University of Padova

Department of Information Engineering

Master Thesis in Computer Engineering

Multi-Span Extractive Question Answering for Named Entity Recognition

Master Candidate
Andrew Zamai

Student ID
2038522

Supervisor
Prof. Giorgio Satta
(University of Padova)

Co-supervisors
PhD Andrea Zugarini
PhD Leonardo Rigutini
(Expert.ai)

Academic Year 2022-2023
24th October 2023

Life happens when you are not making plans

Abstract

Named Entity Recognition (NER) is a Natural Language Processing (NLP) task that involves detecting and categorizing named entities in a text. Named entities can be names of people, organizations, locations and dates or can be specifically defined for the domain in which the NER task is adopted. NER proves helpful in a variety of applications, the most notable of which is Information Extraction, where NER allows the extraction of structured information from unstructured text. Plenty of approaches have been exploited, ranging from rule-based methods to machine-learning algorithms such as Conditional Random Fields and Hidden Markov Models, to deep-learning systems based on Recurrent Neural Networks and Transformer-based architectures.

Lately, with the advent of Large Language Models (LLMs) such as GPT and in-context learning techniques, a new approach to NER has emerged: extracting named entities by posing questions and having the LLM fill in the answer with the named entities (one or more) to extract. These LLMs, employed as Generative Question Answering (GQA) models, still require careful prompt-engineering, both for dealing with inputs not fitting into the context-window, and for producing well-formatted output. If a traditional NER system detects several text spans associated with a named entity, the generative QA model should be prompted to extract the same number of text spans. Furthermore, if a named entity does not appear, the generative LLM used as QA should not produce non-existent content.

Based on the above considerations, NER approached as a QA task might be a viable option, although an Extractive QA method may be more appropriate over a generative one. This thesis delves into the Extractive QA approach to NER, providing a comprehensive exploration of its core concepts and methodologies. In particular, this work will first present the operating principle of single-span EQA models, which can only extract a single span of text for each question, before moving on to the design and development of a new transformer-based model that enables Multi-Span Extractive QA, implemented according to specific guidelines so that it is a natural extension of the single-span operating principle. The proposed model is then evaluated on a NER dataset named BUSTER, where specific named entities from business transaction documents have to be extracted. Experiments and comparisons with other transformer-based NER systems, which constitute the baselines for the classical NER approach, allow to show the equal power of this Multi-span EQA approach to NER. Further investigative experiments reveal that the QA approach to NER through the proposed model implementation achieves better results in reduced training set scenarios. Future work will focus on trying to develop a technique to make the model perform Continual Learning on a sequence of NER datasets while retaining its capability to correctly respond to questions from all previously encountered datasets.

Sommario

Named Entity Recognition (NER) è un task relativo all'elaborazione del linguaggio naturale (NLP) che prevede il rilevamento e la categorizzazione di entità (Named Entities) in un testo. Le entità possono essere nomi di persone, organizzazioni, luoghi e date, oppure possono essere definite in modo specifico per il dominio in cui viene svolta l'attività di NER. NER si rivela utile in diverse applicazioni, la più importante delle quali è l'Information Extraction (IE), la quale consente di estrarre informazioni strutturate da testi non strutturati. Nel corso degli ultimi anni sono stati sfruttati numerosi approcci, che vanno dai metodi basati su regole, agli algoritmi di Machine Learning come i Conditional Random Fields e gli Hidden Markov Models, fino ai sistemi più moderni di Deep Learning basati su Recurrent Neural Networks e Transformers.

Recentemente, con l'avvento di modelli linguistici di grandi dimensioni (Large Language Models) come GPT e di tecniche di apprendimento innovative come "In-Context Learning", è emerso un nuovo approccio al NER: estrarre le entità ponendo domande e facendo in modo che il LLM compili la risposta con le entità (una o più) da estrarre. Questi LLM, impiegati come modelli generativi, richiedono ancora un'attenta ingegnerizzazione dei prompt, sia per gestire gli input che non rientrano nella finestra di contesto, sia per produrre output ben formattati. Se un sistema NER tradizionale rileva diversi span di testo associati ad una categoria, il modello generativo dovrebbe essere indotto a estrarre lo stesso numero di span di testo. Inoltre, se un'entità denominata non compare, il LLM generativo non deve produrre contenuti inesistenti.

Sulla base di queste considerazioni, l'approccio al NER tramite domande potrebbe essere un'opzione valida, anche se un metodo estrattivo potrebbe essere più appropriato di uno generativo. Questa tesi approfondisce l'approccio di question-answering estrattivo al task di NER, fornendo un'esplorazione completa dei suoi concetti e metodologie fondamentali. In particolare, questo lavoro presenterà innanzitutto il principio di funzionamento dei modelli Single-Span Extractive QA, che possono estrarre solo un singolo intervallo (span) di testo per ogni domanda, prima di passare alla progettazione e allo sviluppo di un nuovo modello basato su Transformers che consente l'estrazione di molteplici intervalli di testo come risposta ad una singola domanda, implementato secondo linee guida specifiche in modo da essere una naturale estensione del principio di funzionamento del precedente modello Single-Span. Il modello proposto viene quindi valutato su un dataset per il NER denominato BUSTER, in cui devono essere estratte specifiche entità da documenti riguardanti transazioni commerciali. Gli esperimenti e i confronti con altri sistemi NER basati su Transformers, i quali costituiscono la baseline per l'approccio NER classico, consentono di dimostrare la pari potenza di questo approccio Multi-span EQA al task di NER. Ulteriori esperimenti evidenziano come questo approccio, attraverso l'implementazione del modello proposto, ottenga risultati migliori in scenari in cui il set di dati di addestramento è ridotto.

Contents

Abstract v

Sommario v

List of figures xi

List of tables xiii

List of acronyms xv

1 Introduction 1
- 1.1 Framework . . . 1
- 1.2 Approach motivations . . . 2
- 1.3 Contributions and structure of this thesis . . . 4

2 Theoretical Background 7
- 2.1 Named Entity Recognition . . . 7
  - 2.1.1 Task definition . . . 7
  - 2.1.2 Approaches to NER . . . 9
- 2.2 Language Modeling . . . 15
  - 2.2.1 Task definition . . . 15
  - 2.2.2 Neural Language Models . . . 16
- 2.3 Extractive Question Answering . . . 21
  - 2.3.1 Single-span Extractive QA . . . 22
  - 2.3.2 Multi-span Extractive QA . . . 26
- 2.4 QA for NER . . . 31
  - 2.4.1 In-context learning using LLMs . . . 31
  - 2.4.2 Generative QA for NER . . . 32
  - 2.4.3 Multi-span Extractive QA for NER . . . 34

3 A Multi-Span Extractive QA Model 37
- 3.1 Design guidelines . . . 37
- 3.2 Model architecture . . . 38
  - 3.2.1 Multi-span modeling and training loss . . . 39
  - 3.2.2 Multi-span extraction algorithm . . . 41
  - 3.2.3 Long-documents handling . . . 45

4 Experiments 47
4.1 Datasets . . . 48
4.1.1 SQuAD2.0 . . . 48
4.1.2 BUSTER . . . 49
4.1.3 BUSTER for Question Answering . . . 52
4.2 Experimental Setup . . . 53
4.2.1 Metrics . . . 53
4.2.2 Models . . . 56
4.2.3 Training details . . . 57
4.3 Results . . . 58
4.4 Exploratory investigations . . . 61
4.4.1 Learning efficiency on reduced training data . . . 61
4.4.2 Forgetting . . . 63

5 Conclusions and Future Work 65

References 67

Acknowledgments 69

List of figures

2.1 NER on one sentence example. NER can be seen as the task of highlighting with different colors one or multiple words in a text, where each color corresponds to a different Named Entity category. 8

2.2 BIO labeling for NER task on Figure 2.1 sentence example. 10

2.3 Transformer architecture illustration from [1]. 17

2.4 BERT architecture illustration from [2]. 20

2.5 Single-span Extractive QA architecture (based on Neural LM) illustration from [3]. 23

2.6 MTMSN architecture illustration from [4]. 27

2.7 MTMSN multi-span extraction mechanism illustration from [4]. 29

2.8 A multi-span question from DROP labeled using BIO scheme, illustration from [5]. 30

3.1 An example of k-hot vectors modeling of the start & end positions for two answer spans over the input tokens of a simple question about a short text passage. The input is constructed right-concatenating the text passage to the question, i.e. [q;x]. The special tokens added by the RoBERTa tokenizer are simulated, but a simplified word-level tokenization is used. 40

3.2 The Binary Cross Entropy with Logits loss independently models the probability of each token to be an answer span start (likewise, to be an end position) by pushing the logit value out_i at a starting position to be greater than 0, while pushing the logit values for the tokens that are not a starting position to be less than 0. 41

3.3 A simplified [<s>; *4-only-passage*] input is considered, to not complicate the example modeling also other special delimiter </s> and relative to the question tokens. The model start logits & end logits over such input are reported and the outer sum matrix *SE* is computed. Some constrains over this matrix are then imposed: the fuchsia cells are discarded by the "end must not precede start" constraint, the orange ones by the "max answer length" constraint, and the red ones by "implementation constraints". Finally, the spans in the green slots which value is > 0 are the extracted answer spans to the question. 44

4.1 Two instances from the SQuAD2.0 development set on the same context: the first question is answerable while the second one is not. Although the dataset is oriented towards Single-Span QA, an instance from the development (or test set) may present multiple spans as gold-answers, consisting of small modifications of approximately the same text span by including or excluding one or more words before its beginning or after its end. A gold answer includes the *text span* from the context and its *starting position in characters* from the beginning of it; the *ending position* can be easily computed by adding the length of the text span to the starting position. 49
4.2 Passage of text extracted from BUSTER where only "linguistically evident from the sentence in which they occur" Named Entities are tagged. This can be seen in the second paragraph where, although it is known from the previous sentences that "Security Holding Corp." and "Vuance Ltd." are the ACQUIRED_COMPANY and BUYING_COMPANY respectively, they are not tagged because it is not clear from the sentence in which they occur what their role is. 50
4.3 Macro-F1 scores of the four baseline models and the Multi-Span Extractive QA model on each BUSTER NE category. 59
4.4 $F1$ comparison between the Multi-Span Extractive QA based approach and the best performing NER baseline (RoBERTa) when fine-tuned on reduced training sets of varying number of occurrences per NE category. 62
4.5 SQuAD2.0 and BUSTER score trajectories during fine-tuning of the MS-EQA model on the default BUSTER permutation. 63

List of tables

4.1 BUSTER's tag-set of Named Entities and their associated definitions. The NEs are grouped in three families of tags: (a) *Parties* which groups tags used to identify the entities directly involved in the transaction; (b) *Advisors* which groups tags identifying any external facilitator and advisor of the transaction; (c) *Generic_Info* which identifies tags reporting any information about the transaction. 50
4.2 The statistics about the 5 Gold folds (and Silver) data. 51
4.3 The five permutations of folds used for 5-fold cross-validation. 51
4.4 The question associated with each NE tag. 52
4.5 Micro (μ-) and macro (M-) scores of the four baseline models and the Multi-Span Extractive QA model on BUSTER; averages and standard deviations across permutations evaluated using 5-Fold Cross Validation. 59
4.6 Tag-wise precision, recall and F1-score values obtained with the Multi-Span Extractive QA model, using 5-Fold Cross Validation. 60

List of acronyms

NLP . . . . . Natural Language Processing

NER . . . . . Named Entity Recognition

NE . . . . . Named Entity (plural NEs)

QA . . . . . Question Answering

EQA . . . . . Extractive Question Answering

SS-EQA . . . . . Single-span Extractive Question Answering

MS-EQA . . . . . Multi-span Extractive Question Answering

LM . . . . . Language Modeling

NLM . . . . . Neural Language Model

LLM . . . . . Large Language Model

RNN . . . . . Recurrent Neural Network

MLP . . . . . Multi Layer Perceptron

SPP . . . . . Structured Prediction Problem

TP . . . . . True Positive (plural TPs)

FP . . . . . False Positive (plural FPs)

FN . . . . . False Negative (plural FNs)

TN . . . . . True Negative (plural TNs)

1 Introduction

1.1 Framework

Language represents the primary means by which humans communicate. Daily, people use it to express their needs, thoughts and emotions. While the origins of language are challenging to trace definitively, in its early stages it was rudimentary and simple compared to contemporary forms. Similar to non-human species, language initially served the purpose of expressing basic natural needs, communicating survival-related information and facilitating cooperation. Then, over the course of millennia, as human societies became more complex and interactions between groups increased, languages evolved and diversified to meet a broader range of communicative purposes, resulting in the development of the more structured and sophisticated languages of the present day. Language has evolved over time to reflect emerging human needs and, reciprocally, human progress has built upon language. Language has allowed to pass down knowledge, stories and cultural practices from one generation to the next, it has been fundamental for the establishment of legal systems and governance structures, it has allowed to disseminate new knowledge and build on existing research for scientific and technological progress, and it has been crucial for conducting business, trade and economic activities.

First, the invention of the printing press, followed by the widespread increase in literacy rates and, more recently, the advent of computers, the internet, and social media, have collectively generated an immense volume of linguistic data. This data explosion has surpassed humans' ability to efficiently process and make use of it. Driven by the imperative to manage and extract value from this wealth

of textual information, the field of Natural Language Processing (NLP) has arisen. NLP enables machines to read, comprehend and derive meaning from human language, as well as to generate it. NLP plays an important role in various domains, such as economics and medicine, by enabling, for example, the rapid summarization of lengthy documents to facilitate informed decision-making. NLP also makes possible instantaneous language translation, bridging linguistic gaps between foreign people, allows for the fake-news detection, for sentiment classification about reviews, for assisting customers through chat-bots, and a myriad of other applications.

In this age of abundant linguistic data, Named Entity Recognition (NER) has surfaced as a powerful Natural Language Processing tool facilitating the extraction of structured information from unstructured text data. NER involves detecting and categorizing named entities in a text, which can be names of people, organizations, locations and dates, or can be specifically defined for the domain in which the NER task is adopted. One could think that the NER task could be easily solved by having huge lists of, for example, names of people, places and organizations, but this, in practice, would not work. Language is indeed tricky. It is a complex system of communication, with each language having its own unique vocabulary and grammar. Textual data is constructed by arranging discrete words from a vocabulary according to a specific syntax, enabling the formation of potentially infinite-length sentences through recursively combining phrases. Words in language are symbolic, meaning they can refer to something concrete about the world or to an abstract concept, but they can also be ambiguous. For instance, the word "Apple" could refer to the fruit or the well-known tech company, depending on the context. Moreover, while the vocabulary is finite, new words can always be coined to meet emerging communication needs. Consequently, Named Entity Recognition cannot be simply approached by maintaining extensive lists of named entities; it requires a deep understanding of the text and context. To comprehend text, one must analyze its hidden structure and employ compositionality: this means understanding the meanings of the sub-components and how they combine to convey the overall meaning of it.

1.2 Approach motivations

Many approaches to NER have been explored over the last decades. First, the early transition from lists and grammar rules based systems to feature-based systems involved the adoption of Machine Learning techniques, particularly Hidden Markov Models, Conditional Random Fields and Support Vector Machines. Then, the NLP landscape saw a remarkable transformation with the advent of Deep Learning-based end-to-end systems. These systems introduced the use of Neural Language Models for representation learning. They evolved from basic multi-layer perceptrons to more sophisticated mod-

els, including Recurrent Neural Networks, and most notably, the Transformers architecture. These deep learning models have ushered in a new era of improved NER accuracy and performance.

Notably, all the above approaches rely on treating NER as a "Sequence Labeling Task" where the goal is, given in input a sequence of elements (e.g., words), assign a NE category to each element. In contrast to this approach and to NER, which primarily focuses on text comprehension, in recent years the transformer-decoder architecture has found significant utility as Neural Language Model for generative tasks. In particular, when these models are powered by architectures with a huge number of parameters (e.g., GPT-3 around 175 billion), a new emerging ability has been discovered, that is "in-context learning". This can be simply seen as the technique of letting these Large Language Models to complete text that is coherent with what precedes, namely doing auto-regressive language modeling. By properly formulating the text on which the LLM conditions the generation (called the "prompt"), one can solve various NLP tasks. The model leverages its pre-trained language understanding and generation capabilities to adapt to the provided context and generate coherent and contextually relevant text, being able to solve even properly cast NLP tasks without having to be fine-tuned on training data, just requiring some examples in the prompt.

Among the NLP tasks that can be approached through In-Context Learning on Large Language Models is Named Entity Recognition. Several papers in the literature have attempted to rewrite NER as Language Modeling task. One of the possible techniques is to formulate a test sentence for each word in the passage with a template like "[word] is a ___", provide some examples in the prompt and then let the LLM generate the appropriate NE category for that word. The cons of this approach are evident: if n is the number of words in the passage, it will require to formulate n prompts. A better approach could then be to formulate the prompt as a question. Suppose one has m NE categories to extract, by defining a question for each category (e.g., "Which are the words in the passage that refer to a person?"), one should be able in m prompts to extract all the searched NE categories in the text. The number of NE categories m is always much smaller than the passage length n. If this approach has computational advantages over the above, what are instead its disadvantages and limitations?

Hallucinations are a well known problem of these generative LLMs and refers to their tendency to generate responses that sound coherent but may contain inaccurate or fictional information. A possible implication could then be that the LLM, in response to the question "What are the words in the passage that refer to a person?", generates names of people that may be plausible but are not present in the text passage in the input. Another limitation of this approach regards output-formatting: asking to return the words belonging to a particular NE category may not be sufficient. Especially for NER categories on domains where some NEs concern a semantic action (e.g. "BUYER"), we may want to know not only the names, but also their occurrences (where they appear) in the passage. This

is because there is often the requirement that the NEs to be labeled must be linguistically evident from the sentence in which they occur. In addition to the above problems, LLMs like GPT-3 are computationally intensive models that require powerful hardware to train and run effectively. Despite "in-context learning" allows to use these models without usually the need of fine-tuning them, these models are generally not employed "on-premises", but through APIs. Many API-based LLM services charge based on the number of tokens processed, which can become expensive for extensive or frequent usage. Lastly, privacy is a significant concern when using LLMs. When users input text into these models, the data is often processed on external servers or cloud infrastructure. This raises potential privacy issues, as sensitive or personal information could be exposed during processing; in industrial applications this is a significant concern.

1.3 Contributions and structure of this thesis

Inspired by the possibility of approaching the NER task not through the classical sequence-labelling method, but by posing questions to extract all occurrences from a document relative to a Named Entity category, and knowing that a generative approach through LLM is not suitable due to the above-mentioned disadvantages and limitations, this thesis proposes an "Extractive QA approach to the NER task". In particular, since in NER multiple text spans in a document may be labeled using the same NE category and in this "QA for NER" approach a single question is used for extracting all the Named Entities related to a category, a "Multi-Span Extractive QA system" is designed and implemented.

Such an Extractive QA-approach tackles the hallucinations problem by answering the question by finding spans of text in the document that may be relevant to the query. Of course, wrong answers could still be given, but for sure the system is not hallucinating by providing non-existent content. Moreover, such a model is designed and trained to extract not only the words as response to a question, but also information about their location in the document, so that to label only those NEs that are linguistically evident from the sentence in which they occur. Lastly, also hardware, APIs costs and privacy concerns could be addressed by the installation on-premises of these systems, characterized by a smaller amount of parameters when compared to LLMs like GPT-3.

The thesis is structured by first presenting a theoretical background for the Named Entity Recognition task, along with a brief overview of the classical approaches to it. The operating principles of Question-Answering systems, both Single-Span and Multi-Span, are then also described in the same chapter. Eventually, after more deeply discussing the motivations and strategies for approaching NER through a QA system, the next chapter, Chapter 3, introduces a novel Multi-Span Extraction QA

model, implemented following specific guidelines. This guidelines are oriented to provide a head-start in the NER task by leveraging the model's pre-training on a corpus of general-type single-span questions.

Chapter 4 is then dedicated to presenting the experiments conducted to evaluate the effectiveness of this QA-based approach to NER. Here, the BUSTER dataset is presented. BUSTER is an abbreviation for "BUSiness Transaction Entity Recognition". As the name suggests, it is an industry-oriented Named Entity Recognition dataset in the financial domain with focus on the main actors involved in a business transaction. On this dataset the novel MS-EQA model is tested and eventually compared with other four models that constitute the baselines of the classical approach to NER.

Lastly, two investigative experiments are conducted: one to evaluate the QA-based approach in scenarios with limited training data and another to test how rapidly the model forgets its capabilities on the SQUAD2 dataset, on which is pre-trained, during its fine-tuning on the new BUSTER dataset.

The experiments conducted allows to conclude that, for the same underlying NLM model, the QA approach to NER achieves performance comparable to the traditional NER, while, thanks to the implementation design guidelines, it proves to be a valuable tool in scenarios with limited training data.

2 Theoretical Background

2.1 Named Entity Recognition

2.1.1 Task definition

Named Entity Recognition (NER) is a natural language processing task that involves detecting and categorizing *Named Entities* in a text.

Named Entities are, in the most classic and historic setting, usually: PERSON, LOCATION, ORGANIZATION, TIME, DATE and MONETARY VALUE. Other NER datasets exist, which may consist of a richer set of Named Entities (from 8 up to 200 NE categories), often specifically defined for the domain in which the NER task is adopted.

As Marrero *et al.* [6] indeed report, several and different definitions of what a Named Entity is have been proposed over the years. Named Entities have been defined using the grammatical category of *Proper Nouns*, although are immediately clear the limitations of this definition when NER defines categories also for Named Entities that are not Proper Nouns (especially when NEs are defined for a particular domain). Other works on NER have defined Named Entities using Kripke's definition of *Rigid Designator*: a word (or group of words) is considered a rigid designator when it designates (it denotes/refers to) the same thing in all possible worlds in which that thing exists. The limitations of this notion for practical purposes, where terms like "phosphorus" may refer both to the planet "Venus" (LOCATION) and the chemical element (we may define a NE called CHEMICAL ELEMENT), are also instantly evident. Similar to this notion is the definition based on *Unique Identifier*: expres-

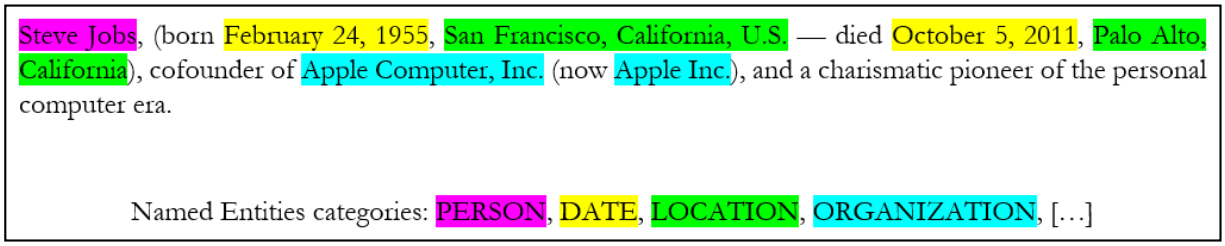


Figure 2.1: NER on one sentence example. NER can be seen as the task of highlighting with different colors one or multiple words in a text, where each color corresponds to a different Named Entity category.

sions to be annotated are "unique identifiers" of entities. However, once again, the same entity (like Phosphorus) might designate different referents, and so what the "unique identifier" is considered is actually the referent of that to which we refer. This implies that not unique entities, which may lead to ambiguity, should not be labeled. We instead know that NER requires also these named entities to be labeled and one of the challenges of NER is indeed resolving ambiguity, from the context or through previous knowledge. The last commonly found definition and, the only one that appears to be consistent with works in the literature, defines Named Entities according to *the purpose and domain of application*. Under this definition we can see the Named Entities in the most classic NER setting as responding to some of the 5 typical questions in the journalism domain: what (ORGANIZATION), where (LOCATION), who (PERSON), when (DATE and TIME) and why (not classic NE associated). Other NER datasets with a greater or different number of defined Named Entity categories are instead oriented to extract specific type of information for the domain in which the NER task is adopted. An example is the dataset that will be presented and used for the experimental evaluations in this research project, which is oriented toward the domain of business transactions.

Let's use the NER example in Figure 2.1 to make a more concrete example of what NER consists of. Making it in simple terms, NER can be seen as the task of highlighting with different colors one or multiple words in a text, where each color corresponds to a different Named Entity category (for the example above we have used the classic ones we have listed: person, location, organization, time, date). It becomes also evident why NER is defined as the task of *detecting* and *categorizing* Named Entities: one has to decide where a NE start and ends (an entity may consist of one or more words), as well as which category to place it in.

2.1.2 Approaches to NER

Gazetteers and Rule-based approaches

As Mikheev *et al.* [7] suggest, one could think that the task of NER could be easily solved by having huge lists of, for example, names of people, places and organizations. But this, in practice, would not work for three main reasons: the lists would be huge (it is estimated that there are 1.5 million unique surnames just in the U.S.), the lists would immediately be out of date since, e.g. for ORGANIZATION, new companies are every day founded. Lastly, named entities can occur in variations, like in our example where "Apple" can appear as "Apple Computer Inc.", "Apple Inc." or just "Apple". This latter case shows also another difficulty of NER, that is: ambiguity in labeling. It must be understood from the context or using some previous knowledge whether "Apple" denotes the fruit item or a company name, and therefore, in the latter case, label it as ORGANIZATION. Thinking of implementing a NER system with lists (often referred to as *gazetteers*), this would correspond to having the same entity in more than one list. One step further in this direction, in the first NER systems, was then to assign the entity to the list (category) with highest probability, with the help of grammatical patterns and rules that were specifically defined to recognize entities based on the context in which they appeared. For example, if a word was capitalized and preceded by a title (e.g., "Mr.", "Dr."), it might be tagged as PERSON. Similarly, if an entity ended with "Ltd." it was likely to be an ORGANIZATION.

These methods had drawbacks, though, since it took a lot of work to develop and maintain lists for various entity categories, and they sometimes struggled to deal with ambiguous or unseen cases.

ML Feature-based approaches

The transition from grammar rules based systems to feature-based systems involved the adoption of Machine Learning (ML) techniques, particularly Hidden Markov Models (HMM), Conditional Random Fields (CRF) and Support Vector Machines (SVM). These techniques allow NER systems to learn patterns and features directly from data, leading to more robust and accurate entity recognition. All these approaches rely on treating NER as a "Sequence Labeling Task" where the goal is, given in input a sequence of elements [1], assign a label to each element [8].

Given the fact that an entity may be composed of multiple elements, a notation capable of representing such is required. An often used notation is the so called *BIO* labeling scheme, which stands for

[1] we use the term *elements* and not *words*, since elements can be tokens at word-level, character-level or sub-word level (more details in this section below)

Steve	B-PERSON
Jobs	I-PERSON
,	O
(	O
born	O
February	B-DATE
24	I-DATE
1955	I-DATE
,	O
San	B-LOCATION
Francisco	I-LOCATION
California	I-LOCATION

Figure 2.2: BIO labeling for NER task on Figure 2.1 sentence example.

Beginning, *Inside*, *Outside*. BIO scheme provides a structured way to encode both the entity boundaries and the entity type by using the tags in the following way:

- B (Beginning): This tag is used to indicate the first element of an entity in the sequence. Each time a new entity starts, the first element of that entity is labeled with *B-TYPE*, where *TYPE* represents the type of the entity (e.g., B-PERSON, B-TIME).
- I (Inside): This tag is used for elements that appear inside an entity, following the beginning element. All subsequent elements that belong to the same entity as the *B-TYPE* element are labeled with *I-TYPE*
- O (Outside): This tag is used for elements that do not belong to any entity. All elements that are not part of an entity are labeled with *O*.

The same sentence in Figure 2.1, would be therefore labeled using the BIO scheme as reported in Figure 2.2.

The scheme explicitly marks the beginning and end of each entity, together with its type. It also allows to handle cases where multiple entities of the same type appear consecutively in the text, ensuring proper recognition of all entities. Lastly, it provides a standardized way of representing entities, making it easier to share and compare annotated data across different NER systems.

Finally, before briefly looking at how each of the above ML-based methods works, it is important to emphasize that the NER Sequence Labeling task must be approached as a "Structured Prediction Problem". Many tasks in NLP can be written mathematically in the form of optimization [3]:

$$\hat{y} = \underset{y \in Y(x)}{\operatorname{argmax}} \Psi(x, y; \theta) \tag{2.1}$$

where:

- x is the input, which is an item of a set X (e.g. a sequence of elements)
- y is the output, which is an item of the set $Y(x)$ (e.g. a structured output associated to x)
- Ψ is a scoring function, also called the *model*, which maps from the set $X \times Y$ to the real numbers (given the input x, tells how good an output y would be)
- $\hat{y}$ is the predicted output, which is chosen to maximize the scoring function Ψ

The SPP nature requires to break down the problem in two phases: the first one of *learning* (finding the model parameters θ that maximize the predictive performance) and the second one of *searching* (finding the candidate output $\hat{y}$ with the highest score relative to the input x). Even though the output structure for NER sequence labeling is simpler than that of other NLP tasks (it is simply a linear sequence of elements, one for each input element), the SPP nature of the task implies that it cannot be approached as a classification problem or by exhaustively searching among all possible output candidates in Y(x), so an effective algorithm for the searching phase is needed. Moreover, when deciding at each time step in the sequence what type of the BIO scheme to assign, these searching algorithms can either predict independently on each element, or take into account also contextual dependencies: the context of neighboring words can help disambiguate whether a word is part of an entity or not (e.g., the presence of the word *Inc.* in the *Apple Inc.* example above).

Let's now present the three most used ML-based approaches that rely on the concepts discussed just above:

Hidden Markov Models HMM is a statistical model which computes the probability of assigning a sequence of tags $t_{1:n}$ to an input sequence $x_{1:n}$, namely $P(t_{1:n}|x_{1:n})$, by breaking it down, using Bayes rule, in two probabilities: the prior probability $P(x_{1:n})$ of observing the input sequence x and likelihood of the words given the tags $P(x_{1:n}|t_{1:n})$. Two assumption are then made: the probability of a word depends only on its own tag t and is independent of neighboring words and tags (i.e., $P(x_{1:n}|t_{1:n}) = \prod_{i=1}^{n} P(x_i|t_i)$); second, the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence (i.e., $P(t_{1:n}) = \prod_{i=1}^{n} P(t_i|t_{i-1})$). HMM can be then defined as a special type of *probabilistic finite state automaton*: it involves a set of hidden states (in NER, these

correspond to the NE tags) and observable emissions (the words in the text). Initially, labeled training data is required to estimate the *transition probabilities* between different entity tags $P(t_i|t_{i-1})$ and *emission probabilities* for each word $P(x_i|t_i)$. The system then uses the *Viterbi* decoding algorithm to find the most likely sequence of entity tags for a given input text.

Support Vector Machines SVM is a machine learning algorithm used for classification tasks. In the context of NER, the task is to classify each word in the text as either part of an entity type or not (sequence labeling through BIO scheme, multi-class classification approached as "one vs the rest"). Feature engineering plays a crucial role here, as various features are extracted from the input text, such as the word itself, its context, and part-of-speech tags. The system uses then labeled training data to learn a decision boundary that separates words belonging to entities from those that do not. The decision at each time step is usually made independently from the other (previous or next) assignments.

Conditional Random Fields CRF is a probabilistic model that extends HMM by directly modeling the conditional probabilities of the labels given the input sequence, that is $P(t_{1:n}|x_{1:n})$. Similar to SVM, feature engineering is an essential step. However, CRF considers the context of neighboring words when making predictions, leading to better handling of dependencies between labels. CRF-based NER systems outperformed HMM and SVM in many cases due to their ability to capture complex interactions between words and labels.

Overall, the transition from grammar rules based systems to feature-based systems using CRF, HMM, and SVM marked a significant improvement in NER performance and flexibility. These machine learning approaches allowed NER systems to learn from data, reducing the reliance on handcrafted rules and enabling the recognition of various NE types through the discovery of more complex patterns in text. However, despite manually engineered features can be effective to some extent, they have several limitations:

- Labor-intensive and time-consuming: creating effective features requires domain expertise and significant manual effort. It involves understanding the problem, analyzing the data, and designing features that capture relevant patterns.

- Limited generalization: manually crafted features are often specific to the task and domain they were designed for. They may not generalize well to different datasets or languages, making them less adaptable and transferable.

- Information loss: handcrafted features might not capture all the relevant information present in the text, leading to the loss of potentially valuable context and semantics.
- Scalability: as the complexity of the NER task and the size of the data increase, handcrafting features becomes less feasible and less efficient.

Deep Learning based approaches

Due to the above mentioned drawbacks, the rise of Deep Learning approaches has revolutionized NER (as well as many other NLP tasks). Deep learning models, especially sequence-to-sequence models, have become popular for NER and demonstrated remarkable performance improvements. Some key advantages of deep learning approaches for NER include:

- End-to-end learning: DL models learn to extract relevant features automatically from raw text, eliminating the need for manual feature engineering.
- Representation learning: DL models can learn rich and distributed representations of words and sentences, capturing complex relationships between words and contextual information.
- Improved performance: DL models have achieved state-of-the-art results on NER benchmarks due to their ability to model complex patterns and dependencies in language.
- Generalization: DL models can generalize better to new and unseen data, making them more robust across different datasets and languages.
- Transfer learning: pre-trained language models can be fine-tuned on NER tasks, providing a head start and improved final performance.

In literature, as the survey on DL-based approaches to NER [9] reports, the amount of DL-based solutions to NER is enormous. Making a list of summaries for each approach would not be helpful; instead, it can be more interesting and useful to describe the three main components of a typical DL-based NER system, to present solutions for each component, and to be aware that different overall systems can be built by combining these options.

Most end-to-end DL-based systems for NER (approached as Sequence Labeling Structured Prediction problem) consist of the following three components:

Distributed representations for input: different representations for the input exist, which can be classified in *word-level, character-level* and *hybrid*. Word-level representations are typically pre-trained over a large collection of text through unsupervised algorithms like continuous bag-of-words (CBOW) or skip-gram models. Other commonly used word embeddings include Word2Vec, GloVe and fastText. Character-level representations have been used for explicitly modeling sub-word information, such as prefix and suffix, as well they provide a straightforward solution to handle out-of-vocabulary words. Architectures employing CNN and RNN models are used to learn character-level representations. Finally, hybrid solutions mix word and char level representations, as well they may also incorporate additional information like part-of-speech tags, linguistic dependencies, lexical similarity, etc.. It is worth to mention that these pre-trained embedding representations can be fixed or further fine-tuned during the overall system end-to-end training.

Context encoder: provided the distributed input representations, this component is in charge of capturing the context dependencies using CNN, RNN or Transformer-based architectures. In particular, Recurrent Neural Networks (along with LSTMs and GRUs variants), have achieved remarkable results in modeling sequential data, with Bidirectional LSTMs efficiently making use of past information (via forward states) and future information (via backward states) at each time step. With the raise of the Transformer architecture and pre-trained Neural Language Models, a new paradigm has established in approaching NER. Due to the importance of this architecture in this research project, a dedicated section is used to provide further theoretical background about it (see Section 2.2).

Tag decoder: this final component takes context-dependent representations as input and produce a sequence of tags (using, for example, the BIO labeling scheme). Three main architectures are employed for tag decoders: MLP + softmax layer, Conditional Random Fields and Recurrent Neural Networks. With the use of the BIO labeling scheme the sequence labeling task is modeled as a multi-class classification problem at each time step. The MLP + softmax architecture differs from the other two in that it predicts independently at each time step, only on the basis of context-dependent representations, without taking into account its neighbors. It has to be remarked that CRFs often provide better results when the chosen context encoder is a BiLSTM, while being too much computationally expensive when the number of entity types is large and not always leading to better performance (compared to MLP + softmax) when adopting contextualized embeddings from transformer based language models.

2.2 Language Modeling

2.2.1 Task definition

In NLP, Language Modeling (LM) is the task of estimating the probability distribution of words in a text sequence. A language model's objective is to calculate $P(w_{1:n})$, that is, the probability of observing a specific sequence of words in a given language.

Assuming a vocabulary V and sentences in a language L modeled as strings over V, a LM mathematically computes the conditional probability of a sequence of words

$$w_{1:n} = (w_1, w_2, \ldots, w_n) \tag{2.2}$$

as the product of the conditional probabilities of each word given its preceding words:

$$P(w_{1:n}) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot \ldots \cdot P(w_n|w_1, w_2, \ldots, w_{n-1}) \tag{2.3}$$

According to the above definition using the Chain Rule for probabilities, language modeling can therefore also be defined as the task of predicting which word comes next in a sequence of words, by computing the probability $P(w_t|w_1 w_2 \ldots w_{t-1})$.

Language modeling represents a crucial task in NLP with significant importance both as a standalone task and as a foundational component for various other NLP applications. As a standalone application LM can help identify and correct spelling and grammatical errors by suggesting contextually appropriate replacements. It also can assist in improving the quality and correctness of written content. As helper component in other NLP applications, LM is instead directly involved in text generation tasks: by modeling the probabilities of word sequences, language models can generate coherent and contextually relevant text. In language understanding tasks LM helps capture syntactic and semantic relationships between words, enhancing the comprehension of text. There are many other NLP applications in which LM plays an important role (including the NER task presented above), in most of them as a method for learning language representations and producing embeddings for the input.

These embeddings can be *static* or *dynamic* (also called *contextualized*), depending on the language model they rely on.

Static word embeddings represent words as fixed vectors in a continuous vector space. These embeddings are learned from large text corpora in an unsupervised way using, for example, *N-gram*

language models [2]. Each word is assigned a unique vector representation, which captures certain linguistic properties like semantic relationships and word co-occurrence patterns. Importantly, these embeddings are context-independent, meaning that the same word will have the same vector representation regardless of the context it appears in. Thus, they are not able to produce different embeddings for words that have multiple senses (polysemy).

Contextualized word embeddings designed to capture the meaning of words within their specific context in a sentence or document, these embeddings consider the surrounding words and their relationships to generate a dynamic representation that reflects the word's meaning in that particular context.

2.2.2 Neural Language Models

Contextualized word embeddings represent one of the outcomes of Language Modeling based on deep neural networks architectures. Good representations can be obtained by LM relying on RNNs architectures, such as from ELMO (standing for *Embeddings from Language Modeling*), which relies on Bi-LSTMs. But the major breakthrough came with the advent of the Transformer architecture and Language Models such as BERT and GPT.

Following is a brief overview of the Transformer architecture that was firstly introduced by Vaswani *et al.* [1] and the description of RoBERTa LLM (developed by Liu *et al.* [10]), which is the model employed in this research project's experiments.

Transformers Architecture

Transformers architecture was introduced in a 2017 paper which name is self-explanatory: *Attention Is All You Need*. The title itself conveys the central role that the attention mechanism plays in this model, which enables the model to weigh the importance of different parts of an input sequence when producing each element of an output sequence. The attention mechanism is used for both capturing context and relationships within a sequence (self-attention), both for aligning source and target sequences (cross-attention).

In Figure 2.3 is illustrated the original proposed transformer architecture, which consists of an encoder and a decoder, both composed of multiple layers. Let's have a look at how an input sequence is processed by the transformer model.

[2] An N-gram language model predicts the probability of a word based on the preceding N-1 words in a sequence, i.e., by computing $P(w_n|w_{n-1}, w_{n-2}, \ldots, w_{n-N+1})$

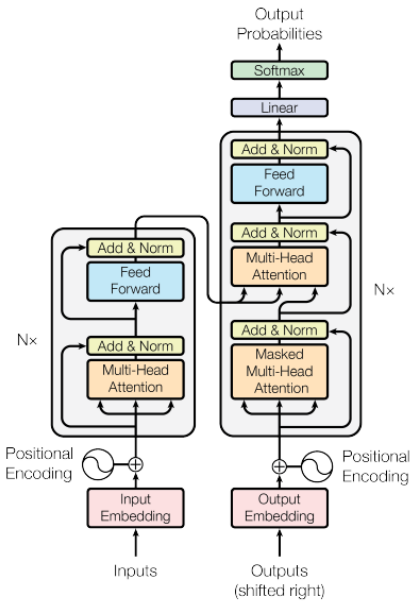


Figure 2.3: Transformer architecture illustration from [1].

Through the encoder:

1. Input Embeddings: the input sequence is first represented as a matrix, where each word is mapped into a high-dimensional vector and stacked as row in the matrix. For this step, the distributed representations at word-level, character-level or hybrid discussed in Section 2.1.2 apply. Input matrices are of fixed dimension, therefore techniques like sentence truncation and padding are used.

2. Positional Encodings: since transformers lack inherent positional information (in RNNs position is implicitly modeled in their recurrent formulation), positional encodings are added to the input embeddings. These encodings provide information about the order of words in the sequence and are usually computed through sine-cosine functions.

3. Self-Attention Mechanism: in each encoder layer, the input embeddings along with their positional encodings pass through a self-attention mechanism. This mechanism computes attention scores for each word based on its relationships with all the other words in the sequence. The attention scores help the model focus on relevant words and capture context. The self-attention mechanism is often multi-head, namely Multi-Head Attention, meaning it runs multiple times in parallel, each focusing on different aspects of the input. The outputs of the multiple attention heads are concatenated and linearly transformed to create the final attention output.

4. Addition and Normalization: The output of the attention mechanism is added to the input embeddings (residual connection) and then normalized (layer normalization).

5. Feedforward Neural Network: The normalized output passes through a feedforward neural network. This network applies non-linear transformations and learns complex patterns in the data.

6. Residual Connection and Normalization: similar to step 5, the output of the feedforward network is added to the previous output and then normalized.

7. Repeating Layers: steps 3 to 7 are repeated for multiple encoder layers. Each layer refines the representation of the input sequence based on the context captured by the self-attention mechanism and feedforward network.

Through the decoder:

1. Input embedding and positional embedding: similar to the encoder, also the decoder output is embedded in the above described way. This characterizes the transformer architecture to be *auto-regressive*, with the true targets during training (teacher forcing), while the already output vectors during inference.

2. Self-Attention Mechanism (Masked): the decoder also uses a self-attention mechanism, but with a slight modification. To maintain causality during training (not using future information), a masking mechanism is applied to ensure that each word can only attend to preceding words.

3. Encoder-Decoder Attention: the decoder's self-attention output is used to compute attention over the output of the encoder. This mechanism helps the decoder focus on relevant parts of the input sequence.

4. Repeating layers: last 2 steps in the decoder block are repeated for multiple decoder layers, along with the same addition and normalization, FNN and residual connections that were discussed for the encoder layers.

Finally the output is generated: the output of the decoder's last layer is passed through a linear transformation followed by a softmax function to produce probabilities for each word in the output vocabulary. The linear transformation is needed to map to the target vocabulary dimension. During inference, the predicted word with the highest probability is selected as the next word in the output sequence. This word is then fed back into the decoder to predict the next word, and the process continues until an end token is generated, in an auto-regressive way as previously mentioned.

RoBERTa Language Model

The following is an overview of the RoBERTa Neural Language Model: a model which has been widely employed in the experiments of this research project.

To properly understand the technical aspects of the RoBERTa model, firstly the BERT (Bidirectional Encoder Representations from Transformers) [2] model is presented, as RoBERTa is an extension and improvement of it.

BERT is a multi-layer bidirectional transformer-encoder-based model. The bidirectional encoding is achieved by dropping the masking in the self-attention, allowing the mechanism to range over the entire input, therefore computing the bidirectional context vector for a word by considering both left and right context words. This is achieved through a Masked Language Model (MLM) pre-training objective, where certain words are masked out, and the model learns to predict them based on the surrounding context. More precisely, during pre-training, a given percentage of tokens (usually 15%) is masked by: a special mask token with probability 0.8, a random token different from the one masked with probability 0.1, the same token with probability 0.1. This pre-training objective helps the model learn bidirectional context by predicting masked words based on surrounding information, capturing deeper linguistic relationships. A second pre-training objective function is used, that is, Next Sentence Prediction (NSP). When doing NSP inputs are two sentences A and B, with a separation token [SEP] in between. With probability 50% the sentences are consecutive in the corpus, in the remaining 50% they are not related. The model has to predict if the sentences are consecutive or not. This latter pre-training objective encourages the model to grasp sentence-level relationships and coherence, which is crucial for tasks on which the model is usually applied on via fine-tuning, like question answering, text completion, and conversation. Figure 2.4 illustrates these pre-training objectives on a sentence pair (*[CLS] sentenceA [SEP] sentenceB [SEP]*). Regarding the input, it is important to say that BERT input is tokenized using WordPiece tokenizer (with a 30000 token vocabulary), which breaks words into smaller sub-word units for flexible handling of morphological rich languages and out-of-vocabulary-words. Input embeddings are enriched with *positional embedding* about the position of the token in the sentence and *segment embedding*, which encode information about the token belonging to the first or second sentence in the pair.

Based on what has been said above for the BERT model, Liu *et al.* [10] found that BERT was significantly undertrained. As a result, they developed a novel recipe for training BERT models. The resulting model was eventually named *RoBERTa*, standing for *Robustly Optimized BERT Approach*.

These modifications include:

- training the model longer, with bigger batches, over more data.

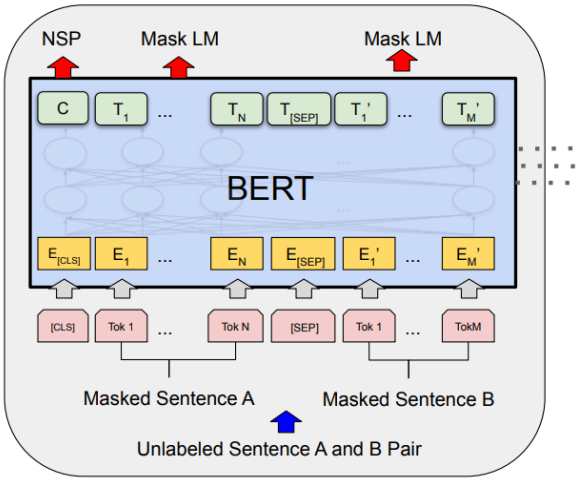


Figure 2.4: BERT architecture illustration from [2].

- removing the Next Sentence Prediction pre-training objective.
- training on longer sequences (to reach the context window size of 512, sentences may span over several documents).
- dynamically changing the masking pattern applied (in BERT pre-training data was ten times multiplied and masked once for all before starting pre-training).
- using a byte-level BPE as tokenizer, with a bigger token vocabulary of cardinality around 50000.
- removing the segment embedding (there is no more the need to indicate which token belongs to which segment, just separate the segments with the separation token $< /s >$).

Summarizing, RoBERTa, which has an architecture of 12 layers, 768 hidden-dimension vectors, and 12 heads, is an upgraded version of BERT.

2.3 Extractive Question Answering

Machine Reading Comprehension (MRC) is a sub-field of NLP that focuses on developing algorithms and models to enable computers to read and understand human language text. This comprehension skill is evaluated in MRC by assessing the algorithms' ability to answer questions about a certain passage of text that is presented in input.

MRC has applications in information retrieval, question-answering systems, virtual assistants, document summarization, and more. It plays a crucial role in making large amounts of textual information more accessible and usable by enabling machines to read and comprehend human language effectively. MRC tasks vary in complexity, ranging from simple factoid questions (where the answer is a direct piece of information from the text), to more complex reasoning and inference questions (where the answer requires combining information from different parts of the passage).

A direct consequence of these two types of MRC tasks can be seen in the two main approaches to MRC, that are:

- Extractive MRC: in this approach, the answer is directly extracted from the passage. The system identifies a span of text within the passage that contains the relevant information needed to answer the question. The extracted span of text is then presented as the answer to the question.

- Abstractive MRC: in this approach, the system generates an answer by synthesizing information from the passage rather than directly extracting it. This may involve paraphrasing and summarizing the content of the passage to create a coherent answer.

The focus of this research project is the development of a model that falls under the first discussed approach to MRC, namely Extractive MRC, also commonly referred to as Extractive Question Answering. For this reason, the following is an overview of approaches in literature to Extractive QA, omitting a more in-depth study of the Abstractive approaches.

According to the survey on Extractive Question Answering by Wang *et al.* [11], EQA models usually consist of four components:

1. Embedding module: context and question in input to the model are encoded into some vectorial representations.

2. Feature extraction module: RNN, CNN or Transformer models are used to compute and extract contextual representations for each word vector embedding fed in input, both of the question and the context.

3. Context-question interaction module: uni-direction or bi-directional attention mechanisms may be used to emphasize the more important parts of the input text that are more associated to the question.

4. Answer prediction module: finally, one or more spans of text are predicted as answer(s).

Based on the output of the last component, it is common in literature to distinguish EQA systems into *single-span EQA* and *multi-span EQA*.

2.3.1 Single-span Extractive QA

As the name suggests, the task of single-span Extractive QA is to extract, given a question about a passage, only a single continuous span of text from the given passage as answer. The output may also be an empty string if there is no answer in the passage to the question.

Formally, let $q = q_1, \ldots, q_N$ be a question and let $x = x_1, \ldots, x_N$ be a passage, where q_t and x_t are tokens at word-level or sub-word level, as discussed for the NER task in Subsection 2.1.2. A span is any fragment $x_i, \ldots, x_j$ of x, with i the start position and j the end position. The goal of a SS-EQA system is to compute the probability $P(x_i, \ldots, x_j | q, x)$ that a span $x_i, \ldots, x_j$ is the answer to q on passage x, so that the span with the highest probability can be selected as the correct answer.

It is now presented the most common nowadays neural approach to Single-span EQA, which relies on contextual embeddings. Other neural approaches based on attention-like mechanism such as Stanford Attentive Reader (by Danqi Chen in 2018) and BIDAF (Bidirectional Attention Flow for Machine Comprehension by Minjoon Seo *et al.* in 2017) have been a landmark in SS-EQA, but an overview of also these approaches would take us in a different direction with respect to what this research project is about.

A Neural Language Model based approach to Single-Span Extractive QA

In line with what has been said in 2.2 about contextual embeddings obtained from pre-trained Neural Language Models, also SS-EQA can leverage its performance on these models. As illustrated in Figure 2.5, a typical NLM-based SS-EQA system is characterized by:

- a Neural Language Model (e.g. BERT) used to encode the input, which is obtained by concatenating the passage on the right of the question, separated by the [SEP] token.
- a MLP classifier on top of the encoder, used to estimate the probabilities of *answer start* and *answer end* for each position i in the passage.

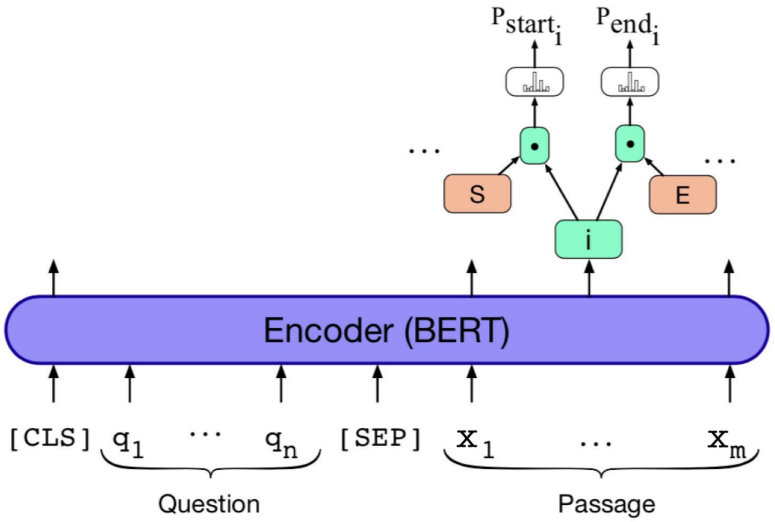


Figure 2.5: Single-span Extractive QA architecture (based on Neural LM) illustration from [3].

Defined $e(x_i)$ the embedding of token x_i within passage x, the probability of the answer starting at position x_i is computed by learning a start vector S such that:

$$P_{start}(i|q,x) = \frac{\exp\left(S \cdot e(x_i)\right)}{\sum_j \exp\left(S \cdot e(x_j)\right)} \tag{2.4}$$

Similarly an end vector E is learned to estimate $P_{end}(j|q,x)$.

Equation 2.4 is computing the dot product between a learnable vector and the passage token embedding at position i, to which is then applied a softmax function. The softmax function exponentiates each element of the input vector (in this case the vector represents a score for each token in the passage, and thus has dimension equal to the length of the passage) and then normalizes the resulting values by dividing them by the sum of all exponentiated values. This normalization ensures that the output probabilities are within the range $[0,1]$ and that they add up to 1. Moreover, it magnifies the differences between the elements in the input vector: larger input values will be exponentiated to larger values, which means they will have higher probabilities in the output distribution. Similarly, smaller input values will result in smaller probabilities. This property makes the softmax useful to convert raw scores (logits) into a distribution probability over the passage tokens. Finally, the softmax function is particularly relevant in the single-span EQA setting as its allows the score of a single starting position to emerge while pushing down all the other scores, doing similarly for the score of a single ending position.

Under the simplifying assumption that

$$P(x_i, \ldots, x_j|q, x) = P_{start}(i|q, x) \cdot P_{end}(j|q, x) \tag{2.5}$$

the score of a candidate span from position i to position j is computed as:

$$S \cdot e(x_i) + E \cdot e(x_j) \tag{2.6}$$

The model answer is then the highest scoring span with $j \geq i$, that is, it satisfies the constraint that the end span position follows or coincides (in case of single token answer) with the starting position. Another constraint is that the start and end positions must not point on tokens belonging to the question or special tokens such as [SEP]. Instead, it is allowed for both the start and end positions to point to the [CLS] token if the answer to the question is not present in the passage and thus the model refuses to select a span of text as the answer.

The simplifying assumption in Equation 2.5 states that the probability of a span can be modeled by predicting independently the *start* and *end* positions, given the question q and the passage x. It is a strong assumption to model where an answer ends without knowing where another classifier has decided to make it begin, but in practice, models using neural language models such as BERT, which rely on a strong bidirectional attention mechanism, are able to capture dependencies and move information, conditioned on the question q, to the locations in the passage x so that the classifiers can, efficiently and independently, decide for each token whether it is a start or end position. This factorization makes the model more efficient during training and inference, making it computationally less expensive compared to generating a distribution over all possible spans (which would be much more resource-intensive), without sacrifices in terms of performance thanks to the strengths of the underlying language model.

Looking instead at Equation 2.6, we can observe that it does not fully reflect the simplifying assumption in Equation 2.5. We know that, when the highest scoring span is selected through an argmax, the softmax functions are negligible, but then, why is there a summation and not a product? The short answer is that, in practice, it works better. The theoretical explanation may be that the additive scoring mechanism provides a way to capture potential variations in the importance of start and end positions when predicting a span. This flexibility acknowledges that not all positions within the span contribute equally to the overall answer. In cases where the answer information is more likely to be at the start of the span, the model can assign a higher weight to $S \cdot e(x_i)$. Conversely, if the information tends to be at the end, a higher weight might be assigned to $E \cdot e(x_j)$. Using an additive score allows the model

to better capture the nuances of how different parts of the text contribute to the overall answer, as opposed to a product score, which would collapse a pair score if one of the starting or ending scores is small, perhaps simply because the information is more concentrated in the opposite part of the span.

We have presented a typical single-span extractive QA model architecture based on Neural Language Models such as BERT, the simplifying assumption adopted, and the extraction algorithm used for inference based on a summation score. Lastly, it has to be described how such a model is trained. Since the underlying model is pre-trained in an unsupervised way on a large corpora, to leverage its pre-learned language understanding, *fine-tuning* is usually performed. This process consists in training the model on task-specific data using a properly defined loss function. During fine-tuning, the model adjusts its parameters based on the new data while retaining the valuable linguistic knowledge it gained during pre-training. For the task of single-span Extractive QA a common loss function used is the average (or sum) of two components: the *start position loss* and the *end position loss*. These two components capture the model's ability to independently predict the correct start and end positions for the answer span. The most commonly used loss functions for this purpose are the negative log likelihood (cross-entropy) loss and, sometimes, variants of it. The start position loss can be mathematically represented as:

$$L_{start} = -\log P_{start}(true\_start\_position) \tag{2.7}$$

where the *true_start_position* is represented through a one-hot vector over the input tokens. This component measures how well the model predicts the correct start position of the answer span by computing the negative log likelihood between the true start position and the predicted probability distribution over all input positions. Similarly, a loss component is defined for the end position, and the overall loss is the average of the two terms:

$$L = \frac{-\log P_{start}(true\_start\_position) - \log P_{end}(true\_end\_position)}{2} \tag{2.8}$$

These loss components encourage the model to assign higher probabilities to the correct start and end, while penalizing incorrect predictions. By using the negative log likelihood loss, the model aims to reduce the discrepancy between the predicted probabilities and the true positions.

In the above loss we can see that there is no account for the constraint of the end position to follow the start position, both because of the simplifying independence assumption, both for the difficulty of expressing such a constraint in a differentiable form.

A final note has to be made to deal with the practical case in which the text passage does not fit into the model context-window input. It is common practice to use a *sliding window approach*: the

long text passage is divided into overlapping segments (a *stride* hyperparameter controls the amount of overlapping text), these segments are processed independently by the model, and finally the model outputs are combined by some aggregation process. The aggregation may involve merging overlapping predictions, selecting the most confident predictions, or any other suitable strategy.

2.3.2 Multi-span Extractive QA

As the name suggests, Multi-span Extractive QA extends the above task by requiring the design of algorithms and models that allow the extraction of not just a single contiguous span of text, but multiple non-contiguous spans from a given passage, in order to answer a question.

In Single-span Extractive QA, the focus is on identifying the most salient piece of information that best answers to the question. In contrast, the Multi-span Extraction QA task requires the algorithm to identify and assemble multiple distinct segments of text, potentially scattered throughout a passage, or even across multiple passages in which a long document has been segmented using the sliding window approach, that together form a coherent and accurate answer to the question. This task presents therefore a higher level of challenge due to its demand for a deeper understanding of context to capture nuanced relationships, dependencies, and interconnections among various pieces of information in the passage, and a more complex mechanism of spans aggregation.

If it is true that for Single-span EQA many datasets have been collected to train these models in a supervised way (*SQUAD2*, the most famous and used as a benchmark, will be presented in the Experiments chapter), the same cannot be said for the multi-span setting. Only in 2019, a first dataset has been released, that is *DROP* (Discrete Reasoning Over the content of Paragraphs) [12], and studies on multi-span extraction have appeared one after the other. DROP dataset contains 96.6K question-answer answer pairs and the passages are extracted from Wikipedia. The answers to these questions cover a variety of types, such as time, dates, numbers, etc. Also, some questions have multiple answers and may require some reasoning to answer, such as numerical comparison. DROP emphasizes the importance of discrete, step-by-step reasoning over the content of paragraphs to answer complex questions, challenging models to go beyond simple information retrieval and engage in deeper comprehension and logical inference.

For the characteristics of the dataset described above, the multi-span extraction QA models that have emerged not only focus on the new ability to extract multiple spans of text, but are also equipped with the ability to perform discrete reasoning through operations such as addition, counting, or sorting.

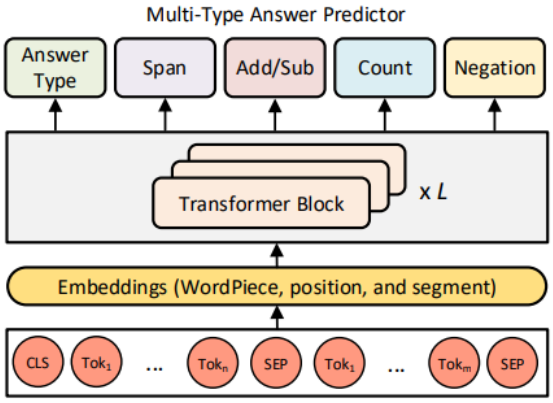


Figure 2.6: MTMSN architecture illustration from [4].

Multi-Type Multi-Span Network

Developed by Hu *et al.*, one of the earliest DROP-oriented systems is called MTMSN, standing for Multi-Type Multi-Span Network [4]. MTMSN is a neural reading comprehension model for predicting various types of answers, as well as dynamically extracting one or multiple spans. As Figure 2.6 illustrates, it utilizes a series of pre-trained Transformer blocks (BERT) to obtain a deep bidirectional representation. On top of it, a multi-type answer predictor is used to predict the type of answer between the categories: text span, count number, arithmetic expression and logical negation. Moreover, when the text span type is selected, the model first predicts the number of answers, and then extracts non-overlapped spans to the specific amount that was predicted. It is interesting to report how they modeled the prediction of the answer type, the number of spans to be extracted, and the probabilities of the start and end indices.

For the answer type prediction they used the contextualized token representations from the last four blocks, denoted as M_0, M_1, M_2, M_3, where M_3 represents the last layer representations. They first split, according to the [SEP] token, the representation M_2 into a question representation Q_2 and a passage representation P_2. Then the model computes two vectors h^{Q_2} and h^{P_2} that summarize the question and passage information respectively:

$$\alpha^Q = softmax(W^Q Q_2) \tag{2.9}$$

$$h^{Q_2} = \alpha^Q Q_2 \tag{2.10}$$

W^Q is a learnable matrix that is used to compute attention scores α^Q over the query vectors Q_2, that are then used to compute a weighted sum of the original query vectors in Q_2 as vectorial representation

of the entire matrix Q_2. h^{P_2} is computed in a similar way over P_2.

The probability distribution over the different answer types is then modeled as a classification problem:

$$p^{type} = softmax(FNN([h^{Q_2}; h^{P_2}; h^{CLS}])) \tag{2.11}$$

h^{CLS} is the first vector of the last layer contextualized representations M_3 and FFN denotes a feed-forward-neural-network with two non linear projections and GeLU activation functions and a layer normalization in between. The semicolon denotes the concatenation of the three vectors.

A similar model to Equation 2.11 is used to predict the number of spans to be extracted, still modeled as a classification problem, with different learnable weights for the FNN:

$$p^{span} = softmax(FNN([h^{Q_2}; h^{P_2}; h^{CLS}])) \tag{2.12}$$

In their experiments they used a maximum number of spans of 8, thus p^{span} models a probability distribution over a vector of dimension $\mathbb{R}^{8\times 1}$.

The probabilities of the start and end indices of an answer span are instead computed as follows. First, they compute three vectors, namely $g^{Q_0}, g^{Q_1}, g^{Q_2}$, that summarize the question information at different levels of question representations:

$$\beta^Q = softmax(FNN(Q_2)) \tag{2.13}$$

$$g^{Q_2} = \beta^Q Q_2 \tag{2.14}$$

g^{Q_0} and g^{Q_1} computed over Q_0 and Q_1 similarly.

Then p^{start} and p^{end} are computed according to the following equations:

$$M^{start} = [M_2; M_0; g^{Q_2} \otimes M_2; g^{Q_0} \otimes M_0] \tag{2.15}$$

$$M^{end} = [M_2; M_1; g^{Q_2} \otimes M_2; g^{Q_1} \otimes M_1] \tag{2.16}$$

$$p^{start} = softmax(W^S M^{start}) \tag{2.17}$$

$$p^{end} = softmax(W^E M^{end}) \tag{2.18}$$

where $\otimes$ denotes outer product between the vector g and each token representation in M.

Putting all together, as Figure 2.7 illustrates, if the p^{type} predicts that a text span answer has to be

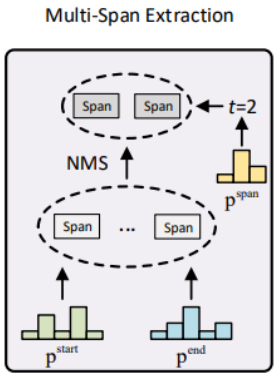


Figure 2.7: MTMSN multi-span extraction mechanism illustration from [4].

used as response, the model first uses p^{span} to predict the number of spans t to extract, and then the model proposes a set of top-K spans S according to the descending order of the span score, which is computed by multiplying the start and end probabilities. A new empty set S' is initialized. The model then iteratively adds the span s_i in S that possesses the maximum span score to the set S', and removes it from S. Also any remaining span s_j that overlaps with the added s_i (where the degree of overlap is measured using a text-level $F1$ function) is deleted from the set S. This process is repeated for the remaining spans in S, until S is empty or the size of S' reaches t.

Multi-span Extractive QA as sequence labeling problem

Below, another interesting approach to Multi-span Extractive QA is presented, proposed by Segal *et al.* [5]. They observed that extracting a variable number of spans from an input text is a standard in many Natural Language Processing tasks, such as Named Entity Recognition. As discussed in Subsection 2.1.2, a common approach to NER is to cast it as a sequence labeling problem. They therefore decided to apply the same approach to multi-span questions by adopting tagging schemes like BIO and IO (a variant of BIO where the beginning tag B- is omitted). Figure 2.8 illustrates how a multi-answer question from DROP would be labeled using BIO scheme.

With respect to the model architecture defined for Single-span Extractive QA, their model uses the same contextualized representations h from the underlying Neural Language Model encoder (such as BERT or RoBERTA), but rather than predicting start and end probabilities, it outputs, for each token, a probability distribution over a set of tags S.

Given a tagging scheme with $|S|$ tags ($|S| = 3$ for BIO and $|S| = 2$ for IO), for each of the m tokens,

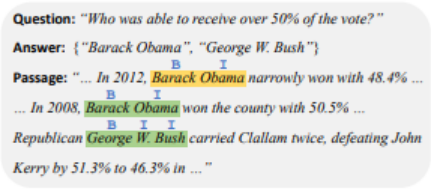


Figure 2.8: A multi-span question from DROP labeled using BIO scheme, illustration from [5].

the probability for the tag of the i-th token is modeled as:

$$p_i = softmax(FNN(h_i)) \tag{2.19}$$

where $p \in R^{m \times |S|}$ and FNN is feed-forward-neural-network with $|S|$ outputs.

They trained the model using cross-entropy as loss function and decided to decode the spans from the model outputs p by using Viterbi algorithm. For BIO tags the set of valid taggings comprises all taggings that do not include an I after an O. In contrast, for the IO scheme, all taggings are valid and maximization can be simply done by predicting the tag with highest probability at each token independently.

2.4 QA for NER

Before explaining what it is meant by saying "QA for NER", it can already be said that this approach represents a counter-trend to what is commonly found in the literature. As stated in Section 2.1, NER can be useful in the task of Information Extraction, for extracting specific Named Entities from natural language text. But NER has proven also helpful as a tool in a variety of other NLP applications, including Question Answering systems. NER techniques are frequently used in QA systems as a mean to facilitate the selection of answers. Indeed, as Marrero *et al.* [6] report, in TREC-8 (Text REtrieval Conference) about 80% of the queries used to evaluate QA systems were of the type who, where and when, which NER categories of the type PERSON, LOCATION, DATE and ORGANIZATION are perfectly suited for identifying spans of text to be used as candidate answers to these questions.

If NER can be an effective tool for the QA task, is the reverse also true? Can QA models be used to solve NER tasks and how? What might be the pros and cons?

The idea behind "QA for NER" is that, instead of approaching NER as a sequence labeling problem and having to decide for each word to which BIO class it belongs to, one tries to find the spans of text in the passage that answer to a question that is properly defined to find the occurrences of a NE category. For example, by asking the question "Which are the words in the passage that refer to a person?", one requires the QA system to return all the named entities that belong to the category PERSON.

2.4.1 In-context learning using LLMs

There is a specific motivation behind approaching the NER task by asking questions, that is: the emergence and spread of Large Language Models such as GPT and the new paradigm of using these models known as "in-context learning". Accurately describing this family of models (Generative Pre-trained Transformers), and the associated emerged technique, would lead to an enormous digression with respect to what this research project is about. However, it can be helpful to have a short presentation that may be just enough to understand why this approach using these generative models would not work, or at least, not that easily.

The term "in-context learning" appeared for the first time in a 2020 paper by OpenAI named "Language Models are Few-Shot Learners" [13]. In this paper they first introduce a new model, which they call GPT-3, and then they focus on experimenting new emerging abilities of it. Similarly to RoBERTa, which was presented in Section 2.2, GPT-3 is a Neural Language Model relying on a Transformer architecture but, differently from RoBERTa, it employs only decoder layers and operates in an auto-regressive manner. RoBERTa performs Language Modeling in a bidirectional way (meaning

it considers both the left and right context of each token when predicting) by having been pre-trained using a masked language modeling (MLM) objective. In contrast, GPT uses a unidirectional language modeling approach: it generates text in an autoregressive manner, predicting the next token in a sequence based only on the preceding tokens. This left-to-right generation process allows GPT to generate coherent and contextually relevant text, but it doesn't have access to future tokens when making predictions.

Despite this unidirectionality may appear a limitation, it is instead what allows such models to generate coherent text, when these models are powered by architectures with a huge number of parameters (e.g., GPT-3 around 175 billion). "In-context learning", the discovered emerging ability of these models, can be in fact simply seen as the technique of letting GPT model to complete text that is coherent with what precedes (what we have defined as Language Modeling). By properly formulating the text on which GPT conditions the generation (called the "prompt"), one can solve various NLP tasks. The model leverages its pre-trained language understanding and generation capabilities to adapt to the provided context and generate coherent and contextually relevant text, being able to solve even properly cast NLP tasks without having to be fine-tuned on training data, just requiring some examples in the prompt. In-context learning allows to do what is called "few-shot learning", that is: making a ML model perform a task using only a small amount of labeled examples, enabling it to generalize and make predictions on new, similar tasks, with limited data.

Suppose one has to do positive/negative classification on film reviews. The binary classification task could be cast into the following prompt:

- Positive example: "A must-see movie! I was captivated by the characters and their emotional journey." The movie is nice.
- Negative example: "Poorly executed. The plot was confusing, and I couldn't connect with the characters.". The movie is bad.
- Test: "The film had its flaws, but overall, it was an enjoyable experience."
  The movie is ___

GPT model will condition the text generation on the examples provided in the prompt and will likely infer that we are doing film reviews, and thus complete the test sentence with words like *nice, great, enjoyable* or *bad, horrible*, according to the sentence in the test slot.

2.4.2 Generative QA for NER

Several papers in the literature have attempted to rewrite NER as Language Modeling task. One of the possible techniques is to formulate a test sentence for each word in the passage with a template like

"[word] is a ___", provide some examples in the prompt and then let the LLM generate the appropriate NE category for that word. The cons of this approach are evident: if n is the number of words in the passage, it will require to formulate n prompts.

A better approach could then be to formulate the prompt as a question, that is what we call "QA for NER". Suppose one has m NE categories to extract, by defining a question for each category (e.g., "Which are the words in the passage that refer to a person?"), one should be able in m prompts to extract all the searched NE categories in the text. The number of NE categories m is always much smaller than the passage length n. If this approach has computational advantages over the above, what are instead its disadvantages and limitations?

HALLUCINATIONS It is a well known problem of these generative LLMs and refers to their tendency to generate responses that sound coherent but may contain inaccurate or fictional information. A possible implication could then be that the LLM, in response to the question "What are the words in the passage that refer to a person?", generates names of people that may be plausible but are not present in the text passage in the input. Similarly, it has been briefly experimented that even with the instruction "respond with NO ANSWER if no word answers the question", it sometimes still generates wrong or non-existent words in the passage.

INPUT CONTEXT-WINDOW SIZE It refers to the number of previous tokens the model considers when generating the next token in a sequence. In the case of GPT-3, the upper limit is set at 2048 tokens. This technical constraint requires that prompts over long documents have to be broken down into smaller passages and their outputs eventually merged. The length of the passage that can be provided as a test sentence in the prompt is further reduced by the fact that the prompt usually includes some examples before the test sentence, to help the LLM make inferences about the task to be performed.

OUTPUT FORMATTING Asking to return the words belonging to a particular NE category may not be sufficient. Especially for NER categories on domains where some NEs concern a semantic action (e.g. "BUYER"), we may want to know not only the names, but also their occurrences (where they appear) in the passage. This is because there is often the requirement that the NEs to be labeled must be linguistically evident from the sentence in which they occur [3]

[3] The NE label must be discernible within the boundaries of the sentence in which it appears. It must not rely on information from previous or subsequent sentences for its identification; each sentence should contain enough linguistic cues to determine the entity type. See Figure 4.2 for an example.

Hardware, API costs, privacy concerns LLMs like GPT-3 are computationally intensive models that require powerful hardware to train and run effectively. Despite "in-context learning" allows to use these models without usually the need of fine-tuning them, these models are generally not employed "on-premises", but through APIs. Many API-based LLM services charge based on the number of tokens processed, which can become expensive for extensive or frequent usage. Lastly, privacy is a significant concern when using LLMs. When users input text into these models, the data is often processed on external servers or cloud infrastructure. This raises potential privacy issues, as sensitive or personal information could be exposed during processing. In industrial applications this is a significant concern.

2.4.3 Multi-span Extractive QA for NER

We have discussed above the potential disadvantages and limitations of facing the NER task using a Generative QA approach. Is the QA-based approach to NER therefore not suitable? Two of the main drawbacks are the hallucinations problem and formatting the output to extract, not only the words, but also information about where they occur in the passage, to label not all word occurrences, but only those which NE category is evident from the sentence.

Extractive Question Answering systems, introduced in Section 2.3, appear to be the perfect candidate to the above problems. They are not generative, instead they answer to questions by finding spans of text in the document that may be relevant to the query: of course, they could still give wrong answers, but we are sure that they are not hallucinating by providing non-existent content. Moreover, they can be designed and trained to extract not only the words as response to the question but also information about their location in the document: e.g., we may train them to return as response to a question a list of triplets:

$$(\textit{retrieved\_text\_span\_words}, \textit{character\_start\_index}, \textit{character\_end\_index})_t$$

where *retrieved_text_span_words* are the words in the text span that begins at character index *character_start_index* and ends at character index *character_end_index*, relative to the beginning of the document. The information about the start and end indices allow to label only those words which NE category is obvious from the sentence.

In addition, an Extractive QA approach to NER would also address the minor drawbacks about context window size and hardware, API cost, and privacy concerns. The sliding window approach of chunking a long document into multiple passages would be a solution to the first problem. The Language Models that are usually employed as encoders in the Extractive QA systems are of the order

of 120 million parameters, differently from the order of 175 billion parameters of LLMs such as GPT-3, which could also enable the installation of these systems on-premises, with no API costs and, of course, no privacy concerns.

It is evident that, since in NER multiple text spans in a document may be labeled using the same NE category and in "QA for NER" a single question is used for extracting all the Named Entities related to a category, a Multi-span Extractive Question Answering system is needed.

Extractive QA for NER in the literature

There are few papers in the literature that have attempted a Multi-span Extraction QA approach to NER.

Liu *et al.*, in their paper "QaNER: Prompting Question Answering Models for Few-shot Named Entity Recognition" from 2022, approach the task of NER by using a Single-span Extractive Question Answering model of the type presented in Subsection 2.3.1. In their work they adopt off-the-shelf BERT Large models fine-tuned on SQuAD 2.0. They manage to use a Single-span Extractive QA model by allowing what they call "repeating examples" during the NER to QA conversion process: for the same context and question pair (C, Q), there may be different answers A_i such that the QA instance is (C, Q, A_i), where $i \in 1, 2, \ldots, I$, and I is the number of time a particular entity type e repeats in an NER example $x_{1:n}$. They proceed to fine-tune the single-span EQA model as usual, while, during the QA decoding process, they retrieve the n-best candidate results by computing over the start/end scores of every token. This allows them to identify all the tokens corresponding to the target entity type in a single prompt inference. If two extracted spans overlap, they choose the span with the higher score to avoid possible prediction contradictions. The QA decoding process uses an n-best size of 20, a maximum answer length up to 30 characters, and a threshold that requires the sum of predicted start and end token probability to exceed 1.

3

A Multi-Span Extractive QA Model

Chapter 2 has provided an overview of the Named Entity Recognition and Extractive Question Answering tasks, ultimately explaining why and how a Multi-Span Extractive QA model could be used to approach the task of NER. In this chapter, we will first briefly discuss the strengths and limitations of the models that can be found in the literature and were described in Section 2.3, and then present a model that has been specifically designed and implemented to overcome these limitations.

3.1 Design guidelines

The idea of Liu *et al.* [14], of building up a Multi-span Extractive QA model by using a Single-span Extractive QA model and allowing what they call "repeating examples" in the conversion of the dataset from NER to QA, appears to be simple yet effective in their experiments. Through this implementation, and adopting as model a pre-trained one on SQUAD2 dataset, they are able to retain most of the model's pre-acquired abilities to perform Single-Span Extractive QA, allowing them to get a head-start in few-shot settings. On the other hand, as emerged in our attempts to apply such a method on this research project dataset (BUSTER), this implementation does not perform well when it has to label only those entities that are *linguistically evident from the sentence in which they occur* [4]. It is believed, in fact, that the "repeating examples" technique introduces noise in the learning process. The resulting

[4] The NE label must be discernible within the boundaries of the sentence in which it appears. It must not rely on information from previous or subsequent sentences for its identification; each sentence should contain enough linguistic cues to determine the entity type. See Figure 4.2 for an example.

single-span QA dataset from the conversion will contain multiple instances characterized by having the same questions over the same passages, but with different gold answer spans. When the model is fine-tuned on them it is clear that it may find difficult to learn from multiple single-span QA instances, with different answers to the same question, that, at inference time, it has to return not one, but all of them. Moreover, it will find even harder to learn that it has to return only those spans that are linguistically evident from the sentence, a pattern difficult to be discovered through this formulation.

In contrast to this approach are those of Hu *et al.* [4] and Segal *et al.* [5], which sacrifice the advantages of using a model pre-trained on a huge corpora of single-span QA questions, in favor of implementing a new model architecture oriented towards Multi-Span Extraction. In particular, Segal *et al.* approach the multi-span task not by extending the operating principle of the single-span setting, but by adopting a new mechanism closer to NER sequence labeling through BIO/IO schemes. While in the single-span setting the model is asked to decide where a response begins and ends by using two independent classifiers, in this approach (e.g. when using a IO scheme) the model is instead asked to decide, for each token, whether it should belong to the response or not. Since their multi-span task modeling does not naturally extend from the single-span, it is clear that they lose all the benefits of using a model that was pre-trained to perform single-span extractive QA on a large collection of data, as the start/end classifiers will be replaced with a newly-initialized classifier to reflect the new modeling.

From the above considerations about the strengths and limitations of existing Multi-Span Extractive QA models in the literature, it is clear that if a Multi-Span model could be designed and implemented in such a way as to naturally extend from the Single-Span operating principle, it would benefit from a head start on the task, by leveraging the already pre-acquired capabilities to answer a wide range of single-span questions. The following is a description of such a Multi-Span Extractive QA model, where the focus has been on diverging as little as possible from the classical single-span modeling described in Subsection 2.3.1.

3.2 Model architecture

In this research project, as a starting point for the development of a new model capable of performing Multi-Span Extractive QA, it has been decided to adopt a Single-Span Extractive QA model with the architecture and training losses described in Subsection 2.3.1, having as underlying neural language model the RoBERTa model presented in Subsection 2.2.2, and already fine-tuned on the single-span QA dataset SQUAD2. This model can be easily obtained from the Hugginface repository, where it

can be found under the name "deepset/roberta-base-squad2". It is clear that the modifications to this model that will be presented below can be applied to any other Single-Span Extractive QA (finetuned) model with similar architecture.

3.2.1 Multi-span modeling and training loss

The changes that have been made to the model concern the way the start & end classifiers work on top of the RoBERTa neural language model. In particular, it has been said that the modeling should deviate as little as possible from the single-span one, so that no pre-trained weights are completely discarded or disruptively changed in learning a new modeling.

The key concepts, introduced in Subsection 2.3.1, about Single-Span modeling are:

- the $P_{start}(i|q,x)$ and $P_{end}(j|q,x)$ independence assumption, which allow the model to independently, through two different classifiers S and E, predict where an answer span starts & ends;
- the way the *target starting position* is modeled through a *one-hot vector* over the input tokens; similarly for the *target ending position*;
- the use of the cross-entropy between the *starting target distribution* and *modeled distribution* (obtained by applying a softmax function over the model output logits) as training loss; similarly for the *ending* modeling;
- the fact that constraints like "max answer length" and "end position must not precede start position" are not expressed in the loss function, but only imposed during inference.

The changes implemented for the transition from single-span to multi-span modeling have no impact on the first and last of the key concepts above, while they are a natural extension of the other two.

k-hot vectors modeling

The first change concerns the way the *target starting position*, or rather the *target starting positions* (plural), are modeled. In single-span modeling the start of a gold answer span is encoded through a one-hot vector over the input tokens. A one-hot vector is a vector where all elements are 0, except one element which value is set to 1. In this setting, the position of the token whose value is 1 marks the start of the gold answer span. Since the independence assumption, a second one-hot vector is used to similarly encode the *target ending position*.

It is natural to extend this representation to the multi-span setting by keeping two distinct vectors to preserve the independence assumption, while allowing more than one element in each vector to be

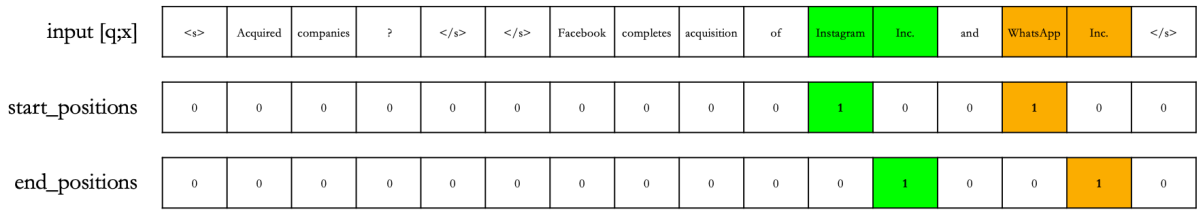


Figure 3.1: An example of k-hot vectors modeling of the start & end positions for two answer spans over the input tokens of a simple question about a short text passage. The input is constructed right-concatenating the text passage to the question, i.e. [q;x]. The special tokens added by the RoBERTa tokenizer are simulated, but a simplified word-level tokenization is used.

set to 1. Each token whose value is set to 1 marks the beginning of an answer span; the same applies for the end positions. Since the number of answer spans to a question can be arbitrarily large k, it has been called "k-hot vectors modeling". Figure 3.1 illustrates an example of k-hot vectors modeling of the start & end positions for two answer spans over the input tokens of a simple question about a short text passage.

Binary Cross Entropy With Logits

A common practice when training a single-span model is the use of the Cross-Entropy loss. This loss compares the probability distribution predicted by the model (obtained by applying a softmax function over the output logits) with the actual target distribution (represented through a one-hot vector); finally, to get a single loss value, the average of the starting and ending position losses is usually taken.

However, this approach encounters challenges when dealing with situations where there can be multiple possible spans for an answer. The issue lies in how the softmax function shapes the probabilities over the input tokens. In cases where there is only one correct span, the softmax function increases the probability at the position that corresponds to the answer's starting or ending point, while reducing the probabilities at other positions. When there are multiple possible spans, using the softmax function in the same way becomes problematic: the function pushes up the probability at only one specific position, but this doesn't align with scenarios where multiple starting and, respectively, ending positions can be valid. Consequently, using the softmax function with cross-entropy loss is not suitable in multi-span setting.

Instead, a more suitable approach is to employ a Binary Cross-Entropy with Logits loss [5], which allows for a more flexible modeling of multiple starting/ending positions.

[5] https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

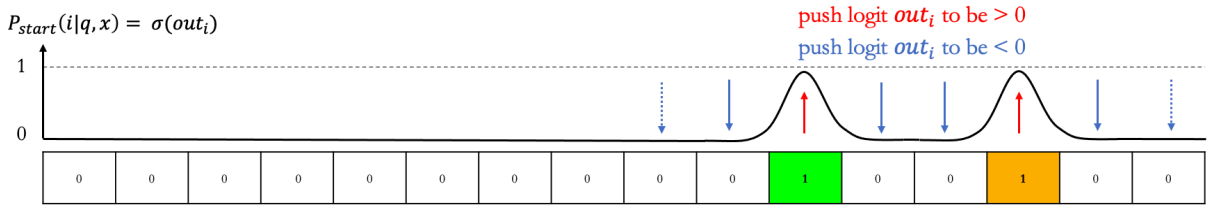


Figure 3.2: The Binary Cross Entropy with Logits loss independently models the probability of each token to be an answer span start (likewise, to be an end position) by pushing the logit value out_i at a starting position to be greater than 0, while pushing the logit values for the tokens that are not a starting position to be less than 0.

The (e.g. *starting*) loss for a [q ; x] instance n is defined as:

$$l_n = -[y_n \cdot \log(\sigma(out_n)) + (1 - y_n) \cdot \log(1 - \sigma(out_n))] \tag{3.1}$$

where y_n is the *k-hot start_positions vector* (target distribution) and out_n is the model's output *start logits vector* (predicted un-normalized distribution). The function is defined similarly for the *ending* loss.

As shown in Figure 3.2, the Sigmoid function σ applied over each output logit $out_{n,i}$ defines a Binomial probability distribution over each token i for it to be, or not to be, a start position; similarly, when modeling the end positions, to be or not to be an end position. The result is that, instead of trying to force a single correct span start to emerge over the input tokens, this loss function independently models the probability of each token to be an answer span start (likewise, to be an end position) by pushing the logit value at a starting position to be greater than 0, while pushing the logit values for the tokens that are not a starting position to be less than 0.

3.2.2 Multi-span extraction algorithm

Consider now, at inference, the model is presented with a question over a passage of text. In this Subsection, the input tokens, obtained by right-concatenating the passage of text to the question (i.e. [*question; passage*]), are considered to be fitting the model input size, so no long-documents handling is required.

In the Single-Span setting the model uses the learned weights to independently predict the start & end probability distributions over the input tokens. Then, the pair with highest score, computed according to Equation 2.6, and satisfying all constraints, is returned. Other spans may be returned by decreasing score order (up to an hyper-parameter *max_answers_returned*) as other potential candi-

dates [6], but only the first one is considered as the only answer.

In Multi-Span setting the model has to do similarly: it has to use the learned weights to independently predict the start & end probability distributions over the input tokens, then return one or more spans of text as answer to the question. It is clear that the extraction algorithm needs to be modified to extract possibly multiple spans from the predicted probability distributions, rather than a single span.

The Multi-Span model by Hu *et al.* [4], presented in Subsection 2.3.2, predicts through a learned Feed-forward Neural Network (see Equation 2.12) the number of answer spans that have to be returned, and then uses a "non maximum suppression" algorithm to extract the number of predicted spans, without overlaps. Another solution proposed by Liu *et al.* [14] consists in fixing the number of maximum spans to be returned to some value n, with the added constraint, relative to their modeling, that the sum of the start & end probabilities for a span candidate has to be greater than 1. Since they use a single-span model to implement a multi-span model through the repeating-examples mechanism, they have decided to take as answer spans all those pairs which probabilities sum is greater than 1, a thresholding value they identified by observing how the modeling is defined. The single-span model predicts two independent probability distributions: within each distribution they consider a token to be a good start/end position if its probability is greater than 0.5; then, to positively evaluate a pair they require the sum to be greater than 1, i.e. the sum of the two thresholding values. This, of course, is an heuristic method, and has its drawbacks: a pair with a high probability starting score (e.g. 0.9) and a very low probability ending score (e.g. 0.1) is considered as good as a pair where both scores are equally (e.g. both 0.5) over the threshold value.

For this research project's multi-span extraction algorithm implementation, we believe that adopting the first solution would cause the overall modeling to diverge from the single-span natural extension we are looking for, while the idea of finding a thresholding heuristic based on how the modeling is defined seems to be a better solution. Following the Liu *et al.* idea, we observe that the Binary Cross Entropy With Logits loss we have adopted pushes the logits over the tokens that are good starting/ending candidates to be greater than 0, while pushes all the other non-candidates to be smaller than 0. Therefore, the heuristic solution that has been defined, is to consider as answer spans all those pairs for which the sum of the start logit and the end logit is greater than 0, namely:

$$logit\_start_i + logit\_end_i > 0 \tag{3.2}$$

This heuristic has the same drawbacks of the Liu *et al.* approach but, as will be shown in the Experi-

[6] These other potential candidates are usually small modifications of the top answer: including/excluding one or more tokens before/after.

ments chapter, it appears to be effective.

Ultimately, as in the single-span setting, there are some constraints that must be satisfied by a pair in order for it to be considered a valid answer span. This constraints can be easily and efficiently imposed on a matrix obtained by computing the "outer sum" between the start & end model output logits. Despite a mathematical definition of "outer sum" does not properly exist, it can be easily defined from slightly modifying the "outer product" operation. Given two vectors $s = [s_0, s_1, \ldots, s_m]$ and $e = [e_0, e_1, \ldots, e_n]$, the outer sum of these vectors is defined as the matrix SE of size $(m+1) \times (n+1)$, where each element SE_{ij} of the matrix is obtained by adding the i-th element of s with the j-th element of e, namely:

$$\begin{bmatrix} s_0 + e_0 & s_0 + e_1 & \cdots & s_0 + e_n \\ s_1 + e_0 & s_1 + e_1 & \cdots & s_1 + e_n \\ \vdots & \vdots & \ddots & \vdots \\ s_m + e_0 & s_m + e_1 & \cdots & s_m + e_n \end{bmatrix} \tag{3.3}$$

Hence, this operation implements the sum of logits between all start & end positions: each element SE_{ij} denotes the sum between the start logit at position i and the end logit at position j, representing a score for the answer span starting at position i and ending at position j, namely (i, j).

To understand how the constraints presented in Subsection 2.3.1 can now be efficiently imposed over such a matrix, consider the start_logits & end_logits vectors in Figure 3.3, where a simplified [<s>; *4-only-passage*] input is considered [7].

END MUST NOT PRECEDE START The first constraint a pair must satisfy to be considered valid is that the end position must not precede the start position, i.e. we have to discard all the elements SE_{ij} where $i > j$. This corresponds to discarding all the elements in the lower triangular matrix, in Figure 3.3 colored in fuchsia.

MAX ANSWER LENGTH It is possible to impose a constraint that discards all answer spans whose length in tokens is greater than L_{max}. This is a hyper-parameter of the model that can be adjusted, for example, according to the domain of the documents and the type of questions. Back to the matrix SE, this constraint is translated in discarding all the elements in the upper triangular matrix with an offset with respect to the diagonal of L_{max}. In Figure 3.3 a $L_{max} = 1$ is assumed and the elements in orange

[7] We consider a [<s>; *4-only-passage*] input to not complicate the example modeling also other special delimiter </s> and relative to the question tokens.

are the ones discarded; therefore, the span starting at token position 1 and ending at token position 4 is invalid, despite its positive score of 4.

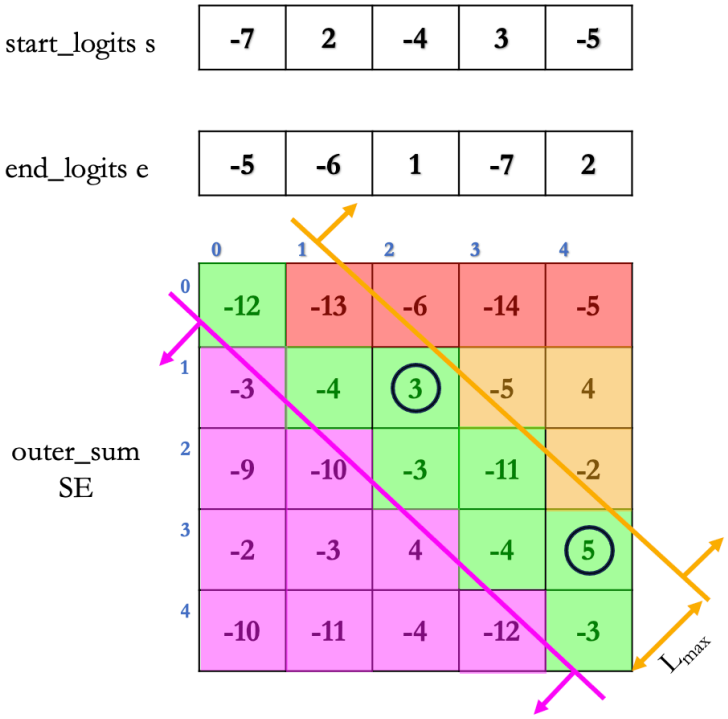


Figure 3.3: A simplified [<s>; *4-only-passage*] input is considered, to not complicate the example modeling also other special delimiter </s> and relative to the question tokens. The model start logits & end logits over such input are reported and the outer sum matrix SE is computed. Some constrains over this matrix are then imposed: the fuchsia cells are discarded by the "end must not precede start" constraint, the orange ones by the "max answer length" constraint, and the red ones by "implementation constraints". Finally, the spans in the green slots which value is > 0 are the extracted answer spans to the question.

IMPLEMENTATION CONSTRAINTS The input tokens in the practical implementation through the RoBERTa model are obtained by tokenizing the text passage concatenated on the right of the question, with the addition of special delimiter tokens (see Subsection 2.2.2). Any span where either the start or end position is on these special tokens or belong to the question are discarded, with the exception of both pointing to the first <s> token, for not answering a question. In Figure 3.3 no special tokens (other than the first <s> token) or relative to the question are represented: the red discarded elements are the ones where the start position is on the <s> special token, while the end position is on any other token.

After all these constraints are imposed, the survived spans which score SE_{ij} is greater than 0 are considered as the answer spans to the question. In the example in Figure 3.3 the answer spans to the question would be: the span starting at position 1 and ending at position 2 with score of 3 and the span starting at position 3 and ending at position 4 with score of 5.

Overlapping spans deletion It may happen that overlapping spans are extracted, e.g. if span (1, 4) would have satisfied the max answer length constraint, it would have overlapped with answer span (1, 2). The extraction algorithm implements a deletion of overlapping spans (where span overlap is considered when one or more tokens belong to more than one answer span), by retaining only the span with the highest score.

Lastly, after the answer spans satisfying all above constraints are extracted and overlaps are resolved, the answer spans are sorted by decreasing score. If the top answer is a no-answer, i.e. the start and end positions both point to the first <s> token, then, regardless of whether there are other extracted answers, they are all deleted and only the no-answer is returned for that question over the passage.

3.2.3 Long-documents handling

In the above Subsection 3.2.2, it was assumed that the input tokens, obtained by right concatenating the text passage to the question (i.e. [*question; passage*]), would fit the model input size, namely the number of input tokens is smaller equal than the model context window size. For the RoBERTa model, this context window size is 512 tokens. In practice, in real-world scenarios, the length of the documents over which the questions are posed is much greater than this input limit. The sliding-window approach presented in Subsection 2.3.1 is usually adopted as solution to this problem, and can be easily adapted to work also in this Multi-Span setting.

Input chunking A long document is divided into overlapping or non-overlapping chunks of text, where a hyper-parameter called "stride" controls the amount of overlapping tokens between two consecutive chunks. Importantly, to each document chunk, before the split, is left-concatenated the question, so that each chunk can be independently processed by the model. A *MAX SEQUENCE LENGTH* hyper-parameter controls the maximum length of a [question; passage] chunk, which may be also set to a value smaller equal than the model context window size. Similarly, a *MAX QUERY LENGTH* hyper-parameter controls the maximum length of a question. The stride value must be smaller than the *MAX SEQUENCE LENGTH - MAX QUERY LENGTH* hyper-parameters difference, otherwise parts of the document will be skipped. A stride value that is too small can cause the

model to cut sentences and miss important information, while a stride value that is too large can cause the model to produce a large number of chunks, resulting in higher inference computational costs.

Model predictions Each [question; passage] chunk can now be independently processed by the model and the answer spans, from each chunk, extracted as described in Subsection 3.2.2. Since the chunks are independent, they can be processed in parallel when the model is deployed on GPUs.

Answers from chunks aggregation Finally, now that a list of answer spans has been extracted from each chunk, they need to be somehow aggregated to produce a list of answers over the entire original long document. This aggregation process requires the deletion of potential overlaps between answer spans extracted from overlapping chunks of text, caused by the sliding window stride. The same overlaps deletion mechanism presented in Subsection 3.2.2 is adopted. Furthermore, the aggregation process must return "no-answer" only if all the extracted answers from all the chunks are "no-answer".

4

Experiments

In Chapter 2, the Named Entity Recognition (NER) task was introduced, along with a brief overview of the classical approaches to it. The operating principles of question-answering systems, both single-span and multi-span, were also described in the same chapter. Eventually, after discussing the motivations and strategies for approaching NER through a QA system, Chapter 3 introduced a novel Multi-Span Extraction QA model, implemented following specific guidelines.

This chapter is dedicated to presenting the experiments conducted to evaluate the effectiveness of the QA-based approach to NER. Firstly, the SQuAD2.0 dataset is introduced, serving as the dataset for pre-training the model in doing single-span QA. Afterwards, the BUSTER dataset is presented. BUSTER is an abbreviation for "BUSiness Transaction Entity Recognition". As the name suggests, it is an industry-oriented Named Entity Recognition dataset in the financial domain with focus on the main actors involved in a business transaction. On this dataset the novel MS-EQA model will be tested and eventually compared with other four models that constitute the baselines of the classical approach to NER. To approach the NER task on BUSTER dataset using a QA system, a conversion of the dataset is needed and, eventually, to make a fair comparison between the approaches, a rigorous metric must be defined.

Lastly, two investigative experiments are conducted: one to evaluate the QA-based approach in scenarios with limited training data and another to test how rapidly the model forgets its capabilities on the SQUAD2 dataset during its fine-tuning on the new BUSTER dataset.

4.1 Datasets

4.1.1 SQuAD2.0

SQuAD2.0, short for "Stanford Question Answering Dataset 2.0", is the dataset most commonly employed to benchmark the performance of Question Answering models, in particular Single-Span QA models. Each instance in the dataset consists of: an ID, a title, a context, a question and a gold-answer. The context is a passage of text from a Wikipedia article and serves as the source of information for the model to answer the associated question. The question paired to each context has been generated by crowdworkers and may, or may not, be answerable. Indeed, what distinguishes SQUAD2.0 from its predecessor is the inclusion of non-answerable questions. These non-answerable questions are designed to be topic relevant and as plausible as possible, challenging the model to reason over the context before returning or not an answer to the question. Finally, each instance is provided with a gold answer that includes both the *span of the text* and its *start* and *end positions in characters* relative to the beginning of the context, as illustrated in Figure 4.1. This is what characterizes the dataset as being Single-Span Extractive QA oriented: all the answers must be a single span of text from the context on which the question is posed, or an empty string if the question is non-answerable. It is worth noting that the development and test sets include multiple spans of text as gold answers, but rather than requiring the model to return them all as would be the case in a multi-span QA scenario, they are used by the provided official evaluation metrics script to evaluate the predicted span against them, keeping the maximum score as final value (more in Subsection 4.2.1). As Figure 4.1 further shows, the multiple gold answers consist of small modifications of approximately the same text span, by including or excluding, one or more words, before its beginning or after its end.

About the dataset statistics: the training set consists of 130319 QA instances from 442 different Wikipedia articles, of which 43498 instances where the question is not answerable; the developments set consists of 11873 QA instances from 35 different Wikipedia articles, of which 5945 non-answerable; the test set (not released) consists of 8862 QA instances from 28 different Wikipedia articles, of which 4332 non-answerable.

More about the dataset can be read from the paper "Know What You Don't Know: Unanswerable Questions for SQuAD" by Rajpurkar et al.[15] in which the dataset is firstly introduced; the training and development sets can be downloaded at this link [8].

[8] https://rajpurkar.github.io/SQuAD-explorer/

context: "Along with advancements in communication, Europe also continued to advance in military technology. European chemists made deadly explosives that could be used in combat, and with innovations in machinery they were able to manufacture improved firearms. By the 1880s, the machine gun had become an effective battlefield weapon. This technology gave European armies an advantage over their opponents, as armies in less-developed countries were still fighting with arrows, swords, and leather shields (e.g. the Zulus in Southern Africa during the Anglo-Zulu War of 1879)."

1) **question:** "What was invented in 1880 that revolutionized warfare?"
gold-answer:
[{"text": "the machine gun", "answer_start": 267},
{"text": "machine gun", "answer_start": 271}]

2) **question:** "What weapons were the Zulus using during the Anglo-Zulu War of 1880?"
gold-answer: " ", "is_impossible": true

Figure 4.1: Two instances from the SQuAD2.0 development set on the same context: the first question is answerable while the second one is not. Although the dataset is oriented towards Single-Span QA, an instance from the development (or test set) may present multiple spans as gold-answers, consisting of small modifications of approximately the same text span by including or excluding one or more words before its beginning or after its end. A gold answer includes the *text span* from the context and its *starting position in characters* from the beginning of it; the *ending position* can be easily computed by adding the length of the text span to the starting position.

4.1.2 BUSTER

BUSTER is an abbreviation for "BUSiness Transaction Entity Recognition". As the name suggests, it is an industry-oriented Named Entity Recognition dataset in the financial domain with focus on the main actors involved in a business transaction. It has been created by the company "Expert.AI" [9] by collecting and annotating real world financial documents from the EDGAR service of the U.S. Securities and Exchange Commission (SEC) [10]. In particular, they downloaded about 120,000 EX-99.1 disclosure documents. An EX-99.1 document is a financial document filed by publicly traded companies to provide a succinct summary of significant corporate events or operations, such as acquisitions, bankruptcy filings, director resignations, or changes in the fiscal year; this document is essential for informing investors and the public about crucial developments that may impact the company's financial health and overall business strategy. After parsing, cleaning and removing any empty, too short or too long documents, they selected the relevant ones using transaction-related keywords (acquisition, acquire, ownership, etc.) obtaining a final raw dataset of about 10,000 text files. They then proceeded to annotate 3779 of these documents using the 7 Named Entity categories in Table 4.1 in a "double-blind" manual procedure: two annotators were trained on financial transactions topic and provided

[9] The company where the internship and research project have been carried out, https://expert.ai/
[10] Electronic Data Gathering, Analysis, and Retrieval system, https://www.sec.gov/edgar

with specific guidelines to follow; a shared set of documents was tagged by both annotators to be used to evaluate the quality output of the annotation process; and finally, the shared set was fixed by a third annotator manually resolving any labelling conflict between the two annotators.

Tag Family	NE Tag Name	Definition
Parties	BUYING_COMPANY	The company which is acquiring the target.
	SELLING_COMPANY	The company which is selling the target.
	ACQUIRED_COMPANY	The company target of the transaction.
Advisors	LEGAL_CONSULTING_COMPANY	A law firm providing advice on the transaction, such as: government regulation, litigation, anti-trust, structured finance, tax etc.
	GENERIC_CONSULTING_COMPANY	A general firm providing any other type of advice, such as: financial, accountability, due diligence, etc.
Generic_Info	ANNUAL_REVENUES	The past or present annual revenues of any company or asset involved in the transaction.

Table 4.1: BUSTER's tag-set of Named Entities and their associated definitions. The NEs are grouped in three families of tags: (a) *Parties* which groups tags used to identify the entities directly involved in the transaction; (b) *Advisors* which groups tags identifying any external facilitator and advisor of the transaction; (c) *Generic_Info* which identifies tags reporting any information about the transaction.

EX-99.1 - August 28, 2007 -- Homeland Security Capital Corporation (OTC Bulletin Board: HOMS), a company focused on acquiring, developing and consolidating homeland security related businesses, announced today that it has completed the sale of its majority owned subsidiary, Security Holding Corp. (SHC), to Vuance Ltd. (formerly SuperCom Ltd.) a provider of incident management and active RFID solutions, for approximately $ 5.1 million in Vuance stock.

Security Holding Corp., which was formed by HOMS in August 2006, wholly owns two operating companies SecurityInc, LLC and AutoAccess ID Security Solutions, Inc. About Vuance Ltd., Vuance Ltd. provides innovative incident management, RFID and credentialing solutions to the public safety, commercial and government sectors.

BUYING_COMPANY, SELLING_COMPANY, ACQUIRED_COMPANY, ...

Figure 4.2: Passage of text extracted from BUSTER where only "linguistically evident from the sentence in which they occur" Named Entities are tagged. This can be seen in the second paragraph where, although it is known from the previous sentences that "Security Holding Corp." and "Vuance Ltd." are the ACQUIRED_COMPANY and BUYING_COMPANY respectively, they are not tagged because it is not clear from the sentence in which they occur what their role is.

The guidelines provided to the annotators included the definitions for each tag name in Table 4.1 and the instruction to label only those which are "linguistically evident from the sentence in which they occur", namely a NE label must be discernible within the boundaries of the sentence in which it

		Gold						Silver
		fold 1	*fold 2*	*fold 3*	*fold 4*	*fold 5*	*Total*	*Total*
	N. Docs	753	759	758	755	754	3779	6196
	N. Tokens	685K	680K	687K	697K	688K	3437K	5647K
	N. Annotations	4119	4267	4100	4103	4163	20752	33272
Parties	BUYING_COMPANY	1734	1800	1721	1707	1717	8679	14558
	SELLING_COMPANY	460	447	456	426	439	2228	4016
	ACQUIRED_COMPANY	1399	1473	1362	1430	1447	7111	9879
	Total	3593	3720	3539	3563	3603	18018	28453
Advisors	LEGAL_CONSULTING_COMPANY	142	132	152	146	153	721	1176
	GENERIC_CONSULTING_COMPANY	256	267	261	248	256	1279	2210
	Total	398	399	413	394	409	2013	3545
Generic_Info	ANNUAL_REVENUES	128	148	148	146	151	721	1274
	Total	128	148	148	146	151	696	1274

Table 4.2: The statistics about the 5 Gold folds (and Silver) data.

appears and it must not rely on information from previous or subsequent sentences for its identification. In Figure 4.2, a passage of text extracted from BUSTER is reported, showing more clearly what needs to be tagged and what does not.

The final BUSTER dataset consists of 3779 manually labelled documents, which fall under the "Gold" BUSTER partition. The "Silver" partition consists of the remaining 6196 crawled documents, which has been instead automatically labelled using the RoBERTa baseline model described later in Subsection 4.2.2. The data has been randomly split into 5 folds to allow a statistically robust benchmark through the standard k-fold cross-validation approach. The statistics about the folds in the Gold and Silver BUSTER partitions are reported in Table 4.2, while the five permutations of folds used for the 5-fold cross validation are illustrated in Table 4.3.

	Folds		
	Training	Validation	Test
Run_1	1, 2, 3	4	5
Run_2	5, 1, 2	3	4
Run_3	4, 5, 1	2	3
Run_4	3, 4, 5	1	2
Run_5	2, 3, 4	5	1

Table 4.3: The five permutations of folds used for 5-fold cross-validation.

4.1.3 BUSTER for Question Answering

Approaching the NER task using a Multi-Span Extractive QA model requires the BUSTER dataset to have a QA format similar to SQuaD2.0: each instance should consist of a context, a question and a list of gold-answers. This conversion from BIO labeling to QA format has been accomplished through the following operations:

1. For each document, a data dictionary is obtained by parsing the BIO sequence and retrieving, for each Named Entity tag as key, a list of entities as values. Each item in the list has the same structure as a gold-answer presented for the SQuAD2.0 dataset: a *span of text*, a *start position* and an *end position* in characters relative to the start of the document. Importantly, this time the items in the list are not in disjunction (OR), but they must all be returned by the model when answering the question.

2. Now, for each Named Entity tag, a question to drive the extraction is defined. Each question has been formulated by attempting to inject the corresponding NE definition from Table 4.1 into the question, to give the model a head start in understanding what to label and what not to label. The question associated with each NE tag is reported in Table 4.4.

3. Finally, on each document, every question is posed, effectively performing a Cartesian product of questions and documents. This results in the generation of a QA dataset with a cardinality equal to the total number of documents multiplied by the total number of Named Entities, namely $\#DOCs \times \#NEs$.

Each instance in *BUSTER_QA* consists of: a document as *context*, a *question* formulated to drive the extraction of all the entities of a desired category and a *list of entities as gold-answers* to be all extracted.

Now, each instance can be processed as described in Chapter 3, using k-hot vectors modeling and, in particular, the long-document handling technique presented in Subsection 3.2.3. The latter will further increase the number of training instances in the dataset.

NE Tag	Question
BUYING_COMPANY	Which is the buying company?
SELLING_COMPANY	Who is selling something?
ACQUIRED_COMPANY	Which is the acquired company?
LEGAL_CONSULTING_COMPANY	Which is a law firm providing legal advice on the transaction?
GENERIC_CONSULTING_COMPANY	Which is a general firm providing non legal advice on the transaction?
ANNUAL_REVENUES	What are the annual revenues?

Table 4.4: The question associated with each NE tag.

4.2 Experimental Setup

4.2.1 Metrics

In the following, various metrics are presented. The first one concerns the classical NER evaluation, the second one is for the evaluation of a single-span extractive QA model on SQUAD2.0, and finally a metric alignment procedure for the evaluation of the multi-span EQA model with classical NER metrics is presented.

NER

In order to evaluate a model on a Named Entity Recognition task, the traditional metrics adopted are *Precision*, *Recall* and $F1$. In the experiments reported in this research project an entity is considered correctly predicted by the model, and thus considered a True Positive (TP), if and only if all of its constituent tokens are well identified, and no additional tokens belong to the entity. This means that if an entity is encoded through the BIO labeling scheme in the sequence [..., O, B-BUYING, I-BUYING, I-BUYING, O, ...], the model has to predict exactly these token types. Any added or missed token of type BUYING will cause the prediction to be counted as both a False Positive (FP) and a False Negative (FN): a FP because the predicted entity does not match the ground truth; a FN because the entity is not correctly identified in all its constituent tokens and is therefore considered missed. Similarly a FP and FN are counted when the model predicts the wrong token types, e.g. ACQUIRED instead of BUYING. For the sake of completeness, True Negatives (TNs) measure the ability of the model to correctly identify non-entity tokens, that are all "O" token types when considering the BIO-labeling context. However, they do not provide much information as the primary focus in NER is on TPs, FPs and FNs.

Now that we have defined what TPs, FPs, FNs are and how they are computed, we can easier define the metrics *Precision*, *Recall* and $F1$.

Precision is evaluated through the formula

$$Precision = \frac{TPs}{TPs + FPs} \tag{4.1}$$

and it measures the accuracy of the model's positive predictions, namely how many of the predicted entities are correct.

Recall is instead computed through the formula

$$Recall = \frac{TPs}{TPs + FNs} \tag{4.2}$$

and it measures the model's ability to avoid missing positive instances.

In other tasks and domains, one might wish to have a higher Precision score than Recall, or vice versa, but in NER one usually wishes to have both metrics equally high. A metric that combines both is the $F1$ score, computed as the harmonic mean between *Precision* and *Recall*:

$$F1 = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall} \tag{4.3}$$

For each Named Entity category, the model's TPs, FPs and FNs can be computed, allowing the model's $Precision_i$, $Recall_i$ and $F1_i$ to be evaluated independently for each category $i \in$ {BUYING _COMPANY, ACQUIRED_COMPANY, SELLING_COMPANY, . . .}. By averaging these scores across all NE categories the Macro scores are computed, namely: $M-Precision$, $M-Recall$, $M-F1$.

Instead, by summing up the TPs, FPs and FNs values across all categories and using the above *Precision*, *Recall* and $F1$ formulas the micro scores are computed, namely: $\mu-Precision$, $\mu-Recall$, $\mu-F1$.

Micro-scores are useful when one wants to assess the overall performance of a multi-label classifier (as in the case of NER), giving equal weight to each NER label. They are particularly valuable when the class distribution is unbalanced, as they account for the unbalance by considering all classes together. Macro scores, on the other hand, provide a way of assessing performance in each class individually and then averaging these scores. They are valuable if one wishes to understand how well the model performs across all classes without giving more weight to any particular NE category. If the model performs poorly in a class with small support, the M-scores will be strongly affected, while the μ-scores not.

QA

Two commonly used metrics to evaluate a Question Answering model are *Exact Match* (*EM*) and $F1$. In a very helpful way, the SQuAD2.0 dataset is provided with an "official evaluation script". However, this script can only be used to evaluate a Single-Span QA model, due to the nature of the SQuAD2.0 dataset, as described in the previous section. Nevertheless, its metrics are here described as they will be used in the experiments to access the ability of the newly implemented Multi-Span Extractive QA model to also answer single-span questions such as those in SQuAD2.0.

For both the *EM* and *F*1 computations, the script first applies some normalization steps to both the predicted answer and the gold answer being compared. These normalization steps include lower-casing the string, removing articles, punctuation and any extra white-spacing.

The *EM* score between a predicted answer and a gold answer is then evaluated as 1 if the sequence of words matches exactly at every position, otherwise 0. *EM* assesses whether the model's answer is an exact match to the ground truth answer and is a strict metric, don't giving any partial credit for partially correct answers. Instead, *F*1 should be seen as a measure of the overlap between the predicted and the gold answer, using the classic formula where *Precision* is the fraction of predicted words that are also in the gold answer, and *Recall* is the fraction of words in the gold answer that were predicted.

Remembering that the SQuAD2.0 development and test sets include a list of gold answers against which to test the predicted answer, the *EM* and *F*1 scores for an instance prediction are computed by taking the maximum score between the predicted answer and one of the gold answers in the list.

Finally, the scores are averaged across all instances to get the overall *EM* and *F*1. These scores are also computed considering only those instances which are answerable, namely computing *HasAns_exact* and *HasAns_f*1; and only those instances which are non-answerable, namely computing *NoAns_exact* and *NoAns_f*1.

Multi-Span QA for NER

In the following experiments, in order to evaluate the performance of the Multi-Span model on NER task, the above official SQUAD2.0 evaluation script cannot be used for two reasons: the list of gold answers is not in disjunction (OR), but they must all be predicted by the model and, more importantly, the evaluation must be defined to be NER oriented, in order to allow a direct comparison with NER models, considered as "the baselines" in this research project.

TPs, FPs, FNs and TNs are computed to match their definition in classic NER metrics. By posing a question relative to a NE category on a document, a TP is any predicted text answer where both the start and end positions in characters match a gold answer entity span. Indeed, this satisfies the NER's TP definition "if and only if all of its constituent tokens are well identified, and no additional tokens belong to the entity". Otherwise, it is considered as a False Positive (FP). It is worth noting that a FP is not here straightforward counted also as a FN. Instead, the number of FNs in a document, relative to a NE category, is the number of gold answer entities that have not been hit by any predicted answer. Finally, a True Negative (TN) is considered when the the model returns an empty answer and no entities of that category are indeed present in the document.

Once the TPs, FPs and FNs for each NE category have been computed by applying the associated question to all the document in the dataset, the $Precision_i$, $Recall_i$ and $F1_i$ for each category

$i \in$ {BUYING_COMPANY, ACQUIRED_COMPANY, SELLING_COMPANY, . . .} can be calculated through Equation 4.1, Equation 4.2 and Equation 4.3, respectively. The scores can then be averaged across the NE categories to computed the Macro scores: $M-Precision$, $M-Recall$, $M-F1$. Instead, just like in NER, by summing up the TPs, FPs and FNs values across all categories and using the same equations the micro scores are computed, namely: $\mu-Precision$, $\mu-Recall$, $\mu-F1$.

4.2.2 Models

In order to access the performance of the proposed Multi-Span Extractive QA model for the task of Named Entity Recognition, we compare its performance on the BUSTER dataset against 4 baseline models. These models are presented in the BUSTER paper and approach the NER task in the classical way by considering it as a Sequence Labeling Problem, as explained in Subsection 2.1.2.

Baselines

Four transformer-based models are considered as NER baselines for the BUSTER dataset. All the models are based on the architecture presented in Subsection 2.1.2, with a Transformer Neural Language Model as underlying Context-Encoder and a "MLP + softmax layer" as Tag-Decoder. The Tag-Decoder performs a multi-label classification at each token position between 13 token types ("B-" and "I-" for each NE category + 1 for "O"). All models are trained by minimizing the sum of the cross-entropies at each token position. During inference, the token type with the highest probability is chosen independently at each position.

The 4 Neural Language Model that has been adopted as underlying Context-Encoder are: BERT, RoBERTa, SEC-BERT and Longformer.

- BERT and RoBERTa models have already been described in detail in Subsection 2.2.2.
- SEC-BERT is a NLM with same BERT architecture and pre-training objectives, but pre-trained on a financial corpora consisting of 260,773 "10-K" filings from 1993-2019, publicly available at U.S. Securities and Exchange Commission (SEC). This pre-training on financial documents should give the model a head start in understanding financial reports such as those in the BUSTER dataset. The model is downloadable from the Hugging Face repository [11].
- Longformer is a specialized transformer-based language model designed to efficiently handle long sequences of text, making it ideal for tasks involving lengthy documents like legal documents. Unlike traditional transformers with fixed sequence lengths, Longformers can process much longer sequences, often up to 4,096 tokens or more, without truncation. They achieve

[11] https://huggingface.co/nlpaueb/sec-bert-base

this through innovations like "global attention", which allows them to consider the entire document, and "sparse attention patterns" that focus on relevant tokens while skipping irrelevant ones. Longformers are highly adaptable and can be fine-tuned for specific tasks, making them valuable tools for natural language understanding in contexts where document length is a critical factor. The model is downloadable from the Hugging Face repository [12].

In particular, Longformer has been identified and adopted as a potential solution to one of the difficulties posed by the BUSTER dataset: the length of its documents. The vast majority of documents in BUSTER has more than 500 words, which typically exceeds the maximum sequence length that NLMs such as BERT, RoBERTa and SEC-BERT can take in input. In the experiments, therefore, no "long-documents handling" is required when using the Longformer model, while some processing is required for the other three: documents are segmented into contiguous chunks that fit the model context-window (in these models of 512 tokens). Importantly, the chunking is done in such a way that no token is truncated and each chunk sequence is filled as much as possible.

4.2.3 Training details

Details about the training procedures and models' hyper-parameters can be found here.

Multi-Span QA model The pre-trained Single-Span model, on which the Multi-Span model is based as described in Chapter 3, has been downloaded from the Huggin Face repository, where it can be found under the name "deepset/roberta-base-squad2" [13]. Since the vast majority of documents in BUSTER has more than 512 tokens when tokenized, which exceeds the maximum sequence length the model can take in input, the "long-documents handling" technique presented in Subsection 3.2.3 has been adopted: as hyper-parameters, a *maximum question length* of 32, a *maximum sequence length* (question + context + special tokens) of 384 and a *stride* of 128 are used. The model has been fine-tuned using AdamW as *optimizer*, with a *learning rate* of 2×10^{-5}, a *linear scheduler* with a number of warm-up steps 5% the number of training steps and a *batch size* of 32. A maximum number of training *epochs* of 20 was set, using *early-stopping* with a patience of 5, maximizing the Macro-F1 metric, evaluating on the validation set every 250 batch steps. For the Multi-span extraction algorithm a *maximum answer length* hyper-parameter of 20 tokens has been used after evaluating the distribution of the entity lengths in Fold 1.

[12] https://huggingface.co/allenai/longformer-base-4096
[13] https://huggingface.co/deepset/roberta-base-squad2

Baselines The 4 pre-trained Neural Language Models have been downloaded from the Hugging Face repository (links in the footnotes above). Furthermore, the Hugging Face tools have been used to fine-tune these models on BUSTER for the task of Named Entity Recognition using the specially designed NE categories tag-set. About the training parameters: the *Trainer state* from each model has been resumed; a *batch size* of 32 has been used for the three models, while a smaller batch size of 2 was required for the Longformer, which takes in input not chunked documents of greater sequence length (*max sequence length* of 4096, instead of 512); a maximum number of training *epochs* of 15 was set, using *early-stopping* with a patience of 5, maximizing the Macro-F1 metric, evaluating on the validation set every 300 batch steps.

Importantly, to provide a more stable estimate of all models performance, "k-fold cross-validation" technique is used. For each model type, 5 models were trained each time using 3 folds from the Gold BUSTER partition as training set, 1 fold as validation set and 1 as test set, as illustrated in Table 4.3. This approach enables the quantification of the standard deviation for each metric across all test folds, providing valuable insights into the model's robustness across varying data distributions.

4.3 Results

The performance of each model, for the task of Named Entity Recognition on BUSTER dataset, is reported in Table 4.5. In particular, the micro (μ-) and Macro (M-) *Precision*, *Recall* and $F1$ metrics of the four baselines and the Multi-Span Exctractive QA model are compared.

Across the baselines, RoBERTa emerges as the best performing model, closely followed by the Longformer model, which confirms its strength in handling long documents. Instead, BERT appears to be the worst performing model. Interestingly, SEC-BERT, which has the same architecture as BERT, performs relatively well, just few points below the best performing model RoBERTa, which also has the same architecture. This shows the importance of the pre-training of a NLM when it is later fine-tuned on a specific task and domain. The careful choice of an NLM, such as SEC-BERT, pre-trained on documents from the same domain, in this case financial documents, can improve the performance for the same architecture. A strong and efficient pre-training confirms the general superiority of the RoBERTa model over BERT.

The Multi-Span Extractive QA model presented in this research project exhibits significant performance in the Named Entity Recognition task, ranking first on the podium, tied with the best-performing baseline model. Curiously, due to the fact that both the best performing baseline model and the MS-EQA model adopt the same RoBERTa Neural Language Model as Context-Encoder, it

Model	μ-*Precision*	μ-*Recall*	μ-*F*1	*M*-*Precision*	*M*-*Recall*	*M*-*F*1
BERT	61.16 ± 1.65	67.42 ± 2.72	64.06 ± 0.90	55.12 ± 1.75	66.60 ± 2.79	59.80 ± 1.23
SEC-BERT	66.76 ± 0.74	74.18 ± 1.99	70.28 ± 0.90	70.30 ± 0.96	78.10 ± 1.82	73.98 ± 1.14
RoBERTa	**69.84 ± 1.41**	**75.08 ± 1.42**	**72.34 ± 0.39**	**72.38 ± 0.64**	**79.34 ± 1.17**	**75.58 ± 0.66**
Longformer	69.28 ± 2.71	73.40 ± 1.31	71.24 ± 1.34	70.02 ± 3.27	77.34 ± 1.49	73.30 ± 2.25
MS-EQA	**70.45 ± 3.76**	**77.92 ± 1.99**	**73.89 ± 1.40**	**73.98 ± 3.38**	**79.26 ± 2.33**	**76.37 ± 1.61**

Table 4.5: Micro (μ-) and macro (M-) scores of the four baseline models and the Multi-Span Extractive QA model on BUSTER; averages and standard deviations across permutations evaluated using 5-Fold Cross Validation.

becomes possible to directly compare the distinct approaches to the NER task: one treating it as a Sequence Labeling Problem, while the second approach, outlined in this research project, through the use of a Question Answering system. In terms of micro and Macro scores, the values between the RoBERTa baseline model and the MS-EQA model are equivalent, with the latter model generally performing slightly better. However, the performance of the MS-EQA model is characterized by a higher standard deviation, which means that the model is somewhat less robust across different data distributions.

Figure 4.3 compares the *Macro F*1 scores of the four baseline models and the Multi-Span Extractive QA model on each BUSTER NE category. All models show equivalent capabilities, except BERT which performs poorly on the less represented classes, namely those NE categories with smaller support. This can be seen from both Figure 4.3 and Table 4.5, where the macro scores are significantly affected by the poor performance of the model on these less represented classes when compared to the micro scores.

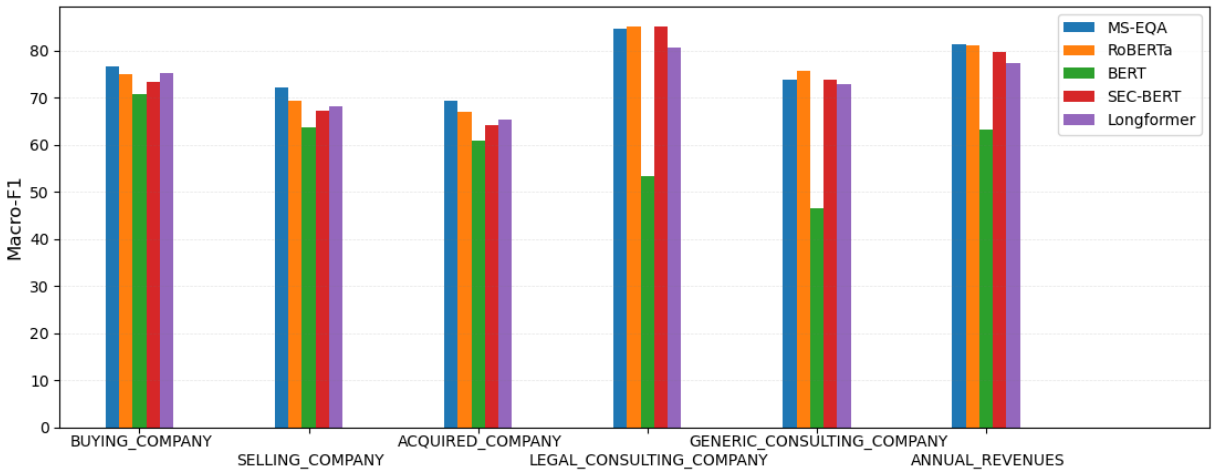


Figure 4.3: Macro-F1 scores of the four baseline models and the Multi-Span Extractive QA model on each BUSTER NE category.

Table 4.6 reports the *Precision*, *Recall* and $F1$ scores of the Multi-Span Extractive QA model on each BUSTER Named Entity category. Across all NE categories, the *Precision* and *Recall* metrics are affected by a higher standard deviation with respect to the $F1$ metric. This can be explained by the fact that the MS-EQA model is more flexible in switching between higher precision and lower recall, or vice versa, while the overall labelling result, captured by the $F1$ metric, nevertheless shows a stable final performance across all k-fold permutations. About each NE category, BUYING_COMPANY appears to be the most stable and probably easier to tag category across all permutations. Errors in this category may be caused by the model returning also spans of text which do not satisfy the requirement of being "linguistically evident from the sentence in which they occur". However, this may not only be due to the model: noise in the dataset from the double-blinded manual labeling procedure also contributes to some degradation in performance. This applies not only to this NE category, but also to all other NE categories. SELLING_COMPANY and ACQUIRED_COMPANY appear to be two difficult categories to label, as they are strongly related and often difficult even for human annotators to disambiguate. LEGAL and GENERIC CONSULTING_COMPANY are somewhat difficult for the model: in the first case, the model suffers from a high standard deviation across different permutations; in the second case, the model has a lower $F1$ value compared to the first category in the Advisors family. This may be due to the model's difficulty in distinguishing between legal and generic consultancies, as well as its difficulty in distinguishing generic consulting companies from CONSULTANTS, a previously existing BUSTER NE category (removed due to its too small support), with a very similar definition to GENERIC_CONSULTING_COMPANY, but related to people. With regard to ANNUAL_REVENUES, the model performs relatively well on this category, with an acceptable standard deviation.

		Precision	*Recall*	$F1$
Parties	BUYING_COMPANY	72.74 ± 3.20	81.28 ± 1.20	76.68 ± 1.40
	SELLING_COMPANY	70.66 ± 4.04	73.82 ± 2.33	72.12 ± 2.12
	ACQUIRED_COMPANY	65.64 ± 5.17	73.90 ± 2.98	69.30 ± 1.80
Advisors	LEGAL_CONSULTING_COMPANY	85.40 ± 4.61	84.22 ± 4.07	84.80 ± 4.22
	GENERIC_CONSULTING_COMPANY	71.46 ± 2.89	76.74 ± 1.41	73.94 ± 1.50
Info	ANNUAL_REVENUES	78.00 ± 5.27	85.60 ± 5.78	81.34 ± 2.62

Table 4.6: Tag-wise precision, recall and F1-score values obtained with the Multi-Span Extractive QA model, using 5-Fold Cross Validation.

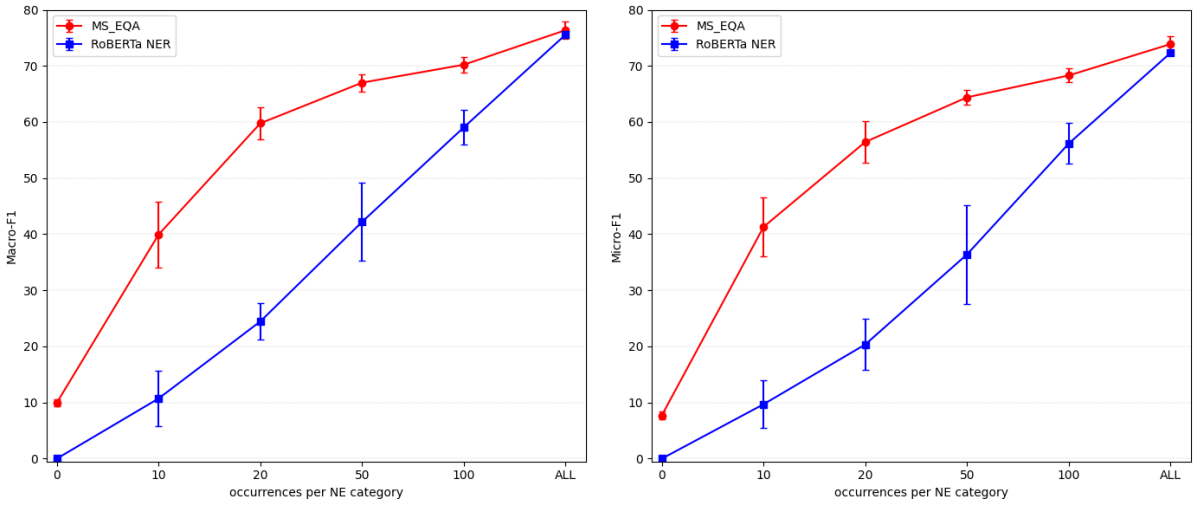
4.4 Exploratory investigations

4.4.1 Learning efficiency on reduced training data

The Multi-Span Extractive QA model achieves comparable performance to the leading NER baseline. However, this tie occurs when the two approaches are compared by exploiting the full available training set, which in BUSTER is of considerable size. To explore the possibility of the proposed question answering approach being more effective than the classical NER strategy in some scenarios, its ability to learn efficiently on a reduced training set is tested here.

Indeed, a key point in the presentation of this QA-approach to NER, as has been emphasized throughout this thesis, is its theorized ability to learn efficiently on a reduced training set, thanks both to the possibility of injecting what to label in the question formulation and to the design guidelines followed for the model implementation, which allow this model to retain all its pre-acquired abilities to perform single-span QA from the SQuAD2 dataset. On the contrary, a classical approach to NER requires the model to infer what to label directly from the training data, usually without any external hints or definitions about the NE categories, and to start its fine-tuning from random classification parameters each time a new set of labels is defined.

The experiment that has been crafted to access the above MS-EQA's theorized ability consists in comparing the $F1$ scores of this model against the leading NER baseline RoBERTa when both fine-tuned on reduced training sets of varying sizes. In particular, each reduced training set is constructed by selecting a subset of documents such that, for each NE category, the sum of occurrences across the selected ones is equal to some chosen value t. This task is cast as an Integer Linear Programming problem, where a binary variable $\{0, 1\}$ is defined for each document in the complete training set. Then, by using an ILP solver, the subset of documents of minimum cardinality such that the constraint above is full-filled is obtained by taking all the documents to which the solver has assigned value 1. Different values for t are explored: 0, 10, 20, 50, 100. Again, to access the models robustness, k-fold cross-validation is adopted. For each permutation, the reduced training set with t occurrences per NE category is obtained as described above, the validation set is dropped, while the test set is left complete for fair comparison between all runs and also with the models trained on the complete training set. The validation set is dropped since no early-stopping is used, but both the model are trained for the same amount of batch steps. In particular, it has been decided to fine-tune the MS-EQA model for 10 epochs and the RoBERTa baseline for a number of epochs such that the number of training batch steps is equal for both approaches, namely they both see their training instances the same amount of times. In fact, it is important to remember that the number of training instances for the QA approach



(a) *Macro* -$F1$ **(b)** *micro* -$F1$

Figure 4.4: $F1$ comparison between the Multi-Span Extractive QA based approach and the best performing NER baseline (RoBERTa) when fine-tuned on reduced training sets of varying number of occurrences per NE category.

is grown by the number of questions defined for each NE category, thus the number of training instances is increased for the same amount of selected training documents.

Figure 4.4 compares the *Macro-* and *micro-* $F1$ scores between the MS-EQA model and the RoBERTa NER baseline when fine-tuned and tested in the reduced data settings described above. Particularly noteworthy is the performance of the MS-EQA model in "zero-shot" setting, i.e. when applied "out-of-the-box" without any examples provided for fine-tuning [14]. With a *Macro-*$F1$ score around 10, it beats the performance of the NER counterpart, which, of course, is 0, since fine-tuning is mandatory in the classical NER approach every time a new set of NE categories is defined. A not greater out-of-the-box $F1$ score is likely due to the BUSTER's financial domain type, which differs from the articles present in SQuAD2.0. Additionally, the model has been pre-trained on a corpus of single-span questions, thus it is currently unable, out-of-the-box, to label multiple occurrences for each NE category.

Overall, the question-answering approach to NER exhibits a steeper learning curve, consistently outperforming the NER baseline counterpart in each experimental setting. This suggests that, although the question-answering approach achieves performance comparable to the traditional NER framework, without introducing any specific enhancements when a considerable in size training set is exploited, it proves to be a valuable tool in scenarios with limited training data.

[14] In order to apply the model out-of-the-box on BUSTER with the novel multi-span extraction algorithm some fine-tuning steps on SQuAD2.0 dataset are required. This is necessary to teach the model to answer the SQuAD2.0 single-span questions with the novel answers emerging algorithm, and not by manually picking the top scoring span.

4.4.2 Forgetting

A second investigation experiment is conducted to assess how rapidly the Multi-Span Extractive QA model loses its ability in answering general-topic single-span questions from SQuAD2.0 while fine-tuning it on the new BUSTER dataset. The BUSTER dataset, differently from SQuAD2.0, consists of a limited number (one for each NE category to extract) of multi-span questions related to financial domain documents only. Consequently, the experiment tests the extent to which the model retains its ability to answer various types of single-span questions across different context domains.

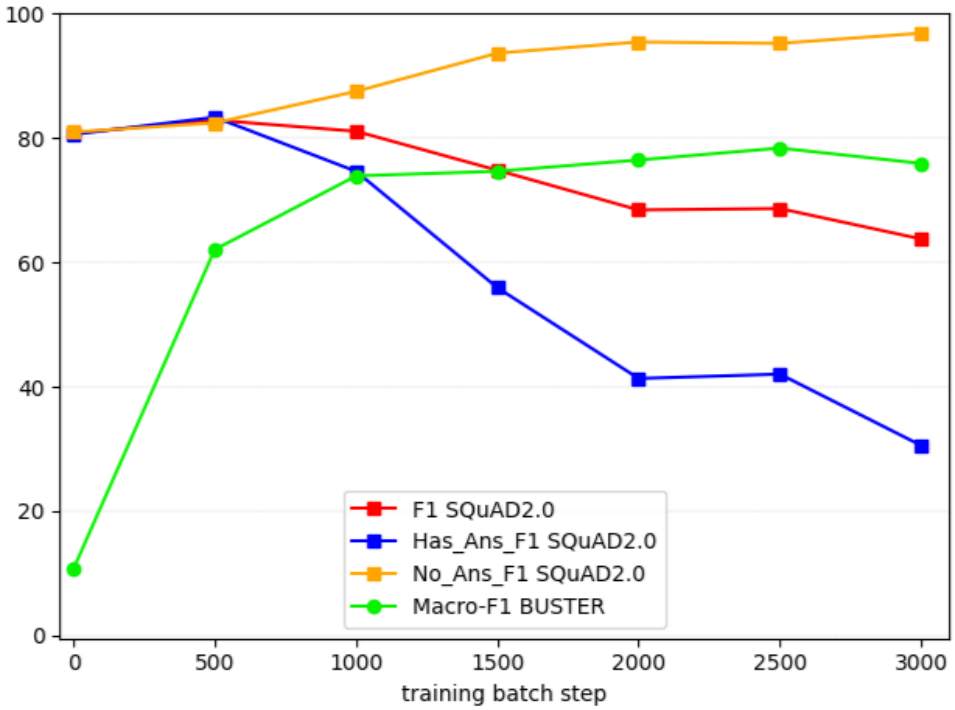


Figure 4.5: SQuAD2.0 and BUSTER score trajectories during fine-tuning of the MS-EQA model on the default BUSTER permutation.

Figure 4.5 illustrates the trajectory of four significant scores during the fine-tuning process of the MS-EQA model on the default BUSTER permutation. Notably, as the *Macro-F*1 score on BUSTER dataset increases, there is a corresponding decrease in the *F*1 score on SQuAD2.0. While this decrease may appear relatively modest, when considering only the *F*1 score for answerable questions (*Has_Ans_F*1), it becomes evident that the model is experiencing a substantial decline in its performance on the SQuAD2.0 task; in other words, the model is forgetting.

5

Conclusions and Future Work

This chapter summarizes the main contributions of this thesis and suggests possible directions for future research.

The motivations for this thesis approach have been presented by analyzing the potential disadvantages and limitations of a Generative QA approach to NER through in-context learning on Large Language Models. In particular, the proposed Extractive QA approach would tackle the hallucinations problem, the output-formatting requirements, the long-documents handling and other concerns about hardware, APIs costs and privacy.

On the basis of the above considerations, a new Multi-Span Extractive QA model has been proposed. The model has been implemented according to specific design guidelines that make it a natural extension of the Single-Span QA operating principle.

In order to fairly evaluate the proposed QA-based approach with classical NER systems, the process for converting a NER dataset in BIO format into a QA format, but still NER-oriented, has been presented together with a metric alignment procedure. The proposed model performance has been evaluated on the BUSTER dataset and compared with four other models that constitute the baselines for the classical NER approach. For the same underlying NLM model RoBERTa, the QA-based approach achieves comparable performance to the leading NER baseline.

To explore the potential of the proposed question answering approach with respect to traditional NER strategies, its ability to learn efficiently with limited training data has been tested. Adaptation of pre-trained QA models to downstream tasks proves to be simpler to achieve and much more effective

on a low-resource scenario. With only 20 examples per class, a QA model pre-trained on SQuAD-2.0 achieves acceptable performance on BUSTER, with $F1$ scores close to 60%, almost three times higher than the NER baseline counterpart.

Finally, it has been investigated how rapidly the QA model forgets past information, learnt in SQuAD-2.0, during fine-tuning on novel data.

Future work is expected to focus on this direction, trying to develop a technique to make the model perform Continual Learning, namely learning on a sequence of NER datasets, each characterized by its style and set of questions associated to each NE category, without forgetting its ability on none of these. Indeed, the primary objective will be to ensure that the model can adapt to new NER datasets while retaining its capability to correctly respond to questions from all previously encountered datasets, thereby preventing any loss of knowledge acquired along the way. This will theoretically give the model an even greater head start in scenarios with reduced training data, as it will already have been exposed to many different types of questions to extract NE occurrences of different categories, acting as a pre-training for the QA approach to NER.

Publication Latest news is that the BUSTER dataset on which the experiments have been conducted has just been accepted for publication at the EMNLP 2023 Industry Track Conference. The paper presents the statistics of the dataset, and a description of the double-blind manual labelling process used by Expert.AI to create the dataset. Personal contribution from my side has been put into the baselines experiments and some refining of wrongly labelled entities with a semi-automatic procedure during my internship at Expert.AI. The paper will be published under the title *BUSTER: a BUSiness Transaction Entity Recognition dataset*, with authors *Andrea Zugarini, Leonardo Rigutini, Andrew Zamai* & *Marco Ernandes*. The dataset will be soon publicly available.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[3] G. Satta, "Natural language processing course," 2023, course materials.

[4] M. Hu, Y. Peng, Z. Huang, and D. Li, "A multi-type multi-span network for reading comprehension that requires discrete reasoning," *arXiv preprint arXiv:1908.05514*, 2019.

[5] E. Segal, A. Efrat, M. Shoham, A. Globerson, and J. Berant, "A simple and effective model for answering multi-span questions," *arXiv preprint arXiv:1909.13375*, 2019.

[6] M. Marrero, J. Urbano, S. Sánchez-Cuadrado, J. Morato, and J. M. Gómez-Berbís, "Named entity recognition: Fallacies, challenges and opportunities," *Computer Standards & Interfaces*, vol. 35, no. 5, pp. 482–489, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548912001080

[7] A. Mikheev, M. Moens, and C. Grover, "Named entity recognition without gazetteers," in *Ninth Conference of the European Chapter of the Association for Computational Linguistics*, 1999, pp. 1–8.

[8] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 3rd edition draft, 2023.

[9] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 50–70, 2020.

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[11] L. Wang, K. Zheng, L. Qian, and S. Li, "A survey of extractive question answering," in *2022 International Conference on High Performance Big Data and Intelligent Systems (HDIS)*. IEEE, 2022, pp. 147–153.

[12] D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner, "Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs," in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:67855846

[13] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[14] A. T. Liu, W. Xiao, H. Zhu, D. Zhang, S.-W. Li, and A. Arnold, "Qaner: Prompting question answering models for few-shot named entity recognition," *arXiv preprint arXiv:2203.01543*, 2022.

[15] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," *arXiv preprint arXiv:1806.03822*, 2018.

[16] A. Mansouri, L. S. Affendey, and A. Mamat, "Named entity recognition approaches," *International Journal of Computer Science and Network Security*, vol. 8, no. 2, pp. 339–344, 2008.

Ringraziamenti

Life happens when you're making plans. Così recitava il testo di una canzone che stavo ascoltando un giorno, disteso a letto con le mie solite cuffiette, dopo una giornata di studio al DEI.

Ecco, per me non è così. *Life happens when you're NOT making plans* avrebbe dovuto dire la canzone. È certamente vero che senza un obiettivo, organizzazione e dedizione non sarei dove sono oggi, ma è anche vero che le direzioni della mia vita sono state tracciate un po' quando ne ho avuto meno il controllo, quando non stavo facendo piani appunto.

Pensavo di aver quasi finito gli esami della sessione estiva, per questo accetto di dare il mio nome come volontario per una conferenza IEEE sull'Intelligenza Computazionale. Qualche ora di lavoro in cambio di buffet di cibo e la possibilità di ascoltare e conoscere esperti del mio campo di studi provenienti da tutto il mondo. Mi butto un po' senza pensarci, "tanto non si viene mai presi" ho pensato.

E invece no, mi ritrovo a questa conferenza, un po' per caso. Il primo giorno mi mettono al guardarobe. D'estate, chi vuoi che abbia un giubboto da lasciare ... un'ora a fare nulla. Poi si accorgono che la posizione che avrebbe dovuto coprire un dottorando è scoperta, mi offro nuovamente senza neanche sapere cosa avrei dovuto fare. Lì conosco Filippo Costanti, dottorando presso l'Università di Siena. Super simpatico, mi invita ad unirmi in pausa pranzo a conoscere tutto il suo gruppo. I giorni passano, ormai sono infiltrato tra i dottorandi Senesi. Il giovedì c'è la cena di gala. Mi siedo ad un tavolo con loro e iniziamo a mangiare e bere, anzi più bere che mangiare. Ci sono ragazzi che non conosco, ma il vino fa fare velocemente amicizie. Sono le 3 di mattina, siamo ai Navigli e sono seduto con un mio Professore, i dottorandi e questi sconosciuti. Tanto sconosciuti ora non lo sono più, si sono presentati e mi hanno offerto un tirocinio presso l'azienda Expert.AI dove lavorano. "Ah, ma allora ho detto qualcosa di intelligente tra tutti quei bicchieri di vino" ho pensato.

E così mi ritrovo a Siena. Andrea Zugarini è il mio tutor aziendale, ma tutti in ufficio sono disponibili e pronti a darmi idee e insegnarmi ciò di cui si occupano. A questo punto forse però dovrei iniziare i ringraziamenti. Ringrazio Andrea Zugarini per essere stato non solo un tutor aziendale competente e di avermi insegnato il rigore nel modo di fare ricerca, ma anche un amico. La ricerca avevo sempre detto non avrebbe fatto per me, ma con lui come supervisore avrò solo da imparare. E così ho tanto da imparare dagli altri professionisti di Expert.AI Leonardo Rigutini, Marco Ernandes e Giovanni Angelini, i quali ringrazio ancora una volta per questa opportunità e di avermi a loro modo seguito ed essersi interessati al mio lavoro. Ringrazio infine il Professore Giorgio Satta per essere stato l'esempio

di Relatore perfetto che tutti meriterebbero. I suoi meeting settimanali e il suo modo di approcciare il lavoro di tesi hanno fatto sì che questo lungo lavoro mai fosse di peso, difficile sì, ma sempre stimolante.

Tornando tra il Friuli e Veneto ringrazio primi tra tutti la mia famiglia. Li ringrazio per avermi sempre fornito il loro supporto in tutte le mie scelte, e di non avermi mai fatto mancare nulla. Non sarò stato tanto presente fisicamente a casa in questi ultimi anni, ma sapere che da casa ho tutto il vostro supporto e tifo mi dà sempre una spinta in più. Ringrazio ugualmente i nonni, gli zii e tutti gli amici di famiglia.

Ringrazio gli amici di Tamai, per le mille avventure assieme. Piano piano ognuno di noi prenderà una direzione che lo porterà lontano da Tamai, ma torneremo sempre al Bar Centrale per berci uno spritz assieme e raccontarci dei nostri futuri traguardi.

Ringrazio gli amici dell'Università per aver reso decisamente meno pesante andare a lezione. Le nostre cavolate in pausa lezione, i pranzi in mensa Piovego, le serate ai Navigli e a casa di Pippo e Sajina rimarranno sempre nei miei ricordi.

Ringrazio infine la persona che si è affacciata nella mia vita proprio quando meno me l'aspettavo, ancora una volta *when I was not making plans*. Monika, per descrivere le tue qualità e come mi fai sentire ogni volta che sono con te, un'altra tesi magistrale mi servirebbe. Solo noi sappiamo il lavoro che questi ultimi mesi ci hanno richiesto per essere qui dove siamo, ma con l'altro affianco è sempre sembrato tutto più leggero. Abbiamo fatto i vagabondi tra la Francia, Toscana, Padova, Verona e Tamai. Abbiamo viaggiato parecchio e raggiunto molti traguardi, e così mi auguro saranno gli anni a venire assieme. Tanti altri traguardi e un mondo da esplorare assieme ci aspettano.