed in Listing III.1, a Python version 3.6 Docker image was used as the basis for the Docker container creation. The source code for this project in located in the `/app` directory and all the necessary code and models are imported prior to starting the API server.

Listing III.2 shows the Python packages and versions used to implement the IE API.

Listing III.2: Requirements file contents.

```
allennlp==2.1.0
allennlp-models==2.1.0
spacy==3.0.7
spacy-transformers==1.0.2
pytest
flask
ipython
pandas
torch==1.7.0
transformers
pytest==5.3.2
scikit-learn==0.22.1
```

```
scipy==1.4.1
nltk==3.4.5
neo4j
```

Listing III.3 shows a possible implementation of the relations schemas presented in this work, in the form of a Python dictionary. The idea behind this data structure is to check whether there exists a predefined relation for a given pair of entities, considering their labels.

Listing III.3: Relations schemas valid conditions.

```
dbpedia = {
    "PER-PER": ["parent", "partner", "influencedBy", "successor"],
    "PER-LOC": ["origin", "deathOrBurialPlace"],
    "PER-ORG": ["influencedBy", "partOf"],
    "LOC-PER": ["keyPerson"],
    "LOC-LOC": ["locatedInArea"],
    "ORG-PER": ["keyPerson"],
    "ORG-ORG": ["influencedBy", "partOf"],
    "ORG-LOC": ["locatedInArea"]
}

wiki = {
    "PER-PER": ["father", "spouse", "mother", "sibling", "child"],
    "PER-LOC": ["head of government", "work location", "country of origin",
                "country of citizenship", "residence"],
    "PER-ORG": ["member of political party", "member of"],
    "PER-MISC": ["participant", "position held", "field of work", "religion",
                 "instrument", "occupation", "participant of"],
    "LOC-ORG": ["member of"],
    "ORG-ORG": ["subsidiary", "owned by"],
    "MISC-LOC": ["country of origin"],
    "MISC-ORG": ["manufacturer", "developer"]
}

tacred = {
    "PER-PER": ["per:parents", "per:spouse", "per:other_family",
                "per:siblings", "per:children", "per:alternate_names"],
    "PER-LOC": ["per:countries_of_residence", "per:stateorprovince_of_birth",
                "per:cities_of_residence", "per:stateorprovinces_of_residence",
                "per:country_of_birth", "per:city_of_birth",
                "per:city_of_death", "per:country_of_death"],
    "PER-ORG": ["per:shcools_attended", "per:employee_of"],
    "PER-MISC": ["per:cause_of_death", "per:age", "per:religion", "per:title",
                 "per:charges"],
    "PER-DATE": ["per:date_of_birth", "per:date_of_death"],
    "ORG-ORG": ["org:alternate_names", "org:subsidiaries", "org:member_of"],
    "ORG-PER": ["org:shareholders", "org:members",
                "org:top_members/employees", "org:founded_by"],
    "ORG-MISC": ["org:number_of_employees/members",
                 "org:political/religious_affiliation", "org:website"],
```

```
    "ORG-LOC": ["org:city_of_headquarters", "org:country_of_headquarters",
                "org: stateorprovince_of_headquarters"],
    "ORG-DATE": ["org:founded", "org:dissolved"]
}
```

The API services available are described as follows:

- `GET /coref`: Apply coreference resolution to a given text;

  Parameters:

  - text (String): The input text

  Returns:

  - text_coref (String): Resolved text

- `GET /coref/vis`: Visualize coreference chains in a given text;

  Parameters:

  - text (String): The input text

  Returns:

  - text_coref (String): HTML file with markups for detected coreference chains

- `GET /entities`: Get entities in a given text;

  Parameters:

  - text (String): The input text
  - language (String): The input text language
  - ruler (List): Custom entities input

  Returns:

  - entities (List): Named entities in the text

- `GET /entities/vis`: Display entities markup in browser;

  Parameters:

  - text (String): The input text
  - language (String): The input text language
  - ruler (List): Custom entities input

Returns:

- entities (List): HTML file with markups for detected named entities in the text

- `GET /relations`: Get relation type for a single pair of entities within a given text;

  Parameters:

  - text (String): The input text
  - source (String): Token(s) corresponding to the source entity
  - target (String): Token(s) corresponding to the target entity
  - schema (String): The relations schema to use ("dbpedia", "wiki" or "tacred")

  Returns:

  - relations (Tuple): The predicted semantic relation between the pair of entities

- `GET /relations/all`: Get all entities and relations from text;

  Parameters:

  - text (String): The input text
  - language (String): The input text language ("en" or "pt")
  - coref (Bool): Whether to apply coreference resolution or not
  - schema (String): The relations schema to use ("dbpedia", "wiki" or "tacred")
  - relation_threshold (Float): Only extract relations with a score greater than this (default: 0.5)

  Returns:

  - ents_rel (Object): Named entities and semantic relations in the text

- `POST /graph`: Creates a Neo4j graph.

  Parameters:

  - data (Object): Input data with entities and relations
  - labels (Object): The named entities labels to consider for import
  - database (String): The Neo4j database name

  Returns:

  - message (String): Success or error message

Listing III.4 shows an example of how the IE API can be accessed from the client side, through a Python script, using HTTP (HyperText Transfer Protocol) or HTTPS (HyperText Transfer Protocol Secure) and a known IP (Internet Protocol) address and port number.

Listing III.4: Information Extraction API request example.

```
import json
import urllib
import urllib.parse
import urllib.request

def ie_pipeline(text):
    # Prepare the URL.
    data = urllib.parse.urlencode([
        ("text", text)])

    url = "http://localhost:5000?" + data
    print(url)
    req = urllib.request.Request(url, data=data.encode("utf8"), method="GET")
    print(req)
    with urllib.request.urlopen(req, timeout=150) as f:
        response = f.read()
        response = json.loads(response.decode("utf8"))
    # Output the annotations.
    return response

text = 'Elon Must is the CEO of Tesla. Musk is also the CEO of SpaceX.'
print(ie_pipeline(text))
```

IV. GRAPH VISUALIZATION SETUP

This appendix describes the steps required to import a JSON file containing named entities and their relations into a Neo4j database for dynamic graph visualization.

The JSON file structure used in this work followed a simple structure with two lists, one for entities and another for relations. Each entity in the entities list is composed of two key-value pairs:

- `label`: The named entity class label (PER, LOC or DATE, for example);
- `title`: The word (token) or group of words (tokens) corresponding to the entity label.

Each relation in the relations list is composed, in turn, of four key-value pairs:

- `source`: The named entity token (or tokens) that acts as the source of a relation (subject);
- `target`: The named entity token (or tokens) that acts as the target of a relation (object);
- `type`: The predefined relation type from a relation schema;
- `confidence`: A decimal value ranging from 0 to 1 that represents the model's confidence on the relation classification task.

Listing IV.1 shows the IE pipeline JSON output for Scenario's 2 example using Wiki's relation schema. Listings IV.2 and IV.3 show the commands used to import both the entities and relations (from the JSON output) into a graph database, respectively. Listing IV.4 shows an alternative approach to this same process, which automates the generation of nodes and links directly in a Python code (Neo4j Python driver [109]).

Listing IV.1: JSON file format example.

```
{
    "entities": [
        {"label": "DATE", "title": "1968"},
        {"label": "LOC", "title": "London"},
        {"label": "PER", "title": "Enrique Rosalio"},
        {"label": "MISC", "title": "Latin"},
        {"label": "MISC", "title": "Virologist"},
        {"label": "MISC", "title": "artist"},
        {"label": "VIRUS", "title": "Alphacoronavirus"},
        {"label": "DATE", "title": "1960s"},
        {"label": "PER", "title": "June Almeida"},
        {"label": "MISC", "title": "venezuelan"},
        {"label": "VIRUS", "title": "Gammacoronavirus"},
        {"label": "DATE", "title": "2007"},
        {"label": "LOC", "title": "United Kingdom"},
        {"label": "DATE", "title": "1913"},
        {"label": "DATE", "title": "1961"},
        {"label": "DATE", "title": "2020"},
```

```
        {"label": "ORG", "title": "International Committee on Taxonomy of
        Viruses"},
        {"label": "DATE", "title": "2009"},
        {"label": "VIRUS", "title": "Coronavirus"},
        {"label": "LOC", "title": "Scotland"},
        {"label": "LOC", "title": "United States"},
        {"label": "PER", "title": "Almeida"},
        {"label": "ORG", "title": "International Committee for the Nomenclature
        of Viruses"},
        {"label": "PER", "title": "Joyce"},
        {"label": "LOC", "title": "Bexhill"},
        {"label": "PER", "title": "E.C. Kendall"},
        {"label": "DATE", "title": "1930"},
        {"label": "ORG", "title": "Common Cold Unit"},
        {"label": "ORG", "title": "Nature"},
        {"label": "DATE", "title": "1993"},
        {"label": "VIRUS", "title": "Deltacoronavirus"},
        {"label": "ORG", "title": "British Medical Research Council"},
        {"label": "PER", "title": "Malcolm Bynoe"},
        {"label": "ORG", "title": "St. Thomas Hospital"},
        {"label": "PER", "title": "Tyrrell"},
        {"label": "PER", "title": "David Tyrrell"},
        {"label": "VIRUS", "title": "Betacoronavirus"},
        {"label": "DATE", "title": "1971"}
    ],
    "relations": [
        {
            "source": "E.C. Kendall",
            "target": "Common Cold Unit",
            "type": "member of",
            "confidence": 0.40488743782043457
        },
        {
            "source": "Malcolm Bynoe",
            "target": "Common Cold Unit",
            "type": "member of",
            "confidence": 0.3821450471878052
        },
        {
            "source": "David Tyrrell",
            "target": "Common Cold Unit",
            "type": "member of",
            "confidence": 0.8484625220298767
        },
        {
            "source": "British Medical Research Council",
            "target": "Common Cold Unit",
            "type": "subsidiary",
            "confidence": 0.9829787611961365
        },
        {
            "source": "June Almeida",
```

```
            "target": "Scotland",
            "type": "country of citizenship",
            "confidence": 0.7918770909309387
        },
        {
            "source": "June Almeida",
            "target": "St. Thomas Hospital",
            "type": "member of",
            "confidence": 0.49163421988487244
        },
        {
            "source": "June Almeida",
            "target": "London",
            "type": "work location",
            "confidence": 0.9257238507270813
        },
        {
            "source": "St. Thomas Hospital",
            "target": "London",
            "type": "headquarters location",
            "confidence": 0.6615065932273865
        },
        {
            "source": "June Almeida",
            "target": "Virologist",
            "type": "field of work",
            "confidence": 0.9864470958709717
        },
        {
            "source": "June Almeida",
            "target": "Bexhill",
            "type": "residence",
            "confidence": 0.9562858939170837
        },
        {
            "source": "June Almeida",
            "target": "Enrique Rosalio",
            "type": "spouse",
            "confidence": 0.9892877340316772
        },
        {
            "source": "Enrique Rosalio",
            "target": "artist",
            "type": "occupation",
            "confidence": 0.8995144367218018
        },
        {
            "source": "Enrique Rosalio",
            "target": "Joyce",
            "type": "child",
            "confidence": 0.9886932969093323
        },
```

```
        {
            "source": "June Almeida",
            "target": "Joyce",
            "type": "child",
            "confidence": 0.9867495894432068
        },
        {
            "source": "Joyce",
            "target": "June Almeida",
            "type": "mother",
            "confidence": 0.9927724003791809
        },
        {
            "source": "Joyce",
            "target": "Enrique Rosalio",
            "type": "father",
            "confidence": 0.9671555161476135
        }
    ]
}
```


Listing IV.2: Command to import entities as nodes into Neo4j from a JSON file.

```
CALL apoc.load.json("file:///data.json")
YIELD value
WITH value AS data
UNWIND data.entities AS entity
WITH data, entity
CALL apoc.create.node([entity.label], {name: entity.title}) YIELD node
RETURN distinct 'done'
```

Listing IV.3: Command to import relations as links into Neo4j from a JSON file.

```
CALL apoc.load.json("file:///data_tacred.json")
YIELD value
WITH value AS data
UNWIND data.relations AS relation
WITH relation
CALL apoc.search.node({PER: "name", LOC: "name", ORG: "name", MISC: "name",
DATE: "name"}, "EXACT", relation.source) YIELD node AS sourceNode
WITH sourceNode, relation
CALL apoc.search.node({PER: "name", LOC: "name", ORG: "name", MISC: "name",
DATE: "name"}, "EXACT", relation.target) YIELD node AS targetNode
WITH sourceNode, targetNode, relation
CALL apoc.create.relationship(sourceNode, relation.type,
{confidence: relation.confidence}, targetNode)
YIELD rel
```

```
15 RETURN distinct 'done'
```

Listing IV.4: Automatic import of nodes and links into Neo4j with Python driver.

```
1  from neo4j import GraphDatabase
2
3  driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "password"))
4  database = "remove"
5
6  def add_nodes(tx, data):
7      tx.run( "WITH $data as data "
8              "UNWIND data.entities as entity "
9              "WITH entity "
10             "CALL apoc.create.node([entity.label], {name: entity.title}) "
11             "YIELD node "
12             "RETURN distinct 'done'", data=data
13         )
14
15 def add_relations(tx, data, labels):
16     tx.run( "WITH $data as data "
17             "UNWIND data.relations as relation "
18             "WITH relation "
19             "CALL apoc.search.node(labels, 'EXACT', relation.source) "
20             "YIELD node as sourceNode "
21             "WITH sourceNode, relation "
22             "CALL apoc.search.node(labels, 'EXACT', relation.target) "
23             "YIELD node as targetNode "
24             "WITH sourceNode, targetNode, relation "
25             "CALL apoc.create.relationship(sourceNode, relation.type, "
26             "{confidence: relation.confidence}, targetNode) "
27             "YIELD rel "
28             "RETURN distinct 'done' ", data=data, labels=labels
29         )
30
31 with driver.session(database=database) as session:
32     session.write_transaction(add_nodes, data)
33     session.write_transaction(add_relations, data, labels)
34
35 driver.close()
```

Listing IV.5 shows eight examples of filtering queries used in Neo4j to retrieve nodes and relationships. These examples cover some of the basic operations that can be performed once the data containing nodes and relations have been correctly imported into the graph database. In line 1 (Listing IV.5), the query is used to return all nodes and all relations in the database. In line 2, only nodes of type `PER` and their corresponding relations are returned. Line 3 shows an example query that returns all nodes and relations for a specific node of type `PER` and property `name` equals to the string "Daniel". In line 4, a similar query is created, but it returns only the nodes that have a relation whose label is `parent` for the node of type

PER and name “Daniel”. The query from line 5 returns only nodes with at least only relation to another node, whereas in line 6 it is presented a query that only returns nodes that have no relation at all. In line 7, only nodes with a specific relation are returned (in this example, only nodes with a relation of type locatedInArea). Finally, in line 8, the query returns only nodes with a confidence score (obtained from the neural RE model) greater than 50%.

Listing IV.5: Neo4j’s filtering queries examples.

```
1 MATCH (n) RETURN n;
2 MATCH (n:PER) return n;
3 MATCH (n:PER)-[r]-(p) WHERE n.name='Daniel' RETURN n,r,p;
4 MATCH (n:PER)-[r:parent]->(p:PER) WHERE n.name='Daniel' RETURN n,r,p;
5 MATCH (n) WHERE (n)--() RETURN n;
6 MATCH (n) WHERE NOT (n)--() RETURN n;
7 MATCH (n)-[r:locatedInArea]-(p) RETURN n,r,p;
8 MATCH (n)-[r]-(p) WHERE r.confidence > 0.5 RETURN n,r,p;
```