AUTOMATIC EXTRACTION OF INFORMATION FROM MEDICAL PUBLICATIONS USING NLP.

IGNACIO TALAVERA CEPEDA, 202102073

MASTER'S THESIS

June 2023

Advisor: Ira Assent

Coadvisor: Amalie Brogaard Pauli

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

AUTOMATIC EXTRACTION OF INFORMATION FROM MEDICAL PUBLICATIONS USING NLP.

IGNACIO TALAVERA CEPEDA

Master's Thesis

Department of Computer Science
Faculty of Natural Sciences
Aarhus University

June 2023

*Y*ou think in prose, he says, but people here are more concise.
Like haiku, says my boyfriend, who compares everything with literature
Seven syllabes plus the present tense

— Stine Pilgaard, *Meter i sekundet*.

*S*oy.

— Andoni Zaballa.

First and foremost, to my family. All that I've accomplished is thanks to them, especially to Mamá, Pilar, Gema, Isabel, Juan, Nena and Antonia.

To Rosa, Erlaitz, Rafa, Nico, Andoni, Nuria, Álvaro, Lucía, Juan, Alicia and Jesús. No ha habido un día en el que no me hayáis querido mucho, y no va a haber un día en el que yo no os quiera mucho también.

To Sophie, for being my biggest fan and letting me be hers.

To Natasja, Mathias, Lea, Jannik, Skursch, Marie, Mikkel, and the rest of 44 Stuen. Thanks for building a home with me. I will always be back.

To my friends in Madrid, for being so present and supportive when I decided to move 2000 kilometres away for them, and for always welcoming me in my own town. Aarhus is as yours as it is mine.

To Ira, Amalie and Rasmus. Thanks for giving me the opportunity and support me along the way.

To my friends at Recognai, now called Argilla, for introducing me to NLP and sharing their passion and love. I'll be forever grateful.

ABSTRACT

Evidence-based approach to medicine (EBM) is a medical approach that uses meta-analyses to provide medical evidence combining the results of different studies. However, those meta-analyses are labour-intensive, time-consuming, and require domain experts. To overcome this, this thesis researches different NLP approaches to automatically extract that information, using Transformer models. Using the PICO corpus as the data source, and focusing on clinical trials that follow the PICO standard (Participants, Intervention, Control and Outcomes), this thesis compares the effectiveness of two main approaches, extractive question answering and sequence tagging, over different data availability scenarios. Three final models are proposed for extracting information from the PICO corpus, a sequence-tagging model that assigns a label to each sequence with valuable information, a question-answering model that extracts answers from the input data given a question, and an ensemble of classifiers that combines those two previous approaches. The proposed models achieve high performances and can be used to produce end-to-end solutions for extracting medical information.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

THESIS

1 INTRODUCTION

1.1 CONTEXT

Evidence-based approach to medicine (EBM) is the medical approach in which healthcare professionals use the best available research evidence to make good clinical decisions about the care of patients [45]. EBM aims to integrate the clinician's experience, the data obtained from the patients, and the scientific information available to guide decision-making about clinical management [18]. Meta-analyses are one of the most fundamental tools in EBM, as they provide medical evidence by combining the results of different research studies to determine the effectiveness of treatments [12]. However, they are labour-intensive and time-consuming as they involve manually reading hundreds of unstructured research articles and structuring the data in a human-supervised way [23]. A current estimation indicates that conducting a review of an unstructured, natural language medical article requires 67 weeks from registration to publication [8]. The number of articles increases rapidly, which adds to the already existing problem [35]. Therefore, current EBM methods are rigorous but do not scale with the volume of unstructured evidence [33].

In this context of evidence-based medicine and meta-analyses, NLP techniques are often used [29, 58], especially using the Transformer architectures [59]. The Transformer architecture is the state of the art in NLP due to their ability to learn contextual relationships between words, thanks to their self-attention mechanism. They can be pretrained on large amounts of data to understand the underlying language structures, therefore being very versatile. Transformers have rapidly become the dominant architecture in NLP, and one of the tasks they are used for is data extraction. Automatic and semi-automatic approaches for data extraction over research articles have been proposed but they are still not ready for practical use, as that kind of data extraction requires a high level of accuracy that is still difficult for NLP pipelines to achieve [33].

1.2 OBJECTIVE

This thesis aims to research NLP pipelines capable of extracting information about clinical academic publications, such as its objective, conclusion, country, ethnicity, age and gender distribution of the study, size of the study population, inclusion and exclusion criteria, outcomes, cohorts, etc. The problem and the data we have available are

highly mutable, which means that it can be solved with different approaches, as it can be defined as several different NLP tasks. In this thesis, two main approaches will be tested and compared:

- As a token classification approach: sequence tagging [2].
- As a text classification approach: extractive question answering [37].

The objective is not only to build the most efficient and accurate pipeline for extracting the medical information, thus helping to create meta-analyses but to compare the approaches conceptually and practically and understand how the data available can be used in the most efficient way possible. Traditionally, this task has been regarded as a sequence tagging problem; with NLP models analyzing each token and trying to detect sequence of tokens that can be identified as a certain entity [1]. However, due to the popularity of prompt-based approaches [30], classic sequence tagging problems have been faced with them, trying to bridge the gap between pre-training and downstream tasks. The extractive question-answering approach that has been researched in this thesis is closer to the input processing that foundational models have during training (a string with an special token, where the model have to generate a sequence of tokens that is treated as the prediction). Therefore, theoretically, the transfer of learning can be more efficient.

1.3 MOTIVATION

The motivation behind this work, is to help health care workers by researching the creation of models capable of easily and accurately extracting relevant information from medical papers in the English language. That extraction can drastically improve their work on investigating previous cases and deciding treatments for current cases. The lack of automatic approaches for this kind of data extraction currently being used is also one of the main reasons to work on the subject.

1.4 DEVELOPMENT

Both the token and text classification approaches have been developed at the same time, refining intermediate models and comparing the results obtained. The models are increasingly more complex, as the feedback obtained in the development and the hardships found are the foundation for the next iterations. At the end of this project, both the best-performing model for the token classification and the

1 NLP pipelines similar to the ones used for the task of Name Entity Recognition have been used [29]

best-performing model for the text classification approaches are trained and compared. This work is supported by data analysis and research on the state of the art in both the NLP and the biomedical fields.

1.5 DATA SOURCES

This thesis is focused on a medical environment, thus the data sources used must also come from this background. The main dataset used is the PICO corpus [35], which is an open-source dataset that describes the core elements of clinical trials in the PICO standard: Participants, Intervention, Control and Outcomes. The data, research abstracts from breast cancer studies, is extracted from PubMed[2]. Annotators labelled text spans that identify the PICO elements, and, for each category, sub-categories were developed to capture detailed information. In total, they created 26 sub-categories, that were called entities. The annotation process also considered binary and continuous outcomes. The corpus has 17,739 entities, with varied frequencies for each entity. The PICO dataset facilitates NLP research on automatic information extraction and was key for the development of this thesis.

1.6 RESOURCES USED

Pytorch[3] [36], Huggingface Transformers[4][64] and spaCy[5] are the main libraries used for developing and training the models, for Python 3. Furthermore, the rest of the Huggingface libraries[6][7][8][9] that support Transformers are used for managing the tokenization, the data preprocessing, and the evaluation of the obtained results. WanDB[10] and Argilla[11] are used to track the experiments and draw conclusions from the obtained results.

This thesis has been carried on in collaboration with the company *Silvi.ai*[12]. They offer tools for end-to-end screening and data extraction tools in the domain of Systematic Literature Review and Meta-analysis. They have provided guidance throughout the whole process.

2 https://pubmed.ncbi.nlm.nih.gov/
3 https://pytorch.org
4 https://huggingface.co
5 https://spacy.io
6 https://huggingface.co/docs/datasets/index
7 https://huggingface.co/docs/evaluate/index
8 https://huggingface.co/docs/tokenizers/index
9 https://huggingface.co/docs/api-inference/index
10 https://wandb.ai/
11 https://www.argilla.io
12 https://www.silvi.ai

2 BACKGROUND

2.1 DEEP LEARNING

Artificial Neural Networks were theorized around the middle of the 20th century. When the computing and data capabilities of the neural networks caught up with the theoretical work, researchers started incrementing the complexity of these networks with more layers and new activation functions. These systems started producing better results than their rule-based counterparts, and they opened a window of chance for domains where rules cannot be made, as their underlying characteristics could not be encoded given the complexity of tasks [61].

Even though Natural Language Processing (NLP) was conceived long before the popularization of neural networks (the first NLP models to obtain good scores on common NLP tasks were based on N-grams[9]), the current shift towards Deep Neural Networks and big, data-hungry models supposed an important breakthrough of possibilities and performance on the field.

2.2 TRANSFORMER MODELS

Transformer models are based on a network architecture introduced in 2017, primarily focused on Natural Language Processing [59]. The main idea that Transformers introduced is the self-attention mechanism, allowing the model to focus on different parts of the input sequence and allowing the modeling of dependencies without regard to their distance in the input or output sentences. A schema of the Transformer model architecture can be seen in Figure 1, where it can be seen that Transformers follow an encoder-decoder architecture (the encoder is on the left of the Figure, and the decoder, on the right) and uses stacked self-attention and fully connected layers for both parts. The encoder is composed of 6 layers made by a multi-head self-attention mechanism, and a densely-connected feed-forward network. The decoder is also composed by 6 layers, but in addition to the two previous sub-layers, it also has a multi-head attention layer for processing the output of the encoder.

Codification of written text in Transformer models is made through pre-trained representations of words based on distribution, which is a process called word embedding. With this technique, capable of

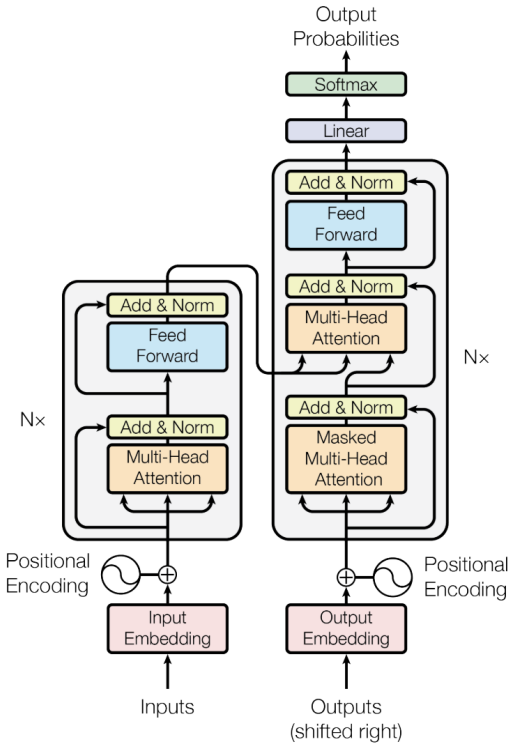


Figure 1: Schema of the Transformer model architecture, from [59].

encoding words while preserving information about their similarity[1], and with systems to obtain these encoded vectors such as context2vec [34], GloVe [38], and ELMo [40], architectures which learn upon that information of word distribution and closeness emerged. Nowadays, NLP models are able to process word relations (like noun-pronoun), sentence structure and word ambiguity. New, disruptive results were obtained on tasks like question answering, textual entailment, Named Entity Recognition (NER) and Sentiment Analysis [6].

2.3 LARGE LANGUAGE MODELS

Current trends on larger architectures and quantities of data popularized Transformer models [14, 59], which established themselves as the standard for NLP tasks thanks to their reusability and capability of handling long-range dependencies between words. Therefore, the focus of AI researchers shifted towards creating big, reliable Transformer models which could be sequentially fine-tuned over a specific task, and making them available to the general public. This approach reduces drastically the time and resources needed for researchers to

1 Word embeddings are mathematical representation of words in a vector space,. These vectors capture semantic and syntactic relationships between words, allowing machine learning models to understand and process textual data more effectively by treating words as numerical inputs.

train a state-of-the-art NLP model, as they only need to train the final part of the pipeline.

In Table 1, a comparison between the most used large language models can be seen, showing their number of parameters and the size of the training dataset. Their development through the years has continuously increased the number of parameters and training dataset sizes, obtaining even larger models with more capabilities. It is expected to see two lines of development over the course of the following years. A first line of development will deal with even larger language models, which will probably surpass the current records of the number of parameters and dataset sizes; and a second line of development will aim for lighter models, balancing size with capabilities. These second type of models cannot reach similar performance on the same downstream tasks used to benchmark larger models, but they offer faster inference time and require a smaller computational impact to train [47]. Figure 2 offers a comparison between these two types of models until January 2020, which can be seen to diverge from January 2019.

Year	Model	Number of parameters	Training data size
2019	BERT [14]	3.4E+08	16GB
2019	DistilBERT [47]	6.6E+07	16GB
2019	ALBERT [25]	2.2E+08	16GB
2019	RoBERTa [31]	3.55E+08	161GB
2020	MegatronLM [51]	8.3E+09	174GB
2020	BETO [13]	3.4E+08	4GB
2020	GPT-3 [10]	1.75E+11	570GB
2021	Switch-C [15]	1.57E+12	745GB

Table 1: Overview of latest large language models until January 2020 [10, 13–15, 25, 31, 47, 51].

2.4 PROMPT LEARNING

Traditionally, NLP models followed a fully supervised learning approach, which relied on manually annotated datasets and feature engineering. Therefore, NLP researchers or engineers were needed to define and extract features from the raw data. With the rise of neural network models, the focus shifted to the architecture of the networks. Pre-training and fine-tuning became the new standard; a model with fixed architecture is to be pre-trained over large datasets and then fine-tuned for specific downstream tasks [30]. Researchers designed specific training objectives for both the pre-training and the fine-tuning stages.

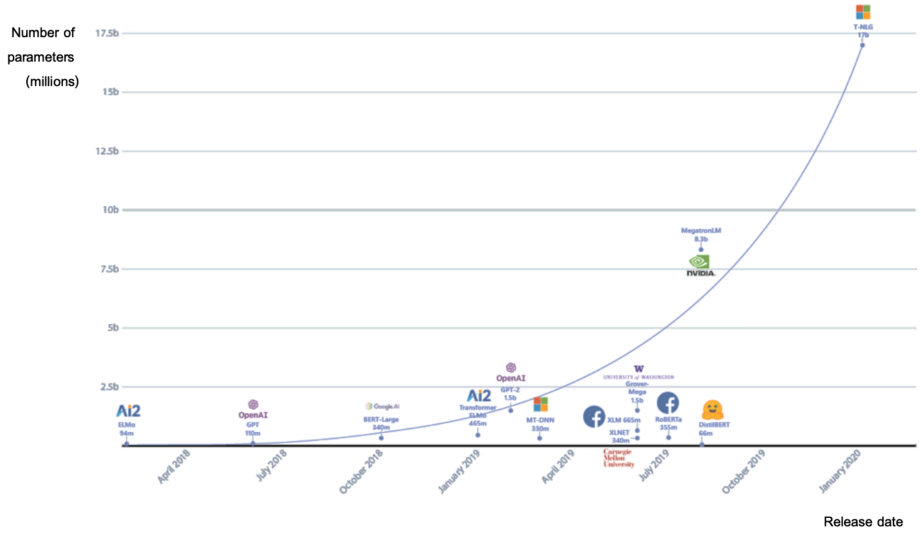


Figure 2: Different language models compared by release date (x axis) and number of parameters, in millions (y axis)[57].

Lately, the trend changed, replacing the pre-training and fine-tuning approach with a new paradigm, consisting of pre-training, prompting and predicting. Instead of adapting pre-trained language models to downstream tasks through objective engineering, tasks are now reformulated to resemble those solved during the original language model training using textual prompts [30]. This approach has the advantages of allowing a single language model to solve numerous tasks and having a pre-training phase more similar to the way large language models are trained but requires prompt engineering to find the most appropriate prompt for each task. This method has been used successfully in various studies [10, 41, 48].

2.5 TRAINING NLP MODELS

The progress made on NLP could not have been possible if there was not a democratizing process in the way that users and researchers train these mathematical models. Backpropagation and the Gradient Descent algorithm, the backbone of the training process of Deep Neural Networks, can be easily implemented using Python libraries like Pytorch [36], TensorFlow [1] and Keras[2]. They also make GPU training accessible. A comparison between the Google search trends of PyTorch and TensorFlow (Keras is usually implemented alongside TensorFlow) can be seen in Figure 4 (with data obtained from Google Trends[3]), where it can be observed that, while TensorFlow used to

2 https://keras.io
3 https://trends.google.com/trends/

Type	Task Example	Input ([X])	Template	Answer ([Z])
Text Classification	Sentiment	I love this movie.	[X] The movie is [Z].	great fantastic ...
	Topics	He prompted the LM.	[X] The text is about [Z].	sports science ...
	Intention	What is taxi fare to Denver?	[X] The question is about [Z].	quantity city ...
Text-span Classification	Aspect Sentiment	Poor service but good food.	[X] What about service? [Z].	Bad Terrible ...
Text-pair Classification	Natural Language Inference	[X1]: An old man with ... [X2]: A man walks ...	[X1]? [Z], [X2]	Yes No ...
Tagging	Named Entity Recognition	[X1]: Mike went to Paris. [X2]: Paris	[X1][X2] is a [Z] entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	[X] TL;DR: [Z]	The victim ... A woman ... ...
	Translation	Je vous aime.	French: [X] English: [Z]	I love you. I fancy you. ...
Regression	Textual Similarity	[X1]: A man is smoking. [X2]: A man is skating.	[X1] [Z], [X2]	Yes No ...

Figure 3: Examples of input, template, and answer for different tasks in prompt learning, from [30].

gather the majority of the attention, Pytorch has outgrown it in the last years.

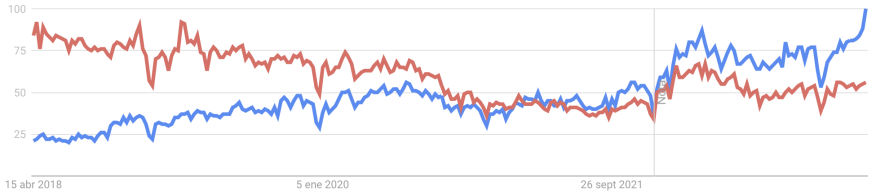


Figure 4: Evolution of search trends of PyTorch (blue) and TensorFlow (red) over the last 5 years. Obtained from Google Trends.

3 TASK & DATA ANALYSIS

In this section of the thesis, both the task and the data available for this task is analysed and discussed in detail, alongside an introduction to automatic data extraction on medical publication that serves as an introduction to the techniques that are proposed. This analysis later lead to a set of initial experiments.

3.1 AUTOMATIC DATA EXTRACTION ON MEDICAL PUBLICACTIONS

As covered in Section 1, meta-analysis combines the results of different studies about the same disease, treatment or outcome. They are essential for enabling evidence-based medicine and clinical decision-making [35]. However, they are time-consuming and expensive, as they require the work of domain experts over hundreds of unstructured, natural-language articles. The number of articles being published increases at a higher rate, which also makes it more difficult to conduct those analyses. That is the motivation behind automatic meta-analysis systems.

Using automatic data extraction techniques to streamline the process of identifying and analyzing medical research is becoming more relevant thanks to the advances in NLP, which allow more efficient systems and pipelines, therefore providing an easier experience for clinicians and research over the meta-analyses, and saving time and resources that would otherwise be spent manually reading through vast quantities of research articles.

3.1.1 *Sequence-tagging Approaches & Extractive Question Answering*

In this subsection, an explanation of both sequence-tagging and extractive question-answering approaches and a comparison both in terms of requirements, performance and results is given.

3.1.1.1 *Sequence tagging*

One of the more natural ways to formulate data extraction problems and design NLP pipelines to solve them is to follow sequence-tagging approaches. Named Entity Recognition (NER) tags in-text entities with their corresponding type. In previous works, NER systems are reformulated as common sequence labelling, with custom entities [14, 46, 65]. By doing that, different kinds of entities that may correspond to the PICO standard [35] can be detected in the input text. We can

see an example on the text in Listing 2, extracted from an abstract of the PICO dataset, where the entities in Listing 1 are annotated. That annotation is then used to convert all the tokens inside the tags in the tokens for the specific entities, and the rest of the tokens are set to O.

```
intervention,"{'answer_start': [0], 'text': ['Tamoxifen']}"
control,"{'answer_start': [219], 'text': ['placebo']}"
eligibility,"{'answer_start': [230], 'text': ['women who are at
    increased risk of breast cancer']}"
```

Listing 1: Example PICO entities as presented in the PICO dataset.

```
"[intervention]Tamoxifen[intervention] for the prevention of
    breast cancer: psychosocial impact on women participating in
    two randomized controlled trials. The purpose of this study
    was to evaluate the psychosocial implications of tamoxifen
    versus [control]placebo[control] in women who are at
    increased risk of breast cancer. The 488 women in the
    psychosocial study were recruited from participants in two
    placebo–controlled, double–blind, randomized, controlled
    trials that investigated the efficacy of tamoxifen in the
    prevention of breast cancer in [eligibility]women who are at
    high familial risk[eligibility]."
```

Listing 2: Abstract extrated from the PICO dataset, with entities annotated using tags.

Training a sequence labelling system requires a lot of labelled training data, and labelling a large corpus of tokens requires domain knowledge. Both factors combined can make the availability of data to effectively train sequence tagging systems limited. These problems aggravate with new subdomains that can appear, and the different languages that the data can be in [29].

3.1.1.2 *Extractive Question Answering*

The current popularity of prompt-based techniques as a new dominant paradigm in NLP has also influenced automatic data extraction. Prompt-based methods have reformulated the input, trying to make them closer to the data that the large language models are trained into, thus closing the gap between the pre-training phase and final, downstream task [30]. Using knowledge from question-answering data, QA models have the potential to enhance sequence-tagging techniques with low training resources.
Extractive Question Answering is a derived technique from prompt-based learning. It extracts a span of text from a given context as the answer to a specified question [37]. It is a prompt-based learning technique, as the system is provided with a question and a text passage, and they are tokenized and included in the system together. The prompt serves as a guide for the system, helping it to focus on the relevant parts of the passage and extract the answer. A visualization

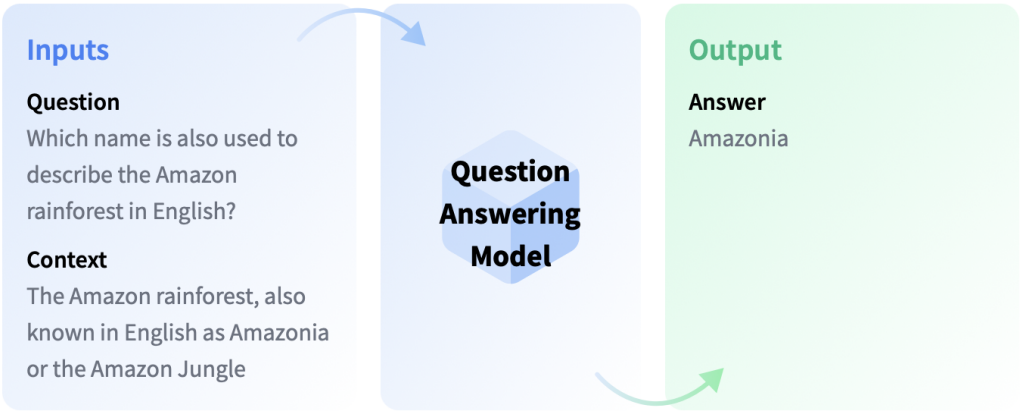


Figure 5: Example of inputs and outputs in an extractive question answering pipeline. Extracted from Huggingface Transformers [63].

of an extractive question-answering approach can be seen in Figure 5. There, a question an a context is introduced into the pipeline as the input, and the model extracts the answer to the question from the context.
Extractive question answering has been proven to be an effective NLP approach. Studies prove its effectiveness in a large corpus of elements [7] and in the few-shot scenario by mutating NER examples into question-answering examples and providing faster and robust results in low-resource conditions [29].

3.1.2 *Generative models*

The usage of pretrained large language models have showcased their capabilities in the field of natural language processing (NLP). On the GLUE benchmark[1] [60], methods based on pre-training are significantly better than other approaches. Among the two principal types of pre-trained models, there are two main types:

- **BERT-like models**, designed for language understanding and best suited for tasks related to text classification and token classification [14]
- **GPT-like models**, mainly utilized for language generation tasks [10].

The GPT-like models are designed for language generation by pre-training a Transformer decoder model on a large text dataset, with the objective of predicting the next work token taking into consideration the word tokens from previous words. GPT-2 [42] and GPT-3

1 The GLUE benchmark is an evaluation framework that assesses the performance of different models on various language understanding tasks. It consists of nine diverse tasks and provides a standardized way to compare models.

[10] have large model sizes and have remarkable results on various downstream tasks, including text classification tasks.
BioGPT, a GPT-like model for biomedical text generation and mining, was proposed, with GPT-2 as the backbone model. After a pretraining on the PubMed abstract corpus[2], it achieves state-of-the-art results on the benchmarks BC5CDR, KD-DTI and PubMedQA question [32]. The study also demonstrated the capability of its biomedical text generation compared to standard GPT models trained with regular text extracted from the Internet.

3.2 TASK ANALYSIS

In order to extract information from medical publications, aiming to help EBM and meta-analyses using NLP, the tasks consist of automatically extracting several types of information from raw text. That information is usually related to medical studies; information about the patients or the population of the study, the problem to solve, the intervention that has been conducted, the outcome of that intervention, and data about comparisons and control procedures in those studies.
The PICO standard [35], which is common in meta-analyses and is the one used on our main dataset, stands for Population, Intervention, Comparator and Outcome. It is a framework used for developing clinical questions and research studies in evidence-based medicine, and it is useful for breaking down complex medical issues into more manageable parts, for easy analysis and comparison.
In the standard, the **population** refers to the patients that are involved in the study or who the question pertains; they could be individuals, groups of patients, healthcare workers, etc. The **intervention** describes the actions carried out within the study; treatments and procedures. The **comparator** specifies the current practice against which the intervention is measured or the alternative approach that is taken. Finally, the **outcomes** define the results that are relevant for explaining the effectiveness of the intervention. The combination of these four categories is used to establish a structure for clear communication about the scope and purpose of a particular medical inquiry or investigation.
Inside those four categories, several entity classes can be extracted, and it is usually up to the designers of the meta-analyses or the datasets to decide the entities that will be included. The source of those datasets is usually a collection of publicly available medical publications, from where the information can be extracted. Either the abstract, a specific part of the text, or the whole, raw text, can be included in those datasets.

2 https://pubmed.ncbi.nlm.nih.gov/download/

Therefore, the complexity of the extraction task is usually dependent on these variables:

- The chosen entity classes; their amount, complexity, and potential conceptual overlapping.
- The amount of text available.
- The quality of the text.
- The frequency of the classes, taking into account potential class imbalance.

To automatically extract the entities belonging to the proposed classes, the NLP pipelines have to correctly adapt to these characteristics. The amount of labelled data available can be especially critical, as we could move from a classical supervised learning scenario, where classic text classification techniques usually excel, to few-shot learning, where new techniques, like prompt-based learning, can be more effective. For this dataset, sequence tagging is considered to be the classical text classification technique that adapts better to the task, while extractive question answering can extract the desired information using the prompt capabilities of the big language models. There is no consensus on which approach is better in every scenario, and different NLP problems require independent analysis, but recent studies have shown that generative models and prompt-based learning techniques can outperform classical text classification techniques on few-shot learning scenarios by exploiting the prompting capabilities of the backbone models, due to way they are trained [10, 30]. Among the reasons that back prompt-based learning techniques in few-shot learning scenarios, some of the most relevant for this specific case are:

- Prompts provide specific instructions to the model helping it understand the desired behavior.
- Prompts allow the model to focus on important information from the textual input.
- Pretrained models are trained over big amounts of textual inputs in a similar way that prompts are introduced into the fine-tuned models, by placing an specific token in the input where the model have to make a prediction.

Choosing which techniques to use in order to most effectively solve this automatic data extraction task requires a specific set of experiments that are described in Chapter 5, and they are also dependent on the specific datasets that are used.

3.3 DATA ANALYSIS

This section explains the data analysis performed over the data corpus used in this thesis and explains the standard that it follows.

3.3.1 *Insights on the PICO standard*

The PICO corpus that has been used in this thesis as the main dataset to research automatic information extraction. It was introduced by Mutinda et al. in 2022 [35], and it contains the main elements of the PICO standard: participants, intervention, control and outcomes. The corpus is made of 1011 abstracts of breast cancer publications extracted from PubMed[3], and it is specially designed to develop automatic extraction pipelines to support EBM.
The breast cancer abstracts included in the corpus belong to a study type called RCT, and they are not meta-analysis or systematic reviews themselves. RCT stands for a randomized controlled trial, and it is a type of experiment used to test if an intervention or treatment has a significant effect under some controlled conditions. Participants of an RCT are assigned to either a group that receives the experimental treatment or a group that acts as a control, receiving either no treatment or standard care [11]. Some examples of RCTs that are not related to breast cancer or oncology are drug testing, surgical procedures, psychological therapies or dietary changes. [22, 39, 52, 53].
A correctly carried out RCT is considered to be a gold standard for clinical trials, and they are commonly used in efficacy testing. They can also provide additional information about adverse effects and reactions, as they provide compelling evidence that a certain treatment causes an effect on human health [19].

3.3.1.1 *Data Exploration over the PICO Corpus*

The PICO Corpus was accessed through the GitHub repository[4]. Prior to the start of the experimentation phase, a data analysis and exploration procedure was performed, to better understand the data corpus.
The abstracts were manually annotated by two annotators, one of them was hired from an annotation company and the second was an author of the original work, that were asked to identify PICO elements [35]. Several sub-categories were designed to capture fine-grained, detailed information within each category. The hierarchy can be seen in Figure 7. 26 sub-categories were annotated, with a high class imbalance between them. The distribution of instances per class can be seen in Table 2 and Figure 6.

3 https://pubmed.ncbi.nlm.nih.gov
4 https://github.com/sociocom/PICO-Corpus

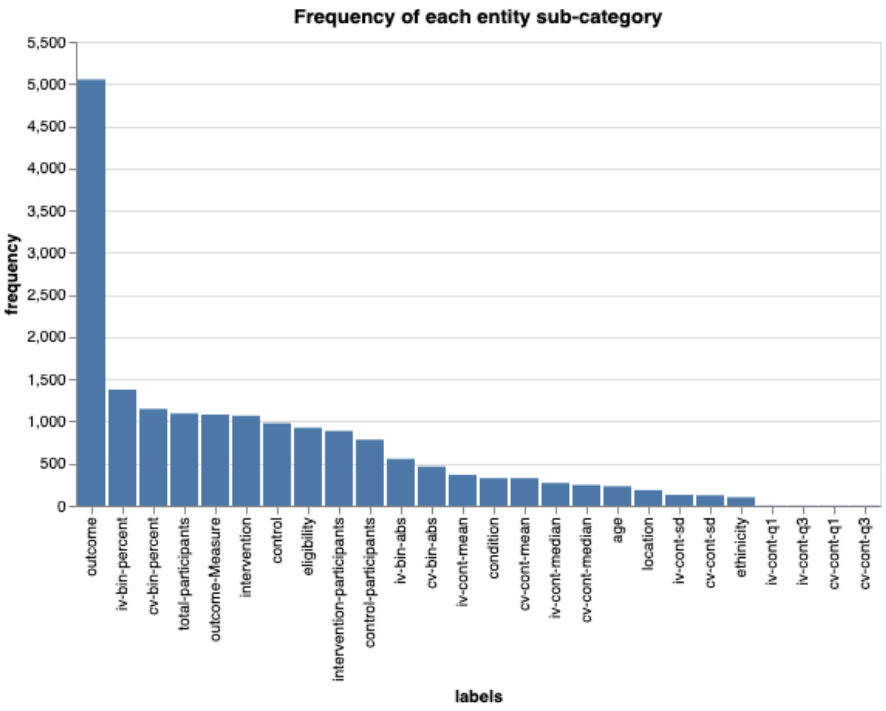


Figure 6: Frequency of each entity, in descending order.

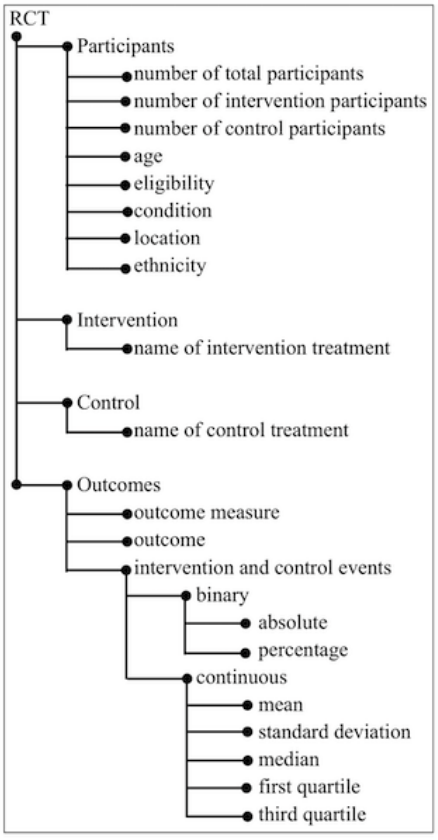


Figure 7: Hierarchy of labels in the PICO corpus.

Category	Sub-category	Number of instances of the entity	Number of abstracts where the entity appears
Participants	total-participants	1094	847
	intervention-participants	887	674
	control-participants	784	647
	age	231	210
	eligibility	925	864
	ethnicity	101	83
	condition	327	321
	location	186	168
Intervention & Control	intervention	1067	1011
	control	979	949
Outcomes	outcome	5053	978
	outcome-measure	1081	413
	iv-bin-abs	556	288
	cv-bin-abs	465	258
	iv-bin-percent	1376	561
	cv-bin-percent	1148	520
	iv-cont-mean	366	154
	cv-cont-mean	327	154
	iv-cont-median	270	140
	cv-cont-median	247	133
	iv-cont-sd	129	69
	cv-cont-sd	129	67
	iv-cont-q1	4	3
	cv-cont-q1	4	3
	iv-cont-q3	4	3
	cv-cont-q3	4	3

Table 2: Frequency of each entity and the number of abstracts in which they are found [35].

For the *participants* category, the eight entities that are used in the corpus include the total number of participants, the number of participants belonging to the intervention and the control groups, their condition, eligibility, age, ethnicity and location. Breast cancer is the main condition, but some of the studies focus on treating associated conditions and not the cancer itself.
Intervention and control categories are merged into one category, as they have only one sub-category each. They reffer to the intervention and control techniques present in the study.
In the outcome section, the annotations include outcome measures[5], and outcomes that were measured[6]. There is also information for the numeric texts that identify the number of participants experiencing a particular outcome. Two types of outcomes are considered: binary (successful or failed, alive or dead) and continuous (e.g., pain on a numerical scale). There are also labels to capture the specific type of numeric texts:

- *cv* stands for control group of the outcome.
- *iv* stands for intervention group of the outcome.
- *bin* stands for binary outcome.
- *cont* stands for continuous outcome.
- *abs* stands for absolute value of the outcome.
- *percent* stands for percent value of the outcome.
- *q1* stands for first quartile of the outcome.
- *q3* stands for third quartile of the outcome.
- *sd* stands for standard deviation of the outcome.
- *cv-cont-sd* stands for the mean value of the outcome.
- *median* stands for the median value of the outcome.

With this label code in mind, we can translate any entity in the outcome category. For example, *cv-cont-sd* stands for an outcome as continuous value, of the control group, using the standard deviation as metric. We can see an example of a binary and a continuous outcome in Table 3. Binary outcomes are discrete units, like the number of patiens in an intervention or control group, while continuous values usually refer to measurements or results. An example of an annotated abstract is presented in Figure 8, where the different PICO entities can be seen highlighted in the text.

5 Examples of outcome measures can be incidence of metastases, mortality or number of tumors.

6 Examples of outcomes can be adverse events, improvement in quality of life or cancer development.

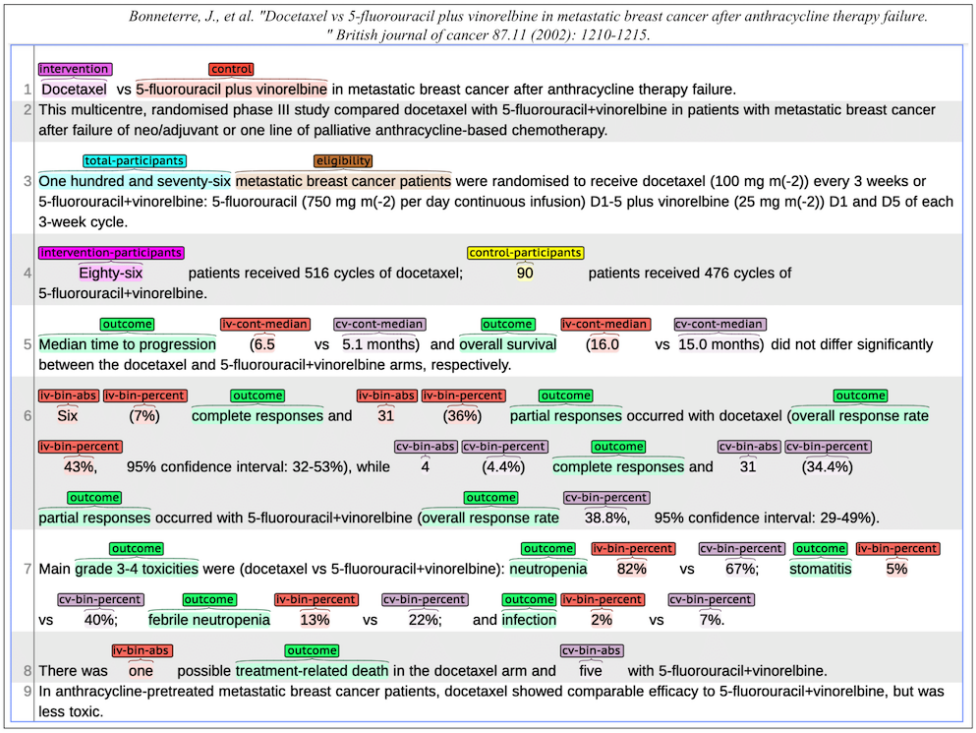
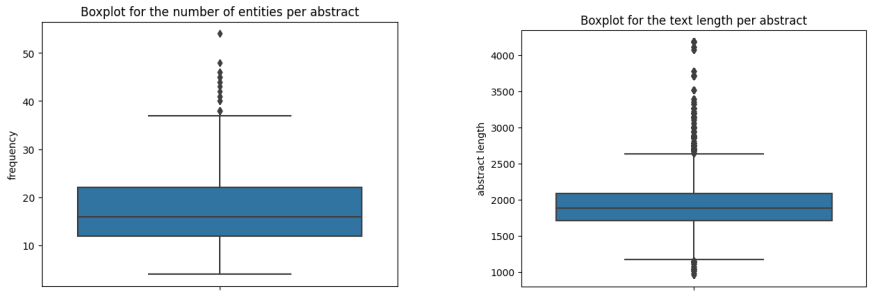


Figure 8: A medical abstract annotated with the PICO standard used in the corpus [35]

In those 1011 abstracts, 17,739 entities are annotated. As seen in Figure 2, *outcome* is the most common entity, and the continuous outcomes for the quartile values (q1 and q3) are the least frequent. Intervention, outcome and control are the entities found in most abstracts (100%, 97% and 94%, respectively). In Figure 9, boxplots showing the entities per abstract and the text length per abstract can be seen. There is an average of 17.5459 entities per abstract, but some extreme outliers with more than 50. The abstracts have an average text length of 1916.7 characters, but also some extreme outliers with 3,000 to 4,000 characters. This dataset has a high density of sequence-tagging instances per dataset instance, and the raw text that needs to be fed into an NLP pipeline is relatively long.

Binary Outcome Example	Continuous Outcome Example
<iv-bin-abs>Four</iv-bin-abs>patients in the intervention group and <cv-bin-abs>two</cv-bin-abs> in the controlgroup were <outcome>lost to follow-up</outcome>.	<outcome>Depression scores</outcome>at follow-up were significantly lower in the exercise group (M = <iv-cont-mean>4.78</iv-cont-mean>, SD = <iv-cont-sd>3.56</iv-cont-sd>) compared to the control group (M= <cv-cont-mean>6.91</cv-cont-mean>, SD =<cv-cont-sd>5.86</cv-cont-sd>).

Table 3: Example of a binary and a continuous outcome [35].



(a) Number of instances per abstract

(b) Abstract's length

Figure 9: Boxplots with the number of instances per abstract in the PICO corpus and with the abstract's length.

Even though 1011 abstracts can be considered few data instances, having 17,739 entities is considered a large number, so we can use this data in a supervised learning scenario. However, there is a clear class imbalance in the dataset, as shown in Figure 6. The outcome entity is extremely overpopulated, while the 14 last entities have fewer than 500 annotated instances. The last four entities, which are the ones referring to the quartiles, have only 4 annotated instances, in three different abstracts (as seen in Table 2). They probably require a special approach to obtain high classification accuracy, given the low data availability.

4

MODEL APPROACH

This section will describe the approaches taken to design NLP models capable of extracting information from medical papers. Several different paths were explored, including text classification techniques, sequence tagging, and ensembles of classifiers that combines both of them. Thus, we provide the theoretical background, how they were executed and the expectations for each, given the related work.

4.1 COMPARING TEXT CLASSIFICATION AND TOKEN CLASSIFICATION

In Natural Language Processing, classification tasks can be grouped into two main approaches: text classification and token classification. Text classification approaches assign predefined labels to an input text [49], which can be a whole document or a piece of it, while token classification approaches assign labels to some tokens in the input text [24]. The objective of the latter is to classify each token to a specific role or category, while text classification tries to fit all of the input text into one or more categories.
In both text and token classification, there are several subtasks that can be relevant to the given problem.

4.1.1 *Text classification*

Text classification cannot be directly applied to the problem if we want to obtain results that not only show if a given entity is present or not in the text. If a multilabel text classification pipeline is applied, and we use a whole publication abstract (or even the whole, raw text), that classifier will probably detect several PICO entities, but would not provide information on where they are or how to distinguish them from other entities, as text classifiers only predict if there if certain categories can be associated with a given input. Therefore, a sub-task of text classification is needed for this dataset.
However, as stated in Section 2, Extractive Question Answering is a subtask beneath text classification that can be applied to this domain. Question Answering models search for an answer in a text. There are several QA variants, but the one that is the most related to our problem is Extractive QA, in which the model extracts the answer from a given text.
Extractive Question Answering, as can be seen in Figure 10, takes as input both a question and a context. Both inputs are tokenized to-

gether, and fed into the NLP pipeline, that learns how to extract an answer for the question present in the text. These pipelines are more suited for few-shot learning scenarios, where there is low data availability, because these models analyze the relationship between the question and the abstract, enabling them to capture nuances. Classic text classification, on the other hand, treats each textual input as an independent unit and does not consider the specific question being asked.

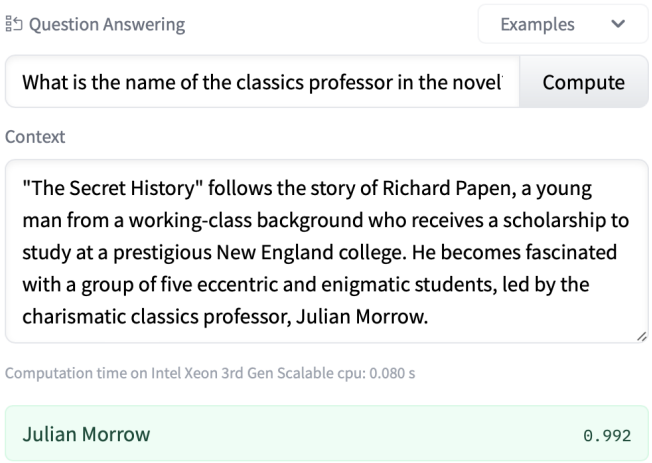


Figure 10: Example of an Extractive Question Answering interface [21].

4.1.2 *Token classification*

Token classification models can be used in this domain by choosing the PICO entities as tags. If we input a text containing medical information, and we train the model with the PICO entities as output labels, we can detect where in the text those entities are present. These models rely heavily on labelled training data containing the token annotations. The quantity and diversity of the training data are key factors in the performance of the overall system [24]. An example can be seen in Figure 11.
There are several popular subtasks in token classification, such as NER tagging [55] and PoS tagging [54], but our case demands a custom sequence-tagging model, not using standards like NER, as we need to introduce the PICO entities as target labels.

4.2 SEQUENCE-TAGGING APPROACH

To create sequence-tagging models, we first need to prepare the data for the task itself. No tokenizing function, nor tokenized input, is provided in the PICO dataset, and just tokenizing the dataset using

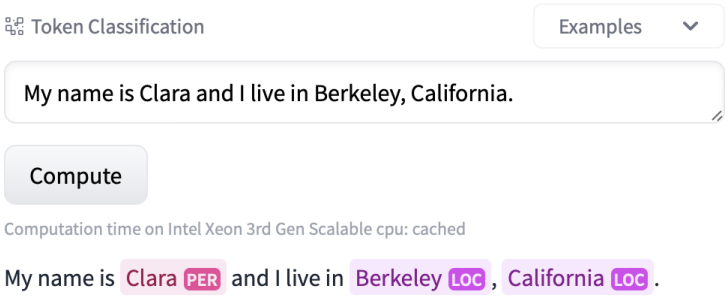


Figure 11: Example of a token classification interface which searches for persons and locations in input texts [21].

a tokenizer from spaCy[1] or HuggingFace Transformers[2][64] can lead to alignment errors. For example, if we try to tokenize the following extract of an abstract:

```
n=40
```

The whole extract is considered only one entity by the spaCy tokenizer, but the PICO dataset only recognises only 40 as an entity. If we use the spaCy tokenizer as default, the whole text would be translated into one token, but Huggingface Transformers cannot recognize entities that are smaller than a token. If this scenario happens, the model would consider that the entity is broken and would raise an error.
To solve this scenarios, we first tokenize the entities, using the information from the PICO dataset to detect them in the text and pass them to the tokenizer as already-tokenized words. Then, the an spaCy English tokenizer is used to tokenized the rest of the abstracts, where there is no entities left that can be tokenized incorrectly.
The type of textual input that is common in this domain is text with a high length. Most of the pretrained models have a limit on the number of tokens they can process. For example, the vanilla version of BERT [14] has a length limit of 512 tokens. Therefore, we have to find ways to mitigate this limitation, trying to include as much input information as possible. We will compare truncation techniques with Longformer models [5], which are Transformer models with attention mechanisms that scale linearly, allowing the processing of thousands of tokens at the same time. Some Longformer pretrained models available in HuggingFace Models[3] are able to process inputs of 4096 tokens, which can fit the entire PICO dataset; the longest abstract, after the spaCy tokenizer, has a length of 852 tokens.
HuggingFace Models also offers several models already pretrained over bio-medical data, which can give more accurate results with less

1 https://spacy.io
2 https://huggingface.co
3 https://huggingface.co/models)

training required. Therefore, the two main dimensions that we will take into account to test different sequence-tagging models are:

- Models pretrained with general textual data against models pretrained with biomedical data.
- Pretrained models with a length limit of 512 tokens, where truncation must be applied, against Longformer models, which can fit the whole tokenized input.

4.3 EXTRACTIVE QUESTION ANSWERING APPROACH

Using the Extractive Question Answering approach requires a preprocessing phase, in which the format of the PICO dataset is adapted to the format required by HuggingFace Transformers for this kind of model. The format of the PICO dataset can be seen in Listing 3. All of the entities listed belong to the same abstract, and each entity has ordered information about the number of the entity for the given abstract, the type of PICO entity, the starting and end characters where the entity is present in the text, and the text itself. T3, for example, shows that it is the third entity for the given abstract, it is a control entity that appears between characters 519 to 540 in the text, and the text where the entity appears is *no adjuvant treatment*.

```
T2	total-participants 309 312	960
T3	control 519 540 no adjuvant treatment
T4	intervention 187 208	adjuvant radiotherapy
T5	outcome 942 966 risk of local recurrence
T6	outcome 1039 1068	risk of distant dissemination
T1	eligibility 212 288	early breast cancer patients with more than 15 years of follow-up evaluation
T7	outcome-Measure 1453 1469	overall survival
```

Listing 3: Example of entities predicted over a medical abstract from the PICO dataset. Each entity contains information about its number, its label, the starting and ending characters of the answer in the text and the text of the answer.

Huggingface extractive question-answering models are expecting to obtain a list of dictionaries. Each dictionary contains a tuple *abstract-prediction*, and they contain information about the id of the answer, the character in which it starts, and the text, while also including the original question and context. Therefore, each instance of the PICO dataset is augmented to each unique *abstract-prediction*. The example shown in Listing 3 will become seven different entities when it gets

adapted to the HuggingFace standard. An example of a tuple *abstract-prediction* can be seen in Listing 4.

```
{'answers':
    {'answer_start': [515],
    'text': ['Saint Bernadette Soubirous']},
'context': 'Architecturally, the school has a Catholic character.
     Atop the Main Building\'s gold dome is a golden statue of
    the Virgin Mary. Immediately in front of the Main Building
    and facing it, is a copper statue of Christ with arms
    upraised with the legend "Venite Ad Me Omnes". Next to the
    Main Building is the Basilica of the Sacred Heart.
    Immediately behind the basilica is the Grotto, a Marian place
     of prayer and reflection. It is a replica of the grotto at
    Lourdes, France where the Virgin Mary reputedly appeared to
    Saint Bernadette Soubirous in 1858. At the end of the main
    drive (and in a direct line that connects through 3 statues
    and the Gold Dome), is a simple, modern stone statue of Mary.
    ',
'id': '5733be284776f41900661182',
'question': 'To whom did the Virgin Mary allegedly appear in 1858
     in Lourdes France?',
'title': 'University_of_Notre_Dame'
}
```

Listing 4: Expected format to create a Extractive Question Answering pipeline.

To create Extractive QA models, it is required to perform a preprocessing phase, in which all the PICO instances are transformed into this standard. There is an average of 17 entities per abstract, and the total number of entities is 17,739. Thus, the PICO dataset adapted to the QA standard has the same number of instances.
HuggingFace has a large variety of models that can be fine-tuned to be extractive question-answerers, usually based on the BERT architecture. A similar comparison as with the sequence tagging models will be made, between models pretrained with general textual data and models pretrained with biomedical data.
As also happens with the sequence taggers, some examples in the dataset may have a long context that exceeds the maximum input length. To deal with longer text inputs, we will create several training features from each instance in the dataset, with a sliding window between them. If we take as example the context and the question from Listing 4, and we use as the maximum length of each feature a length of 100 characters, truncating only from the context with a

stride of 50[4], we obtain the four features seen on Listing 5. We will refer to this process as *QA Chunking*.

```
1 '[CLS] To whom did the Virgin Mary allegedly appear in 1858 in
      Lourdes France? [SEP] Architecturally, the school has a
      Catholic character. Atop the Main Building\'s gold dome is a
      golden statue of the Virgin Mary. Immediately in front of the
       Main Building and facing it, is a copper statue of Christ
      with arms upraised with the legend " Venite Ad Me Omnes ".
      Next to the Main Building is the Basilica of the Sacred Heart
      . Immediately behind the basi [SEP]'
2 '[CLS] To whom did the Virgin Mary allegedly appear in 1858 in
      Lourdes France? [SEP] the Main Building and facing it, is a
      copper statue of Christ with arms upraised with the legend "
      Venite Ad Me Omnes ". Next to the Main Building is the
      Basilica of the Sacred Heart. Immediately behind the basilica
       is the Grotto, a Marian place of prayer and reflection. It
      is a replica of the grotto at Lourdes, France where the
      Virgin [SEP]'
3 '[CLS] To whom did the Virgin Mary allegedly appear in 1858 in
      Lourdes France? [SEP] Next to the Main Building is the
      Basilica of the Sacred Heart. Immediately behind the basilica
       is the Grotto, a Marian place of prayer and reflection. It
      is a replica of the grotto at Lourdes, France where the
      Virgin Mary reputedly appeared to Saint Bernadette Soubirous
      in 1858. At the end of the main drive ( and in a direct line
      that connects through 3 [SEP]'
4 '[CLS] To whom did the Virgin Mary allegedly appear in 1858 in
      Lourdes France? [SEP]. It is a replica of the grotto at
      Lourdes, France where the Virgin Mary reputedly appeared to
      Saint Bernadette Soubirous in 1858. At the end of the main
      drive ( and in a direct line that connects through 3 statues
      and the Gold Dome ), is a simple, modern stone statue of Mary
      . [SEP]'
```

Listing 5: Training features obtained after QA chunking with a maximum length of 100 characters and a stride of 50 characters.

HuggingFace tokenizers are able to do QA Chunking, and they also return a variable called *overflow to sample mapping*, which maps each feature to the example it originated from. When we only tokenize one example, a list of zeroes is returned, but if we tokenize more examples, we obtain a list with the index of the record each feature comes from. This variable is essential for transforming our PICO dataset into the QA standard.

Thanks to this technique using the HuggingFace tokenizers dedicated to this model, we can avoid the use of Longformers or other

4 The stride is the number of overlapping tokens between two successive chunks.

chunking techniques like in the sequence taggers. It is also more suited for this type of standard: each instance is a unique *abstract-prediction* tuple, and we only have to find each entity once in the text. Therefore, we can independently search for each entity in each of the features obtained from one abstract. The sequence tagger mode processes the whole input at the same time while searching for all the possible entities. Dividing instances into different features can be problematic, as it would be losing contextual information. However, this technique creates more training and inference instance from an already augmented version of the PICO dataset, so we should expect more computationally-demanding training, validation and testing processes.

4.4 COMPARING QA AND SEQUENCE TAGGERS OVER THE AMOUNT OF AVAILABLE DATA

NER-like systems, like the proposed sequence-tagging models, require big training sizes to accurately make predictions [24]. On the other side, the Extractive Question Answering approach, by being closer to a prompt-based learning scenario (which means that it is also close to the way the big LMs are pretrained in the first place) are more suited for scenarios with fewer or no training instances [30].
A comparison between both main approaches is then suggested, altering the training sizes of the whole dataset. If this hypothesis holds, hybrid models will be proposed, combining the predicting capabilities of both approaches, depending on the number of training instances available. An ensemble of classifiers model that uses the sequence-tagging approach for those entities frequent in the dataset and the QA approach for the underpopulated classes could obtain a higher performance than both the QA and the sequence-tagging models.

5 EXPERIMENTS

This section contains the experiments carried out from the hypothesis set in Section 4, the results obtained from them, and further experimentation after analyzing their results. We have trained and tested models based on the Sequence-Tagging approach, the Extractive Question Answering approach and Ensembles of Classifiers combining the two approaches into one pipeline. Final conclusions after the experiments are discussed in Section 6.

5.1 EXPLANATION OF HYPERPARAMETERS USED

The term *hyperparameters* refers to fixed values that define how the neural network model works in training and inference. Unlike the variables that are learned during training, hyperparameters remain constant throughout the whole process and are set by the Machine Learning practitioner. They serve to guide the behaviour of algorithms towards optimal generalization capabilities. The following list contains an explanation of hyperparameters that are specifically taken into consideration in this thesis:

- **Backbone Language Model**: refers to the pretrained model that serves as the foundation for a larger NLP pipeline. In this thesis, we use large language models that have been pretrained on vast amount of data, and we transfer that learning by fine tuning the model with the data that we have available. We have used both models pretrained with general data and with biomedical data.
- **Learning Rate**: determines the step size taken when moving from one parameter estimation to the next one in order to minimize the loss function. Higher learning rates mean larger steps are taken and convergence may speed up, but this also may lead to instability, as the step could be high enough that a local minimum of the loss function is missed and is left unexplored. Lower learning rates can provide smoother convergence but longer computation times. If the value is too low, convergence may not happen [17].
- **Weight Decay**: it is the hyperparameter which controls a penalty term that is included in the objective function, discouraging weights from becoming large. This creates an optimization process that trades off reducing error terms and maintaining small weights, which prevents the model from memorizing the training data [62].

- **Batch Size**: it represents the number of instances processed together in each iteration, offering faster convergence and requiring fewer iterations per epoch [17].
- **Learning Rate Scheduler**: methods used to determine how much the learning rate should be changed during the course of training [26].

One of the main scopes of the experiments is to compare different approached, and to do so we try to obtain the best-perfoming models following processes called *HyperParameter Optimization*, or HPO. They are a search process over a defined search space of possible hyperparameters for the best set that maximizes the performance, given a target score [66].

5.2 SEQUENCE-TAGGING

In this section, we explore the proposed sequence-tagging models. We have developed a baseline model consisting of regular-sized models[1] trained only on the PICO dataset, with minimal tuning over the hyperparameters. Next, we enhance the baseline using pretrained models that were pretrained on large amounts of clinical and biological texts prior to fine-tuning. Finally, we investigate the potential benefits of utilizing the Longformer architecture, which can process input sequences of greater length compared to conventional transformers.

5.2.1 *Baseline model*

We start the experimentation by producing a baseline model using pretrained models with an input data size of 512 tokens. These models follow a truncation process when the abstract cannot fit the input size, deleting content from the final part of the string. They also have a padding procedure to fill the input vectors with zero values when the input is not big enough to fit the size; nevertheless, the factor that can be limiting the performance of the model is the truncation.
To avoid doing an extensive HyperParameter Optimization process [66] we fix a learning rate of *2e-5* [2] while keeping the rest of the hyperparameters in the default values of the HuggingFace Transformers Trainer[3]. We experiment with three different pretrained models:

1 In this thesis, we call regular-sized models to those pretrained, backbone models that accept an input of regular size, which is usually around 512 tokens for BERT-like models. We use this term to differentiate these models from the Longformer architecture, which is specially designed to accept longer input sizes.

2 In [59] a learning rate of 0.001 with gradient clipping at *4x*, which would correspond to roughly *2e-5* assuming a batch size of 8; and in [14] a learning rate scheduler from *1e-6* to *9e-5* is used. Both suggest that *2e-5* is a good starting point for fine-tuning.

3 This default values, for the hyperparameters that we are focusing in, are the following: batch size value of 8, weight decay value of 0, and learning rate scheduler deacti-

- **xlm-roberta-base**[4]: XLM-RoBERTa is a multilingual variant of RoBERTa [31], a Transformer model that undergoes self-supervised pre-training. It is trained on a massive corpus of raw text data, including 2.5TB of filtered CommonCrawl data [5] spanning 100 languages. It has a comprehensive understanding of multiple languages, enabling the extraction of valuable features for downstream tasks such as training classifiers using the model's generated features as inputs.

- **biobert-v1.1**[6][27]: BioBERT is a specialized language model designed for the biomedical domain. It is pre-trained on large-scale biomedical datasets, offering notable improvements over BERT [14] and previous state-of-the-art models in various biomedical text mining tasks.

- **Bio_ClinicalBERT**[7]: BioClinical BERT is a combination of BioBERT and the clinicalBERT models [4]. This model was specifically initialized from BioBERT and trained on MIMIC III[8], a database containing health records from ICU patients.

We perform a training phase of each one of these models over 20 epochs, as shown in Figure 12. The best-performing result, with a micro-averaged F1 Score of 0.6746 at the end of the training, is **Bio_ClinicalBERT**. This model, therefore, is chosen to do a further HPO process.

5.2.2 *BioClinical BERT models*

The development of sequence-tagging models continues by doing a Hyperparameter Optimization process over the **Bio_ClinicalBERT** model, which gives the best results and was chosen as the baseline model. The chosen ranges for the learning rate, the weight decay and the batch size (the hyperparameters chosen) can be seen in Listing 6.

```
lr = [1e-5, 2e-5, 3e-5, 4e-5, 5e-5, 6e-6, 7e-5]
weight_decay = [0, 0.01, 0.02, 0.03]
batch_size = [4, 8, 16]
```

Listing 6: Range of the learning rate, the weight decay and the batch in the HPO process for the BioClinical BERT models.

vated. Extracted from https://huggingface.co/docs/transformers/main_classes/trainer.

4 https://huggingface.co/xlm-roberta-base

5 https://commoncrawl.org

6 https://huggingface.co/dmis-lab/biobert-v1.1

7 https://huggingface.co/emilyalsentzer/Bio_ClinicalBERT

8 https://www.nature.com/articles/sdata201635

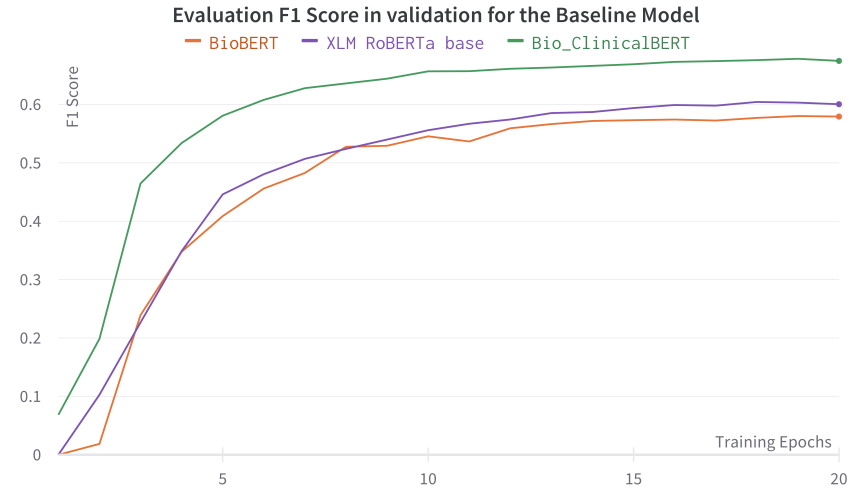


Figure 12: Evaluation F1 Score in the validation for the baseline model experimentation.

After 84 runs (the best 50 runs are shown in 13), the best run corresponds to the hyperparameters shown in Listing 7. From the figure, we can extract that the best learning rates are the higher ones, from *3e-5* onwards; the best batch sizes are 4 and 8, and the runs are evenly distributed over the different weight decay values. The F1 Score of the best model over the test dataset is 0.6727.

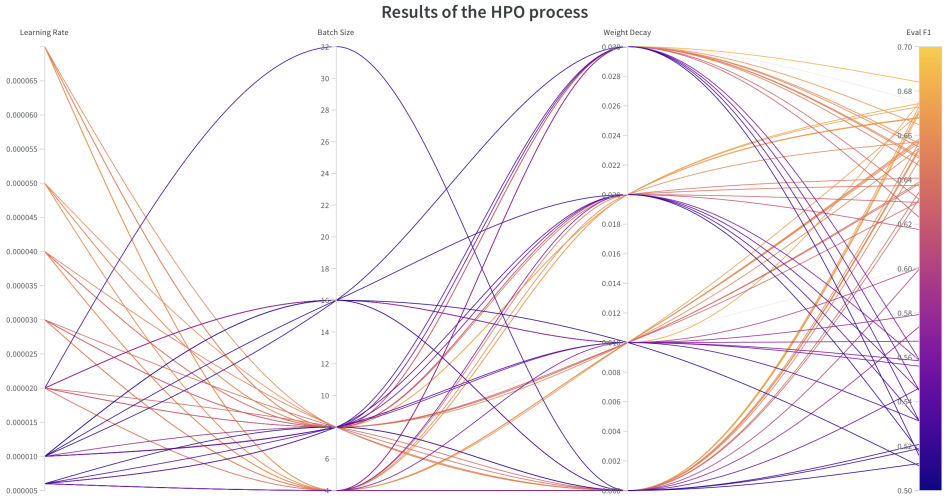


Figure 13: 50 best runs of the BioClinical BERT HPO process and corresponding F1 Score to each run.

```
lr = 7e-5
weight_decay = 0.03
batch_size = 4
```

Listing 7: Hyperparameters of best run after optimization process for the BioClinical BERT models.

5.2.3 *Longformer models*

After the experimentation over BERT-based models with normal input sizes, we start the experimentation over Longformer models, where all of the textual input can be fitted without truncation. We mainly iterate over two Longformer models:

- **Longformer Base**[9]: *longformer-base-4096* is a BERT-like model started from a RoBERTa model that supports sequences of length up to 4,096 tokens by using a combination of a sliding window attention and global attention [5].
- **Clinical Longformer**[10]: *Clinical-Longformer* is a clinical version of Longformer that was further pre-trained using the MIMIC-III dataset[11]. It allows textual inputs of up to 4,096 tokens. It consistently outperforms ClinicalBERT across 10 baseline datasets, with experiments covering NER, QA, Natural Language Processing and other text classification tasks [28].

5.2.3.1 *Longformer Base*

These models require more memory to train, so we have to fix the batch size value to 2, which also affects the training and inference times. The HPO process took 18 runs, over the ranges for the hyperparameters seen in Listing 8. The results of these 18 runs are shown in Figure 14, with higher learning rate and weight decay values being, usually, more effective. The best run, which gives a validation F1 Score of 0.7185, corresponds to the hyperparameters in Listing 9.

```
lr = [2e-5, 3e-5, 4e-5, 5e-5, 6e-6, 7e-5]
weight_decay = [0, 0.01, 0.02]
batch_size = [2]
```

Listing 8: Range of the learning rate, the weight decay and the batch in the HPO process for the Longformer Base models.

```
lr = 7e-5
weight_decay = 0.02
batch_size = 2
```

Listing 9: Hyperparameters of best run after optimization process for the Longformer Base models.

9 https://huggingface.co/allenai/longformer-base-4096
10 https://huggingface.co/yikuan8/Clinical-Longformer
11 https://www.nature.com/articles/sdata201635

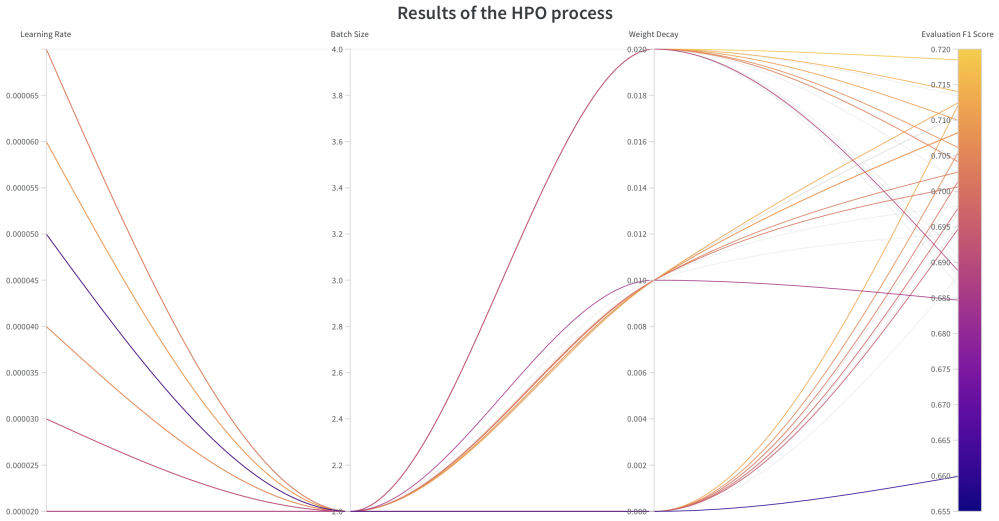


Figure 14: Runs of the Longformer Base HPO process and corresponding F1 Score to each run.

5.2.3.2 *Clinical Longformer*

A second HPO process is performed using **Clinical Longformer** as the backbone language model. We face the same limitation with the batch size as with the base Longformer model, and we cannot use a higher batch size value than 2, so the hyperparameters that are optimized are the learning rate and the weight decay. And, given the previous results, and knowing that best runs usually correspond to high learning rate values, we constrained the search space more. The optimized ranges can be seen in Listing 10, and the result of the HPO process, with the corresponding validation F1 score for each run, can be seen in Figure 15.

```
lr = [3e-5, 4e-5, 5e-5, 6e-6, 7e-5]
weight_decay = [0, 0.01, 0.02]
batch_size = [2]
```

Listing 10: Range of the learning rate, the weight decay and the batch in the HPO process for the Clinical Longformer models.

The best model obtains a validation F1 Score of 0.7134, and the most optimal hyperparameters are the ones shown in Listing 11, which are the same as the ones obtained for the Longformer Base model.

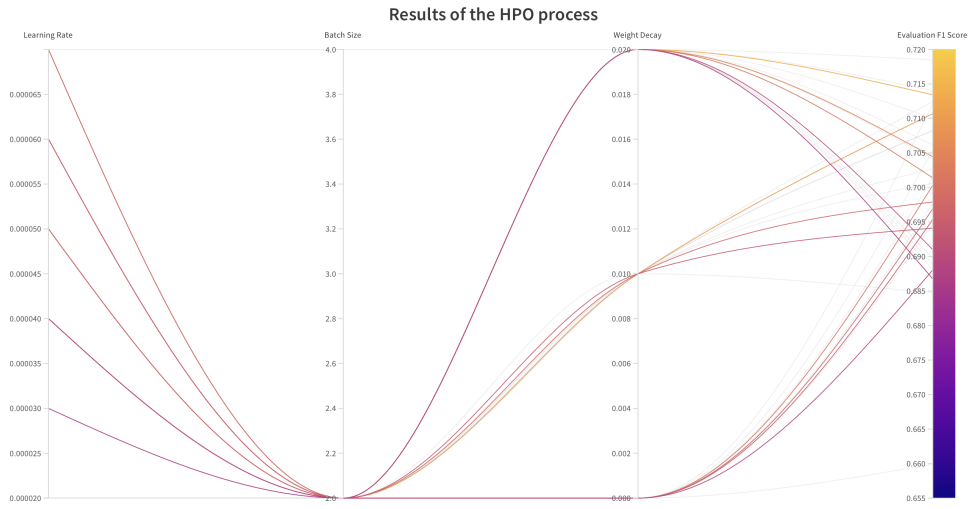


Figure 15: Runs of the Clinical Longformer HPO process and corresponding F1 Score to each run.

```
lr = 7e-5
weight_decay = 0.02
batch_size = 2
```

Listing 11: Hyperparameters of best run after optimization process for the Clinical Longformer models.

5.2.4 *Obtained results*

After all the sequence-tagging models obtained in this part of the experimentation, we have grouped the best-performing of each category in Table 4[12]. Both regular and Longformer models perform well on the sequence-tagging task, with the Longformer models performing slightly better on both validation and test splits. That difference in predicting capabilities is probably due to the fact that the Longformer does not need truncation to process the input dataset, and they process more input information. However, given the bigger size of the Longformer models, both training and inference times are higher, and retraining these models is more computationally intense.
Figure 16 shows the test F1 score distribution per class obtained by the best model of this section, which is the Clinical Longformer model after HPO and fine-tuning processes. We can see that the four least common entity classes, the ones corresponding to the quartiles, have

12 Note that there is no test F1 score in the baseline and in the Clinical Longformer model because they were outperfomed in the validation phase by BioClinical BERT and Longformer Base. Test results were only calculated after optimization, and the comparison of different models was also part of that optimization; test results were only calculated for the best regular model and the best Longformer model.

Models	Best Learning Rate	Best Weight Decay	Best Batch Size	Best F1 Validation	Best F1 Test
Baseline	2e-5	0	8	0.6746	
BioClinical BERT	7e-5	0.03	4	0.6845	0.6727
Longformer Base	7e-5	0.02	2	0.7185	0.6886
Longformer Clinical	7e-5	0.02	2	0.7134	

Table 4: Best sequence taggers model obtained in the experimentation phase.

a F1 Score value of 0, effectively not getting any correct prediction. We can also observe that the most common entity, *outcome*, is not the one getting correctly predicted most often, so there is not a direct correlation between the number of instances and F1 Score. On the other hand, the worst predicted categories are also below the average, indicating that the model finds it difficult to make predictions on low data scenarios. Nevertheless, this scenario is far from corresponding to a few-shot learning one, and the sequence tagger system obtains fairly good results, given the high number of instances that it has to train.

5.3 EXTRACTIVE QUESTION ANSWERING

In Question Answering, both the SQuAD dataset and the SQuAD metrics are very often used to obtain state-of-the-art models [50, 67]. SQuAD stands for Standford Question Answering Dataset, and it is a reading comprehension dataset with questions created by volunteers on a set of Wikipedia articles, where the answer to every question is a segment of the text, or the question is unanswerable [43]. The current version, SQuAD 2.0, has 150,000 questions written adversarially to look similar to answerable ones. NLP pipelines trained with this dataset must be able to not only answer questions correctly but also determine when no answer is supported by the paragraph and abstain from answering.
This dataset and task have an official metric standard associated, which has become the standard in Extractive Question Answering tasks. The HuggingFace Evaluation library[13] includes this standard, which combines two different metrics [44]:

- **Exact Match**: it measures the percentage of predictions that match any of the ground truth answers exactly.

13 https://huggingface.co/spaces/evaluate-metric/squad

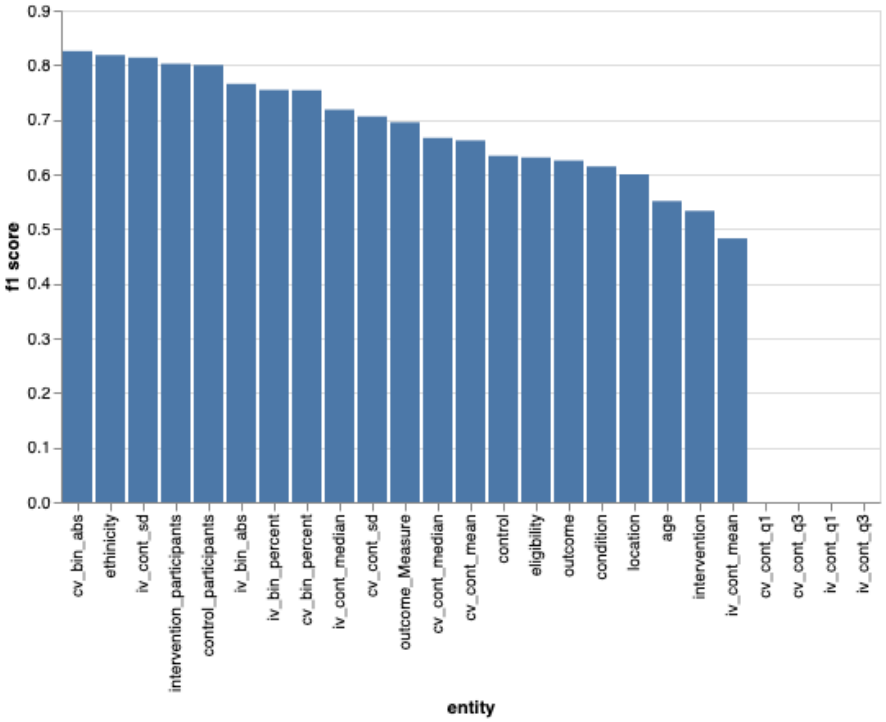


Figure 16: Test F1 Score per class obtained by the best Clinical Longformer model.

- **F1 Score**: it is the macro-averaged overlap between the prediction and the ground truths. Each prediction and reference is treated as a bag of tokens, and the F1 is computed. The maximum value of all the ground truth answers for a given question is taken and then averaged over all the other answers from all the other instances.

Both metrics have a range from 0 to 100 but can be normalized to a range of 0 to 1. In Listings 12 and 13, an example of a prediction and a reference being compared using the standard is shown. The library expects a list of dictionaries as input for both predictions and references. The dictionary containing each prediction must show the predicted text and an id, while the dictionary containing each reference must offer a list of possible answer starts and texts and the id. Ids must match.

```
from evaluate import load
squad_metric = load("squad")
predictions = [{'prediction_text': '1976', 'id': '56
    e10a3be3433e1400422b22'}, {'prediction_text': 'Beyonce', 'id'
    : '56d2051ce7d4791d0090260b'}, {'prediction_text': 'climate
    change', 'id': '5733b5344776f41900610e1'}]
references = [{'answers': {'answer_start': [97], 'text': ['1976'
    ]}, 'id': '56e10a3be3433e1400422b22'}, {'answers': {'
```

```
    answer_start': [233], 'text': ['Beyonce and Bruno Mars']}, '
    id': '56d2051ce7d4791d0090260b'}, {'answers': {'answer_start'
    : [891], 'text': ['climate change']}, 'id': '5733
    b5344776f41900610e1'}]
results = squad_metric.compute(predictions=predictions,
    references=references)
```

Listing 12: Example of usage of the HugginFace Evaluate library for obtaining SQuAD metrics.

```
{'exact_match': 66.66666666666667, 'f1': 66.66666666666667}
```

Listing 13: Example of results of the HugginFace Evaluate library for obtaining SQuAD metrics.

These metrics are the ones used to evaluate and compare the models in this section, as they express the precision of the predictions obtained by a model. F1 Score is specially important, both because partially good answers are also taken into consideration, and because it is used for the rest of the approaches in the thesis, so comparison is easier.

5.3.1 *Baseline Model*

For the QA models, we start by developing a baseline model, with very few optimizations, and pushing the accuracy higher on following iterations. As explained in Section 4, from the PICO dataset, each tuple *abstract-entity* was transformed into an independent instance to form the QA dataset. In this first iteration, each instance in the dataset follows the standard seen in Listing 14, where the question field is filled using the PICO entity. The QA pipeline receives the context text, and the word "*intervention*", or any of the other labels, as the question.

```
Context:  A randomized, prospective study of endometrial
    resection to prevent recurrent endometrial polyps in women
    with breast cancer receiving tamoxifen [...]
Question:  intervention
Answer:  {'answer_start': [35], 'text': ['endometrial resection'
    ]}
```

Listing 14: Instance of the QA dataset during training, showing the tuple *abstract-question*.

This baseline model has been obtained before developing the code that allowed to output of validation scores during training, thus only using the training and validation loss to avoid overfitting (Figure 17). The model is trained using the default values, and a learning rate of *2e-5*, in the same way that the baseline model was done for the sequence tagging model. After 2 epochs of training, we obtain the results shown in Listing 15.

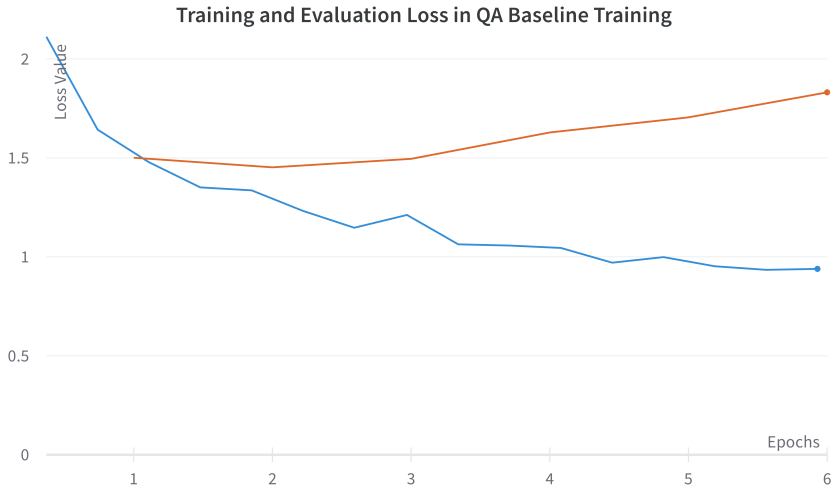


Figure 17: Training and validation losses during the training of the baseline QA model.

```
{'exact_match': 34.92108229988726, 'f1': 43.225651330054774}
```

Listing 15: Test metrics obtained after training the baseline model.

5.3.2 *Exact Match and F1 Score during training*

Obtaining validation metrics during training is important to be able to track the real performance of the model in each epoch, and be able to understand when the model stops learning and starts to overfit. However, given the nature of the predictions made by the question-answering models, allowing the model to obtain these metrics during training requires the use of a special Trainer object, *QuestionAnsweringTrainer* (see Listing 24), and two custom functions, one for passing the reconstructed predictions from the training pipeline to the evaluation pipeline (*post_process_function()*) and another that computes the metrics using that information (*compute_metrics()*).

The documentation regarding this special type of trainer can be found in the source code from HuggingFace[14], where a script to run question-answering pipelines[15] is provided, and the chapter on the NLP course regarding Question Answering[16]. From there, we can extract how those two custom functions work.

```
    # Initialize our Trainer
    trainer = QuestionAnsweringTrainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset if training_args.do_train
            else None,
        eval_dataset=eval_dataset if training_args.do_eval else
            None,
        eval_examples=eval_examples if training_args.do_eval else
             None,
        tokenizer=tokenizer,
        data_collator=data_collator,
        post_process_function=post_processing_function,
        compute_metrics=compute_metrics,
    )
```

Listing 16: Initialization of a *QuestionAnsweringTrainer* object, with the arguments of *compute_metrics* and *post_process_function* being used.

```
metric = evaluate.load("squad_v2" if data_args.
    version_2_with_negative else "squad")

def compute_metrics(p: EvalPrediction):
        return metric.compute(predictions=p.predictions,
            references=p.label_ids)
```

Listing 17: Example of a custom *compute_metrics* function using SQuAD v1 and v2 metrics.

The parameter that *compute_metrics()* receives is an *EvalPrediction* object[17]. *EvalPrediction* is a custom container class that holds the predictions, label *ids* and inputs as NumPy arrays[18]. With those arrays, the

14 https://github.com/huggingface/transformers/blob/main/examples/pytorch/question-answering/trainer_qa.py
15 https://github.com/huggingface/transformers/blob/main/examples/pytorch/question-answering/run_qa.py
16 https://huggingface.co/learn/nlp-course/chapter7/7?fw=tf
17 https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_utils.py
18 A NumPy array is a homogeneous collection of data which can be thought of as an ordered array of elements that share a common type (e.g. integers, floats etc)

compute_metrics() function is able to reconstruct the predictions and calculate the SQuAD metrics on validation.
However, *metric.compute()* is expecting an input similar to the standard shown in Listing 12, not NumPy arrays. To convert the predictions to the input expected for the SQuAD metric, we need to use the *post_process_function()* (Listing 25). HuggingFace provided an example function to transform the predictions of question-answering models to substrings of the original text, which originally return start and end logits[19]. Therefore, that function is customised to work for both the prediction and the reference standard seen in Listing 12, and both *post_process_function()* and *compute_metrics()* are passed to the custom Trainer. The custom functions can be seen in Appendix A.1.
With those two functions, we are able to do training while getting the validation F1 Score and exact match metrics after each training epoch, and therefore obtain insight into how well the model is learning to generalize and when real learning stops and overfitting starts.

5.3.3 *QA with Manual Prompting*

Up until now, we have made the *abstract-question* pairs filling the question field with the name of the PICO entity. However, using prompts closer to natural language should be beneficial, as they are closer to the way LMs are pretrained [30]. Especially, prompts that include the "five Ws" (Who, What, When Where, and Why) question words work the best, as they match the formulation of QA questions, therefore matching the structure of the SQuAD-based datasets [29].
To create the next model and try to improve the performance of the question answerers, we handcraft a natural language prompt for each of the PICO categories. That list can be seen in Table 5. Translating each category to the natural language prompt, and with the ability to obtain F1 Score metrics during training, we try to obtain a better-performing model by following an HPO process focused on the learning rate, the batch size and the weight decay. We also use the sweep method available on WanDB[20] to automate the exploration of the search spaces. It allows us to use a Bayesian search method, which takes a probabilistic approach to find the best configuration. This algorithm builds a probabilistic model of the unknown objective function (the best hyperparameters possible) and maps each configuration to its performance. As more configurations are evaluated, the

along with some additional metadata such as shape or size. These arrays can be manipulated using mathematical functions to perform tasks such as linear algebra operations or to apply filters. They form the foundation upon which other packages are built on top of Python.

19 https://github.com/huggingface/transformers/blob/main/examples/pytorch/question-answering/utils_qa.py

20 https://docs.wandb.ai/guides/sweeps

model is updated, and the sweep becomes more precise in picking the hyperparameters, focusing on more promising regions [21].
The backbone model chosen is DistilBERT base, a small and fast version of the BERT base model, pretrained with SQuAD dataset. The hyperparameters are optimized inside the following ranges:

- **Learning Rate**: search is made over a uniform distribution between 1e-5 and 6e-5.
- **Batch Size**: either 4, 8, or 16, as 16 is the highest our hardware could train with.
- **Weight Decay**: either 0.1, 0.2, or 0.3. We increase the values from previous HPO processes, given the poor variation that lower values gave on previous HPOs.

The results of the HPO process can be seen in Figures 18 and 19. Afterwards, we retrain the model with the best hyperparameters and obtain the results seen in Listing 18. The obtained test F1 Score is lower than the score obtained in the baseline model, which could indicate that the natural language prompts are worse for our dataset and our task. However, we are also using a relatively small model, so we decided to keep iterating using bigger backbone models.

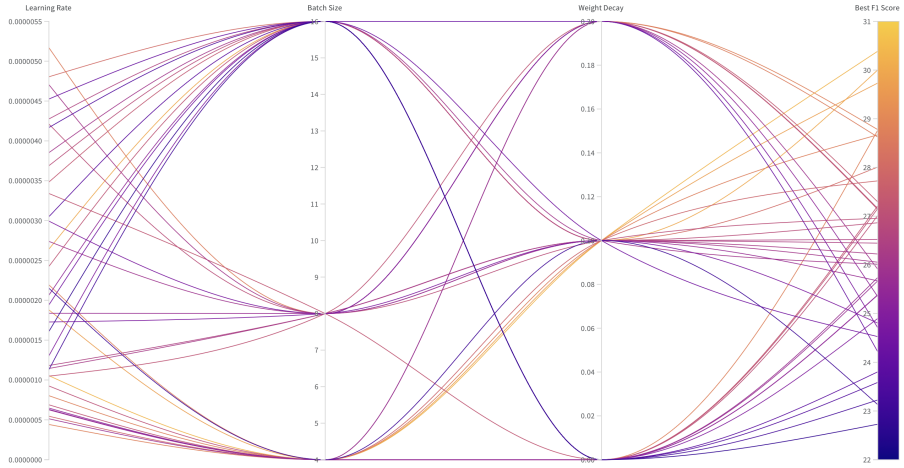


Figure 18: Learning rate, batch size and weight decay of the HPO runs using natural language prompts and DistilBERT, with the corresponding best evaluation F1 Score.

```
{'exact_match': 31.4543404735062, 'f1': 40.43054966176619}
```

Listing 18: Test exact match and F1 Score for the best run using natural language prompts and DistilBERT.

21 https://wandb.ai/site/articles/bayesian-hyperparameter-optimization-a-primer

PICO entity	Natural Language Prompt
intervention	What is the intervention of the study?
condition	What is the condition of the study?
total-participants	What is the number of total participants of the study?
age	What is the age of the participants of the study?
ethinicity	What is the ethinicity of the participants of the study?
intervention-participants	What is the number of the intervention participants of the study?
control-participants	What is the number of the control participants of the study?
outcome-Measure	What is the outcome measure of the study?
iv-bin-abs	What is the absolute value of the binary outcome of the intervention group?
outcome	What is the outcome that was measured on the study?
cv-bin-abs	"What is the absolute value of the binary outcome of the control group?"
control	"What is the control of the study?"
eligibility	What is the eligibility criteria of the study?
cv-bin-percent	What is the percentage value of the binary outcome of the control group?
iv-bin-percent	What is the percentage value of the binary outcome of the intervention group?
iv-cont-median	What is the median value of the continuous outcome of the intervention group?
cv-cont-median	What is the median value of the continuous outcome of the control group?
location	What is the location of the study?
iv-cont-q1	What is the first quartile value of the continuous outcome of the intervention group?
iv-cont-q3	What is the third quartile value of the continuous outcome of the intervention group?
cv-cont-q1	What is the first quartile value of the continuous outcome of the control group?
cv-cont-q3	What is the third quartile value of the continuous outcome of the control group?
iv-cont-mean	What is the mean value of the continuous outcome of the intervention group?
cv-cont-mean	What is the mean value of the continuous outcome of the control group?
iv-cont-sd	What is the standard deviation value of the continuous outcome of the intervention group?
cv-cont-sd	What is the standard deviation value of the continuous outcome of the control group?

Table 5: Handcrafted natural language prompts for the PICO entities.

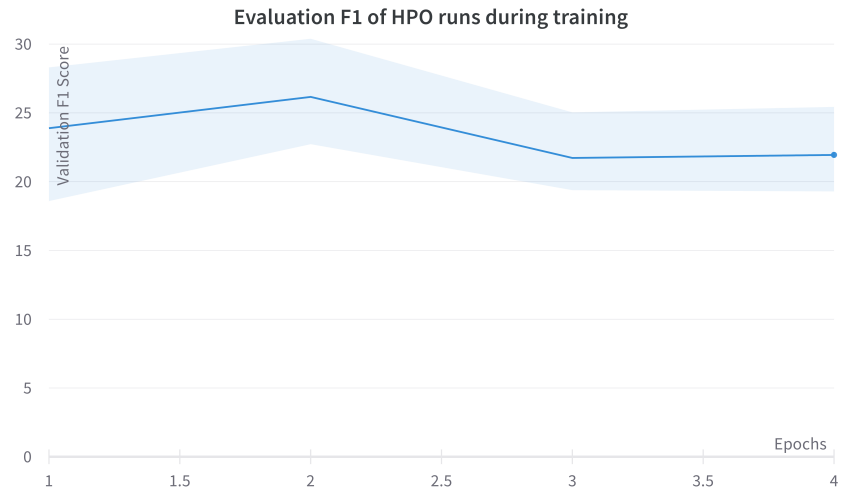


Figure 19: Evaluation F1 Score during the training of the runs were made for the HPO using natural language prompts and DistilBERT. Showing the mean value, and maximum-minim value espectrum.

5.3.4 *Bigger models and Manual Prompting*

For this type of model, we keep most of the conditions used for the natural language QA models developed in the last section but we use bigger, more specialized models. The three backbone models that are tested are the following:

- **RoBERTa Base SQuAD 2**[22]: a RoBERTa base model, fine-tuned using the SQuAD 2.0 dataset.
- **BioM ELECTRA Base SQuAD 2**[23] [3]: model fine-tuned on the SQuAD2.0 dataset and then on the BioASQ8B-Factoid training dataset[24].
- **DeBERTa v3 Base SQuAD 2**[25]: DeBERTa v3 base model, fine-tuned over the SQuAD 2 dataset. DeBERTa [20] is a variant of BERT that aims to improve BERT in its performance of language understanding tasks.

These models are not only bigger in terms of parameters, but the second model is pretrained with biomedical data. Therefore, we theorise that we can get increased performance, not only comparing the previous natural language prompt model but also from the baseline. The HPO process follows the same ranges as the previous models, but now the backbone model is included as a hyperparameter. Still using

22 https://huggingface.co/deepset/roberta-base-squad2
23 https://huggingface.co/sultan/BioM-ELECTRA-Base-SQuAD2-BioASQ8B
24 https://github.com/dmis-lab/bioasq8b
25 https://huggingface.co/deepset/deberta-v3-base-squad2

a Bayesian search algorithm, and with 20 runs per backbone model, we obtain the validation F1 Score results that can be seen in Figure 20.

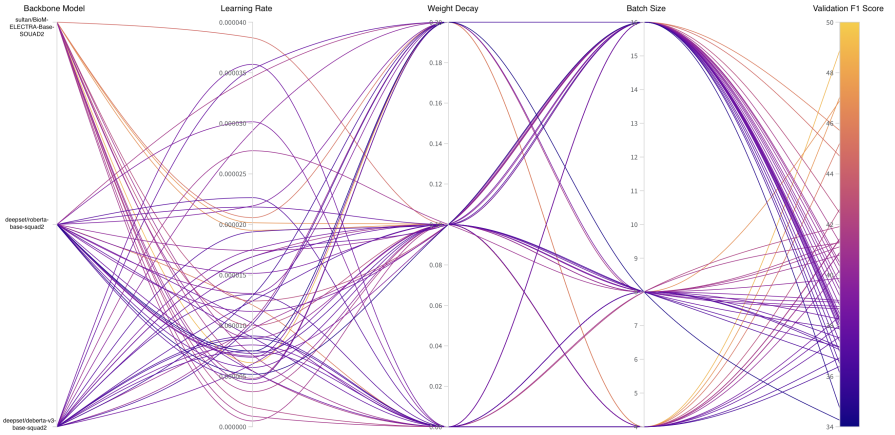


Figure 20: Backbone model, learning rate, batch size and weight decay of the HPO runs using natural language prompts, with the corresponding best evaluation F1 Score.

After the HPO process, we retrain the best-performing model and obtain the exact match and F1 Score values over the test dataset shown in Listing 19. From these experiments, we conclude that BioM ELECTRA Base SQuAD 2 is the best-performing backbone model, as the HPO search space distribution shows. Therefore, using a backbone model already pretrained on biomedical data is beneficial for our task, at least in the context of extractive question answering.
We have produced a model that reaches 0.51 F1 Score in testing, surpassing both the baseline model and the previous natural language prompt model. Therefore, the usage of natural language prompts, combined with bigger, biomedically pretrained models, is beneficial for creating extractive question-answering pipelines for the PICO dataset.

```
1 {'exact_match': 43.74295377677565, 'f1': 51.28892627544654}
```

Listing 19: Test exact match and F1 Score for the best run using natural language prompts and BioM ELECTRA Base SQuAD 2.

5.3.5 *Ensemble prediction using Generative Models*

In the natural language prompt models, we have manually handcrafted prompts that are close to the human language, to ask the model for the different PICO entities that can be present in the text.

Entities with ensemble prediction	Entities with no ensemble prediction
intervention	iv-bin-abs
condition	cv-bin-percent
total-participants	iv-bin-percent
age	iv-cont-median
ethnicity	cv-cont-median
intervention-participants	iv-cont-q1
control-participants	iv-cont-q3
outcome-Measure	cv-cont-q1
outcome	cv-cont-q3
control	iv-cont-mean
eligibility	cv-cont-mean
location	iv-cont-sd
	cv-cont-sd
	cv-bin-abs

Table 6: PICO entities that were augmented using automatic promp-tuning and those that were not.

This approach allows us to do fine-grained control over the model behavior, but to do quality manual prompt-tuning, domain expertise is required [16].
Automatic prompt tuning, on the other hand, involves using automated techniques, like text-generating models, evolutionary algorithms and reinforcement learning, as the idea is to produce synthetic prompts that can be improved over training or feedback. It can save the effort of using the domain knowledge and iterating over it, however, using external computational models requires additional resources.
As we do not have specific domain knowledge, we include an ensemble prediction section using generative models, to try to push further the capabilities of the extractive QA models. To do so, we use a generative NLP model to generate new prompts for each of the entities. We generate four new prompts for each of the entities, and with the prompt that we had already written, we obtain five prompts per entity.
Not all of the PICO entities receive automatically-generated models. Some of the entities, like *cv-bin-percent* or *iv-cont-sd*, ask the model for a very specific detail, and the generative models are not able to produce extra prompts that are different enough to be included, after analyzing them as human-readable sentences. Table 6 shows the PICO entities that were augmented, and the ones that were not.

To generate the new prompts, we use *OpenAssistant LLaMa 30B SFT 6*[26] through HuggingChat[27]. The LLaMA models [56] are a collection of language models ranging from 7 billion to 65 billion parameters, trained exclusively on public datasets. LLaMA-13B outperforms GPT-3 (175B) on most benchmarks, and LLaMA-65B is competitive with the best models currently available. The obtained prompts can be seen in Tables 10 and 11.
Our model is altered to only output a prediction if, at least, three out of the five prompts obtain the same prediction when introduced in the model. We retrain the best model from the last section following this approach, under the same conditions. The entities that are not augmented are predicted using only one prompt, in the same way as the rest of the previous QA models. Afterwards, the following model was asked to perform inference over the test dataset, and we obtained the following test results shown in Listing 20.

```
{'exact_match': 47.20969560315671, 'f1': 54.49576445810399}
```

Listing 20: Test exact match and F1 Score for the best run using ensemble prediction and BioM ELECTRA Base SQuAD 2.

We can see that the previous best score has also been surpassed, and we have obtained an extractive question-answering model capable of obtaining a test F1 Score of 0.54. The baseline has been improved by more than 0.1.

5.3.6 *Results obtained*

Table 7 shows a summary of the best models obtained with each of the different question answering models. Better models are more complex, especially with the use of prompts close to the natural language. The best-performing model uses a natural language prompt, and some of the PICO entities were being classified by an ensemble using NLP generative models, and it achieved 0.11 test F1 Score more points than the baseline model. The use of large language models as backbone models also improved the performance; *DistilBERT* was not able to even outperform the baseline model when it was used with natural language prompts, but *BioM ELECTRA Base SQuAD 2* can outperform it by a big margin.

26 https://huggingface.co/OpenAssistant/oasst-sft-6-llama-30b-xor
27 https://huggingface.co/chat/

Approach	Backbone Model	Learning Rate	Batch Size	Weight Decay	Test F1 Score
Baseline	DistilBERT Base SQuAD	2e-5	16	0	0.4322
Natural Language Prompts	DistilBERT Base SQuAD	2.53e-5	4	0.1	0.4043
Natural Language Prompts with bigger models	BioM ELECTRA Base SQuAD 2	1e-5	4	0.2	0.5128
Natural Language Prompts with ensemble prediction	BioM ELECTRA Base SQuAD 2	1e-5	4	0.2	0.5495

Table 7: Best question answering models obtained in the experimentation phase.

5.4 SEQUENCE TAGGERS AND QA MODELS OVER DIFFERENT TRAINING SIZES

As we explored in Section 4, the key difference between the sequence taggers and the question-answering approach is that sequence-tagging approaches may be more reliable when there is sufficient data to perform regular training, but the QA models can outperform them in few-shot learning scenarios. To research this balance between both approaches over different training sizes, we have designed an experiment that trained two of the best-performing models obtained in the previous section over different training sizes.

For the sequence-tagging models, we choose the best model obtained, as seen in Table 4. For the QA models we use the model with hand-crafted natural language prompts and *BioM ELECTRA Base SQuAD 2* as the backbone model (the third model in Table 7), instead of the model with ensemble prediction, as this last model has slower inference times due to the use of multiple prompts for the same entities. We retrain these two architectures with subsets of the training dataset and then test them over the test dataset, which remains unchanged. The training is done following the same hyperparameters as in their final training phases, after the HPO processes.

The training datasets for this experiment are subsets of 100, 200, 300, 400, 500, 600, 700 and 800 abstracts of the original dataset. We train versions of both the sequence-tagging model and the QA model for each of the subsets and then made predictions using the test dataset. For those subsets, we make sure that the instances that are inside the test split were not present, to ensure that no training was done over them and that we were testing the actual prediction capabilities of the models.

Figure 21 shows the training process for both approaches, with the QA model being shown in blue, and the sequence-tagging models

being shown in orange. We can observe how each QA model is really similar to the others, no matter the amount of training data that they use, and that they converge in fewer training epochs than the sequence taggers. These models show a bigger difference in performance depending on the amount of data available for training while needing more training cycles to converge, but they reach higher F1 Scores.
Furthermore, Figure 22 shows the comparison in test F1 Score of the two selected models from both the QA and the sequence-tagging approach, compared by training size. Our hypothesis, as the question-answering model is stable in its performance, regardless of its training size, but the sequence-tagging model starts performing better than the QA model from a training size of 500 abstracts.

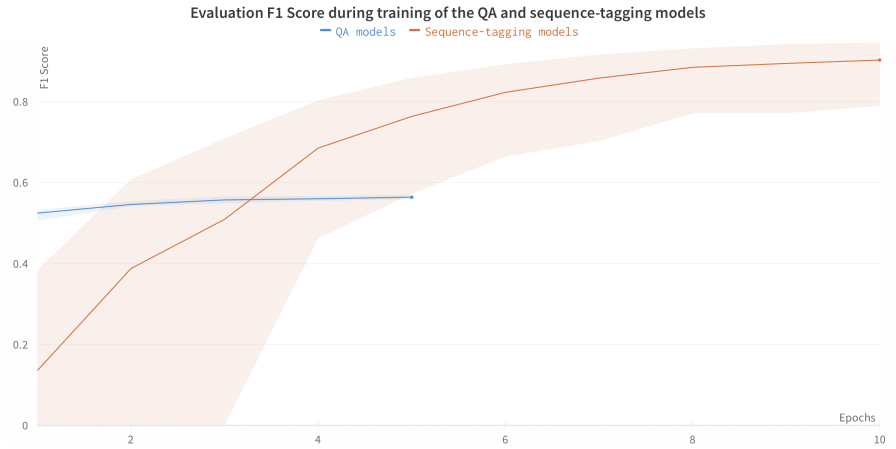


Figure 21: Validation F1 Score during training for the QA and the sequence-tagging models. The colored line represents the median value, and the colored spectrum represent the maximum-minimum intervals.

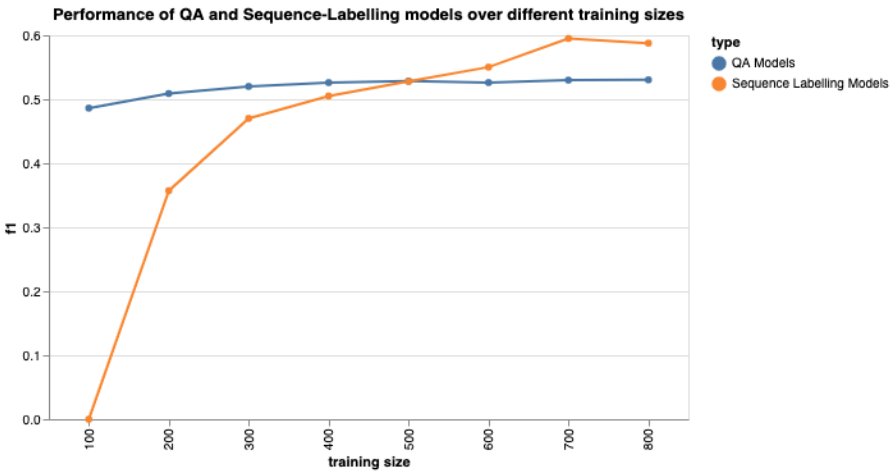


Figure 22: Test F1 Score for the QA and the sequence-tagging models, by training size.

Even though the question-answering model has lower performance, we can see that it can be a more convenient approach in a few-shot learning scenario, as it obtains good performance even if it has been trained with only 100 abstracts (note that, on average, each abstract has 17 entities, so it would be a training corpus of 1700 PICO entities). But, with the amount of data at our disposal, the sequence-tagger is a better choice, as there is almost a difference of 0.1 testing F1 Score.

5.5 ENSEMBLE OF CLASSIFIERS

The results obtained comparing the sequence-tagging and the QA models suggest that QA behaves better in training scenarios with smaller datasets, and NER obtain a solid accuracy with sufficient data. As observed in the data exploration, there is a high class imbalance in the PICO entities. For those reasons, we theorise that an ensemble of classifiers that uses the best sequence-tagging model for some entities and the best extractive question-answering model for others can improve the performance of the best-performing model. The sequence tagger should aim to predict those classes with the highest frequency, and the QA model should perform better on the more rare entities.
To build the ensemble of classifiers, we run both models over all the classes, and analysed for which classes the QA model was outperforming the sequence tagger. We use the same models used in the training size analysis:

- **Sequence taggers**: for the sequence-tagging model, we choose the best model obtained, as seen in Table 4, Longformer Base)
- **QA models**: instead of picking the best-performing model, we use the model with handcrafted natural language prompts and *BioM ELECTRA Base SQuAD 2* as the backbone model (the third model in Table 7), because the best-performing model has a bigger training dataset size and higher inference times.

5.5.1 *Transforming QA predictions into sequence-tagging predictions*

In order to compare the performance of both a QA model and a sequence tagger, we need to transform both predictions into the same prediction standard. The standard used for the QA predictions is the SQuAD standard, as seen in Listings 21 and 22 , and the standard used for the sequence-tagging model is the SeqEval standard, seen in Listing 23. SeqEval, unlike SQuAD, offers information about the accuracy, the recall and the micro-averaged F1 Score per class, so we decide to transform the QA predictions, based on the SQuAD standard, into SeqEval.

```
[{'score': 0.893930196762085,
'start': 643,
'end': 670,
'answer': 'patients with breast cancer'},
228946403]
```

Listing 21: SQuAD standard for predictions.

```
{'answers': {'answer_start': [643], 'text': ['patients with
    breast cancer']},
 'id': '228946403'}
```

Listing 22: SQuAD standard for references.

```
predictions = [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC',
    'O'], ['B-PER', 'I-PER', 'O']]
references = [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'],
    ['B-PER', 'I-PER', 'O']]
```

Listing 23: SeqEval standard for predictions and references.

To make this transformation, we follow these steps:

1. Starting from a QA prediction, we group all the predictions that belong to the same abstract
2. Afterwards, we use these predictions, that contain the index of the first character and the text, to mark all the parts of the text that are part of the several predictions that belong to an abstract.
3. We tokenize the text, making sure that the marks that we did to mark the different predictions stay untokenized, so they can be found afterwards.
4. We transform all of the tokens that are not marked into the NER-like token O, and the tokens that represent a prediction with the token of the corresponding PICO entity.

After this process, we can obtain the NER-like predictions that are suitable for the SeqEval standard. The same process is followed for transforming both the predictions and the references, and it is performed after the question-answering model is done with the inference. The code used for transforming both the predictions and the references can be found in the Appendix, in Listings 26 and 27.

Model	PICO Entity	Longformer Sequence Tagger	QA	Better approach	Best F1 Score
F1 Scores per Class	total-participants	0.90	0.75	Sequence-tagger	0.90
	intervention-participants	0.80	0.64	Sequence-tagger	0.80
	control-participants	0.80	0.72	Sequence-tagger	0.80
	age	0.55	0.72	QA	0.72
	eligibility	0.63	0.49	Sequence-tagger	0.63
	ethnicity	0.82	0.91	QA	0.91
	condition	0.61	0.70	QA	0.70
	location	0.60	0.69	QA	0.69
	intervention	0.53	0.65	QA	0.65
	control	0.63	0.67	QA	0.67
	outcome	0.63	0.13	Sequence-tagger	0.63
	outcome-measure	0.69	0.29	Sequence-tagger	0.69
	iv-bin-abs	0.77	0.42	Sequence-tagger	0.77
	cv-bin-abs	0.83	0.51	Sequence-tagger	0.83
	iv-bin-percent	0.75	0.32	Sequence-tagger	0.75
	cv-bin-percent	0.75	0.33	Sequence-tagger	0.75
	iv-cont-mean	0.48	0.39	Sequence-tagger	0.48
	cv-cont-mean	0.66	0.45	Sequence-tagger	0.66
	iv-cont-median	0.72	0.43	Sequence-tagger	0.72
	cv-cont-median	0.67	0.48	Sequence-tagger	0.67
	iv-cont-sd	0.81	0.48	Sequence-tagger	0.81
	cv-cont-sd	0.71	0.48	Sequence-tagger	0.71
	iv-cont-q1	0	0.00	QA	0.00
	cv-cont-q1	0	0.00	QA	0.00
	iv-cont-q3	0	0.00	QA	0.00
	cv-cont-q3	0	0.00	QA	0.00
Micro-average F1		**0.685760825**	**0.407883931**		**0.701634386**

Table 8: Test F1 Scores of the Sequence-Tagging model, the QA model, and the Ensembler of Classifiers that combines the two, per class.

5.5.2 *Results*

The test F1 Score per class using each model can be seen in Table 8. Figure 23 shows which classes are better classified with the sequence-tagging model (colored in blue) and with the question answerer (colored in orange). With the results obtained with the sequence-tagger and the QA model, we choose which categories to predict with each of the models. Afterwards, we create the ensemble of classifiers using the insight gained, we recalutate the test F1 values, and we compute the micro-averaged test F1 Score, which surpases the result of the sequence-tagging Longformer-based model. Therefore, we obtain the best-performing model of the thesis, by combining the sequence-tagging and extractive question answering.
After analyzing which entities are better predicted when using the question-answering model in Figure 23, it can be seen that the QA model outperforms the sequence tagger with some PICO entities that are around 1000 entities or below, and that are closer to human language (all of them were chosen to be augmented with the generative prompts in the Ensemble prediction on subsection 5.3.5). Therefore, the QA model is behaving as expected, in low-instance scenarios, and with entities that can be easily understood by the pretrained model.

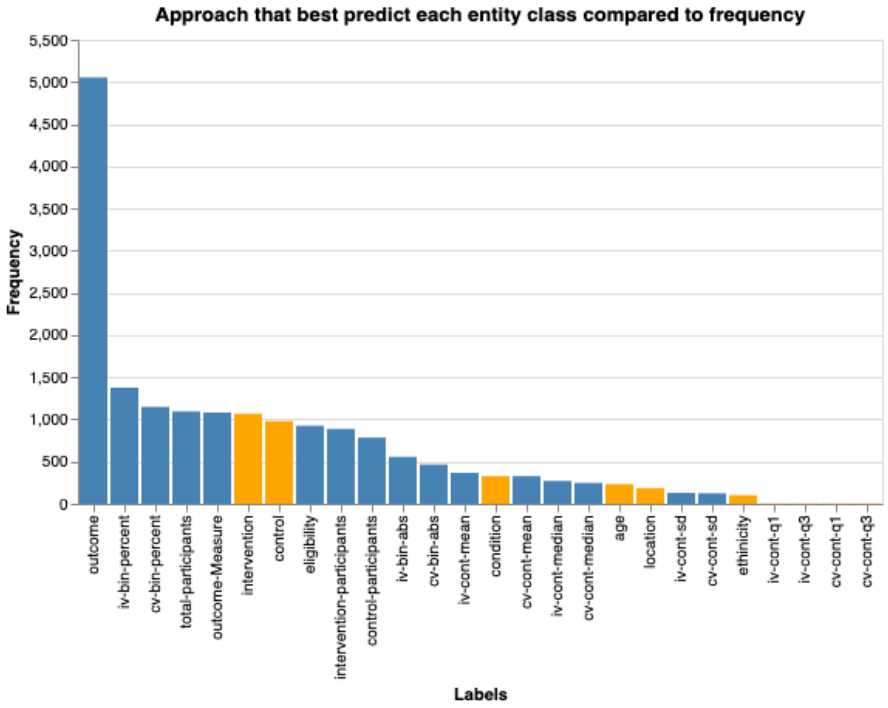


Figure 23: Approach that best predict each entity, compared to the frequency. Entities in blue are better predicted by the sequence-tagging model, and entities in orange are better predicted by the question-answering model.

6 DISCUSSION & CONCLUSION

This last section serves as a summary of the work that has been done in this thesis, including an overview and a discussions of the results that were obtained, the overall conclusions of the whole project, and the future line of work that is suggested.

6.1 SUMMARY & DISCUSSION OF RESULTS

The objective of this thesis was to research NLP pipelines capable of extracting information from medical publications, and two main approaches were suggested: a token classification approach focused on tagging each of the tokens to locate sequences containing relevant information, and a text classification approach, using the extractive question answering technique. Sequence-tagging models were considered to be the classical technique used for this kind of problem, offering consistent results when enough data is available for training. On the other hand, extractive question answerers could be a better solutions on few-shot learning scenarios, due to the way they analyze the relationship between the question and the abstract, without treating each abstract as an independent unit.
The first step taken was a data analysis of the PICO dataset. The corpus consists on 1011 abstracts from breast cancer publications that were manually annotated into 26 different sub-categories, also referred to as PICO entities. A high class imbalance between the entities was found, with one of the categories being the 30% of the total dataset, 10 categories having around 1500 and 500 instances, and the rest of them having less than 500 instances. Abstracts were 1916 characters long in average, with some outliers being from 3000 to 4000 characters long. A high density of PICO entities per abstract was also found.
The first approach to be carried out and analyzed was sequence-tagging. We started the experimentation phase by producing a baseline model with backbone models pretrained over general text, focusing on obtaining a working model from which we could iterate. Afterwards, we experimented with backbone models pretrained over bio-medical data that relied on truncation, as their limit of input tokens were lower than the tokenized input data. These models were compared to Longformer models, which were able to process the whole input without truncation. Both Longformers trained with general data and Longformers pretrained with clinical and biomedical data were compared. The best model approach for the sequence-

tagging models was the regularly-trained Longformer model, due to its long input size and the fact that the PICO dataset has enough training instances for the model to correctly learn how to find PICO entities.

From there, we shifted the scope of our experimentation to extractive question-answering pipelines. A similar procedure was followed, and a first baseline model was developed with limited hyperparameter optimization. Afterwards, we handcrafted prompts close to the human language, and combined them with the abstracts to make the predictions. Combining these prompts with backbone models that have enough number of training parameters pushed the performance of the baseline, and so we used generative models to obtain more prompts per entity and perform a technique that we called *ensemble prediction*[1]. After hyperparameter optimization, this last technique was the best choice inside this approach, but was still around 0.15 test F1 Score points below the best sequence-tagging model (these comparisons are done using micro-averaged F1 Score). In order to obtain validation metrics during training, special functions were developed to transforms predictions into text spans.

This difference between both approaches led to an in-depth comparison between both, focusing on the size of the training dataset as the most important factor. We trained both QA and sequence-tagging models over subsets of the training dataset, and compared their performance over the same test dataset, and we observe two different behaviors:

- The extractive question-answering model does not loose performance when training with limited data, and it converges must faster during training
- The sequence-tagging model only outperforms the QA model when it trains with 500 abstracts or more, and it converges slower during training.

The leap in performance between the two approaches is significant when there is sufficient training data, but the difference is also big when we have low training sizes. From these results, we concluded that the QA model is a better choice for few-shot learning scenarios, but sequence-tagging is more robust on the PICO dataset from a training size of 500 abstracts onward.

As the final phase of the experimentation, and after the insight obtained, we theorized that performance could be pushed forward by combining both approaches. The QA approach could work better than the sequence-tagging approach for those classes that are underpopulated. For these experiments, we firstly developed a method for

1 Not to be confused with the Ensemble of Classifier that was later developed, which combined both the sequence-tagging and the extractive question answering approaches.

Approach	Backbone Model	Test F1 Score	Observations
Sequence-tagging	Longformer Base	0.6886	
Question Answering	BioM ELECTRA Base 2	0.5495	Model using natural language prompts with ensemble prediction
Ensemble of Classifiers	Longformer Base & BioM ELECTRA Base 2	0.7016	Ensemble made with the best sequence-tagging model and the QA model that uses natural language prompts, but no ensemble prediction.

Table 9: Summary with the best models of each section and their test F1 Score.

transforming QA predictions into the sequence-tagging standard. Afterwards, and following the results of a test F1 Score analysis per PICO entity, we obtained a ensemble of classifiers that use a QA model for six classes that have 1000 entities or less, and that are close to the human language, and therefore its prompt works optimally in the QA model. The rest of the entities are predicted with the sequence-tagging model. This ensemble of classifier is the final model made in the experimental phase of the thesis, and it the best-performing model that is made for the PICO dataset.
Table 9 shows the test F1 Score of the best model obtained in each section of the experimentation:

6.2 CONCLUSIONS

From this set of experiments, we can conclude that, for the PICO dataset, which has a high amount of training data, a sequence-tagging model, using backbone models pretrained over biomedical data, is the best-performing approach. A pipeline that yielded an F1 Score of 0.69 over the test dataset was obtained using a sequence tagger, while the best QA model obtained a test F1 Score of 0.55.
Sequence-tagging model are more suited for scenarios with sufficient amount of training instances, while extractive Question Answering model, due to the way that they process the input data, perform better on scenarios with scarce data availability. QA models also reach convergence faster during training. In the case of the PICO dataset, the sequence taggers outperform the QA pipeline from training sizes of 500 abstracts[2].
An ensemble of classifiers combining both approaches can push the performance of the best sequence-tagging model further. We developed an ensemble of classifiers that uses a QA pipeline for several PICO entities[3] that are scarce in the dataset and close to the human language, and predicts the rest with a sequence-tagging model. This model reaches a test F1 Score of 0.70, which is the highest obtained in this thesis, and therefore is the best approach for the PICO dataset.

2 There is an average of 17 PICO entities per abstract.
3 Those entities are *intervention*, *control*, *condition*, *age*, *location* and *ethnicity*.

6.3 FUTURE WORK

After experimenting over the different approaches covered in this thesis and analyzing the results, the insight gathered was also used to identify those research directions that could be expanded in related future work. Due to the fact that we used fine-tuned models, and there are new backbone language models being released that improve performance and capabilities, the results obtained on the different models will probably also improve in further iterations. However, we have identified several specific approaches that can be taken, that do not require the use of new backbone models.

In the sequence-tagging models, the Longformer architecture has proven to be very efficient, thanks to its ability to process longer textual inputs. However, the need for picking a Longformer as a backbone model for the sequence taggers limits the models that can be chosen. They may be other backbone models that perform better for this current task, or similar ones.

Further research can be done on the natural language prompts used for the extractive question-answering models. In this thesis, we have used the PICO entities as question prompt, we have handcrafted natural language prompts and we have created some with generative models. However, the performance of these models could be improved if research on what is the best way to obtain prompts for each of the entities was conducted, specially given the difference between some entities that are very close to the human language, and others that are very difficult to understand by NLP models.

The ensemble of classifiers approach was made using an *ad-hoc* approach; we decided which entities were being predicted with each of the two models after knowing the performance of each model over all the entities, and therefore, the approach cannot be generalized to other problems or datasets. Further research can be made to know when to use a QA-like model or a sequence-tagging model, analyzing the amount of training instances and other important characteristics of the class, like its closeness to human language or how well it is processed as input by the language models.

New approaches can be tried out, like including reinforcement learning and human feedback in a similar way ChatGPT uses its own variant of Reinforcement Learning [42]. After an initial set of predictions, human experts could rank the quality of the given predictions, or even manually annotate the correct answers when the models are dealing with unlabeled data. That feedback, and potentially new labeled data, can be reintroduced in the training loop, fine-tuning the models in sequential phases. After several iterations of this process, the models should enhance their predicting capabilities, and be more reliable in the extraction of information from medical papers.

Part II

APPENDIX

A

SUPLEMENTARY CODE

A.1 EXACT MATCH AND F1 SCORE DURING TRAINING FOR QA MODELS

```python
class QuestionAnsweringTrainer(Trainer):
    def __init__(self, *args, eval_examples=None,
        post_process_function=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.eval_examples = eval_examples
        self.post_process_function = post_process_function

    def evaluate(
        self,
        eval_dataset=None,
        eval_examples=None,
        ignore_keys=None,
        metric_key_prefix: str = "eval",
    ):
        eval_dataset = self.eval_dataset if eval_dataset is None
            else eval_dataset
        eval_dataloader = self.get_eval_dataloader(eval_dataset)
        eval_examples = self.eval_examples if eval_examples is
            None else eval_examples

        # Temporarily disable metric computation, we will do it
            in the loop here.
        compute_metrics = self.compute_metrics
        self.compute_metrics = None
        eval_loop = (
            self.prediction_loop
            if self.args.use_legacy_prediction_loop
            else self.evaluation_loop
        )
        start_time = time.time()
        try:
            output = eval_loop(
                eval_dataloader,
                description="Evaluation",
                # No point gathering the predictions if there are
                     no metrics, otherwise we defer to
                # self.args.prediction_loss_only
                prediction_loss_only=True if compute_metrics is
                    None else None,
                ignore_keys=ignore_keys,
                metric_key_prefix=metric_key_prefix,
            )
```

```
        finally:
            self.compute_metrics = compute_metrics
        total_batch_size = self.args.eval_batch_size * self.args.
            world_size
        if f"{metric_key_prefix}_jit_compilation_time" in output.
            metrics:
            start_time += output.metrics[f"{metric_key_prefix}
                _jit_compilation_time"]
        output.metrics.update(
            speed_metrics(
                metric_key_prefix,
                start_time,
                num_samples=output.num_samples,
                num_steps=math.ceil(output.num_samples /
                    total_batch_size),
            )
        )
        if (
            self.post_process_function is not None
            and self.compute_metrics is not None
            and self.args.should_save
        ):
            # Only the main node write the results by default
            eval_preds = self.post_process_function(
                eval_examples, eval_dataset, output.predictions
            )
            metrics = self.compute_metrics(eval_preds)

            # Prefix all keys with metric_key_prefix + '_'
            for key in list(metrics.keys()):
                if not key.startswith(f"{metric_key_prefix}_"):
                    metrics[f"{metric_key_prefix}_{key}"] =
                        metrics.pop(key)
            metrics.update(output.metrics)
        else:
            metrics = output.metrics

        if self.args.should_log:
            # Only the main node log the results by default
            self.log(metrics)

        if self.args.tpu_metrics_debug or self.args.debug:
            # tpu-comment: Logging debug metrics for PyTorch/XLA
                (compile, execute times, ops, etc.)
            xm.master_print(met.metrics_report())

        self.control = self.callback_handler.on_evaluate(
            self.args, self.state, self.control, metrics
        )
        return metrics

    def predict(
```

```
        self,
        predict_dataset,
        predict_examples,
        ignore_keys=None,
        metric_key_prefix: str = "test",
    ):
        predict_dataloader = self.get_test_dataloader(
            predict_dataset)

        # Temporarily disable metric computation, we will do it
            in the loop here.
        compute_metrics = self.compute_metrics
        self.compute_metrics = None
        eval_loop = (
            self.prediction_loop
            if self.args.use_legacy_prediction_loop
            else self.evaluation_loop
        )
        start_time = time.time()
        try:
            output = eval_loop(
                predict_dataloader,
                description="Prediction",
                # No point gathering the predictions if there are
                     no metrics, otherwise we defer to
                # self.args.prediction_loss_only
                prediction_loss_only=True if compute_metrics is
                    None else None,
                ignore_keys=ignore_keys,
                metric_key_prefix=metric_key_prefix,
            )
        finally:
            self.compute_metrics = compute_metrics
        total_batch_size = self.args.eval_batch_size * self.args.
            world_size
        if f"{metric_key_prefix}_jit_compilation_time" in output.
            metrics:
            start_time += output.metrics[f"{metric_key_prefix}
                _jit_compilation_time"]
        output.metrics.update(
            speed_metrics(
                metric_key_prefix,
                start_time,
                num_samples=output.num_samples,
                num_steps=math.ceil(output.num_samples /
                    total_batch_size),
            )
        )

        if self.post_process_function is None or self.
            compute_metrics is None:
            return output
```

```
126
127         predictions = self.post_process_function(
128             predict_examples, predict_dataset, output.predictions
                    , "predict"
129         )
130         metrics = self.compute_metrics(predictions)
131
132         # Prefix all keys with metric_key_prefix + '_'
133         for key in list(metrics.keys()):
134             if not key.startswith(f"{metric_key_prefix}_"):
135                 metrics[f"{metric_key_prefix}_{key}"] = metrics.
                        pop(key)
136         metrics.update(output.metrics)
137         return PredictionOutput(
138             predictions=predictions.predictions,
139             label_ids=predictions.label_ids,
140             metrics=metrics,
141         )
```

Listing 24: Custom *QuestionAnsweringTrainer* class used to train QA pipelines and allowing metrics on training.

```
def postprocess_qa_predictions(
    examples,
    features,
    predictions: Tuple[np.ndarray, np.ndarray],
    version_2_with_negative: bool = False,
    n_best_size: int = 20,
    max_answer_length: int = 30,
    null_score_diff_threshold: float = 0.0,
    output_dir: Optional[str] = None,
    prefix: Optional[str] = None,
    log_level: Optional[int] = logging.WARNING,
):
    """
    Post-processes the predictions of a question-answering model
        to convert them to answers that are substrings of the
    original contexts. This is the base postprocessing functions
        for models that only return start and end logits.

    Args:
        examples: The non-preprocessed dataset (see the main
            script for more information).
        features: The processed dataset (see the main script for
            more information).
        predictions (:obj:`Tuple[np.ndarray, np.ndarray]`):
            The predictions of the model: two arrays containing
                the start logits and the end logits respectively.
                 Its
            first dimension must match the number of elements of
                :obj:`features`.
        version_2_with_negative (:obj:`bool`, `optional`,
            defaults to :obj:`False`):
```

```
24            Whether or not the underlying dataset contains
                  examples with no answers.
25        n_best_size (:obj:‘int‘, ‘optional‘, defaults to 20):
26            The total number of n-best predictions to generate
                  when looking for an answer.
27        max_answer_length (:obj:‘int‘, ‘optional‘, defaults to
              30):
28            The maximum length of an answer that can be generated
                  . This is needed because the start and end
                  predictions
29            are not conditioned on one another.
30        null_score_diff_threshold (:obj:‘float‘, ‘optional‘,
              defaults to 0):
31            The threshold used to select the null answer: if the
                  best answer has a score that is less than the
                  score of
32            the null answer minus this threshold, the null answer
                   is selected for this example (note that the
                  score of
33            the null answer for an example giving several
                  features is the minimum of the scores for the
                  null answer on
34            each feature: all features must be aligned on the
                  fact they ‘want‘ to predict a null answer).
35
36            Only useful when :obj:‘version_2_with_negative‘ is :
                  obj:‘True‘.
37        output_dir (:obj:‘str‘, ‘optional‘):
38            If provided, the dictionaries of predictions, n_best
                  predictions (with their scores and logits) and,
                  if
39            :obj:‘version_2_with_negative=True‘, the dictionary
                  of the scores differences between best and null
40            answers, are saved in ‘output_dir‘.
41        prefix (:obj:‘str‘, ‘optional‘):
42            If provided, the dictionaries mentioned above are
                  saved with ‘prefix‘ added to their names.
43        log_level (:obj:‘int‘, ‘optional‘, defaults to ‘‘logging.
              WARNING‘‘):
44            ‘‘logging‘‘ log level (e.g., ‘‘logging.WARNING‘‘)
45    """
46    if len(predictions) != 2:
47        raise ValueError(
48            "‘predictions‘ should be a tuple with two elements (
                  start_logits, end_logits)."
49        )
50    all_start_logits, all_end_logits = predictions
51
52    if len(predictions[0]) != len(features):
53        raise ValueError(
54            f"Got {len(predictions[0])} predictions and {len(
                  features)} features."
```

```
        )

    # Build a map example to its corresponding features.
    example_id_to_index = {k: i for i, k in enumerate(examples["
        id"])}
    features_per_example = collections.defaultdict(list)
    for i, feature in enumerate(features):
        features_per_example[example_id_to_index[feature["id"]]].
            append(i)

    # The dictionaries we have to fill.
    all_predictions = collections.OrderedDict()
    all_nbest_json = collections.OrderedDict()
    if version_2_with_negative:
        scores_diff_json = collections.OrderedDict()

    # Logging.
    logger.setLevel(log_level)
    logger.info(
        f"Post-processing {len(examples)} example predictions
            split into {len(features)} features."
    )

    # Let's loop over all the examples!
    for example_index, example in enumerate(tqdm(examples)):
        # Those are the indices of the features associated to the
             current example.
        feature_indices = features_per_example[example_index]

        min_null_prediction = None
        prelim_predictions = []

        # Looping through all the features associated to the
            current example.
        for feature_index in feature_indices:
            # We grab the predictions of the model for this
                feature.
            start_logits = all_start_logits[feature_index]
            end_logits = all_end_logits[feature_index]
            # This is what will allow us to map some the
                positions in our logits to span of texts in the
                original
            # context.
            offset_mapping = features[feature_index]["
                offset_mapping"]
            # Optional 'token_is_max_context', if provided we
                will remove answers that do not have the maximum
                context
            # available in the current feature.
            token_is_max_context = features[feature_index].get(
                "token_is_max_context", None
            )
```

```
96
97              # Update minimum null prediction.
98              feature_null_score = start_logits[0] + end_logits[0]
99              if (
100                 min_null_prediction is None
101                 or min_null_prediction["score"] >
                        feature_null_score
102             ):
103                 min_null_prediction = {
104                     "offsets": (0, 0),
105                     "score": feature_null_score,
106                     "start_logit": start_logits[0],
107                     "end_logit": end_logits[0],
108                 }
109
110             # Go through all possibilities for the ‘n_best_size‘
                    greater start and end logits.
111             start_indexes = np.argsort(start_logits)[
112                 -1 : -n_best_size - 1 : -1
113             ].tolist()
114             end_indexes = np.argsort(end_logits)[-1 : -
                    n_best_size - 1 : -1].tolist()
115             for start_index in start_indexes:
116                 for end_index in end_indexes:
117                     # Don’t consider out-of-scope answers, either
                             because the indices are out of bounds or
                             correspond
118                     # to part of the input_ids that are not in
                            the context.
119                     if (
120                         start_index >= len(offset_mapping)
121                         or end_index >= len(offset_mapping)
122                         or offset_mapping[start_index] is None
123                         or len(offset_mapping[start_index]) < 2
124                         or offset_mapping[end_index] is None
125                         or len(offset_mapping[end_index]) < 2
126                     ):
127                         continue
128                     # Don’t consider answers with a length that
                            is either < 0 or > max_answer_length.
129                     if (
130                         end_index < start_index
131                         or end_index - start_index + 1 >
                                max_answer_length
132                     ):
133                         continue
134                     # Don’t consider answer that don’t have the
                            maximum context available (if such
                            information is
135                     # provided).
136                     if (
137                         token_is_max_context is not None
```

```
                        and not token_is_max_context.get(str(
                            start_index), False)
                    ):
                        continue

                    prelim_predictions.append(
                        {
                            "offsets": (
                                offset_mapping[start_index][0],
                                offset_mapping[end_index][1],
                            ),
                            "score": start_logits[start_index] +
                                end_logits[end_index],
                            "start_logit": start_logits[
                                start_index],
                            "end_logit": end_logits[end_index],
                        }
                    )
        if version_2_with_negative and min_null_prediction is not
             None:
            # Add the minimum null prediction
            prelim_predictions.append(min_null_prediction)
            null_score = min_null_prediction["score"]

        # Only keep the best `n_best_size` predictions.
        predictions = sorted(
            prelim_predictions, key=lambda x: x["score"], reverse
                =True
        )[:n_best_size]

        # Add back the minimum null prediction if it was removed
            because of its low score.
        if (
            version_2_with_negative
            and min_null_prediction is not None
            and not any(p["offsets"] == (0, 0) for p in
                predictions)
        ):
            predictions.append(min_null_prediction)

        # Use the offsets to gather the answer text in the
            original context.
        context = example["context"]
        for pred in predictions:
            offsets = pred.pop("offsets")
            pred["text"] = context[offsets[0] : offsets[1]]

        # In the very rare edge case we have not a single non-
            null prediction, we create a fake prediction to avoid
        # failure.
        if len(predictions) == 0 or (
```

```
            len(predictions) == 1 and predictions[0]["text"] == "
                "
        ):
            predictions.insert(
                0, {"text": "empty", "start_logit": 0.0, "
                    end_logit": 0.0, "score": 0.0}
            )

        # Compute the softmax of all scores (we do it with numpy
            to stay independent from torch/tf in this file, using
        # the LogSumExp trick).
        scores = np.array([pred.pop("score") for pred in
            predictions])
        exp_scores = np.exp(scores - np.max(scores))
        probs = exp_scores / exp_scores.sum()

        # Include the probabilities in our predictions.
        for prob, pred in zip(probs, predictions):
            pred["probability"] = prob

        # Pick the best prediction. If the null answer is not
            possible, this is easy.
        if not version_2_with_negative:
            all_predictions[example["id"]] = predictions[0]["text
                "]
        else:
            # Otherwise we first need to find the best non-empty
                prediction.
            i = 0
            while predictions[i]["text"] == "":
                i += 1
            best_non_null_pred = predictions[i]

            # Then we compare to the null prediction using the
                threshold.
            score_diff = (
                null_score
                - best_non_null_pred["start_logit"]
                - best_non_null_pred["end_logit"]
            )
            scores_diff_json[example["id"]] = float(
                score_diff
            )  # To be JSON-serializable.
            if score_diff > null_score_diff_threshold:
                all_predictions[example["id"]] = ""
            else:
                all_predictions[example["id"]] =
                    best_non_null_pred["text"]

        # Make ‘predictions‘ JSON-serializable by casting np.
            float back to float.
        all_nbest_json[example["id"]] = [
```

```
            {
                k: (
                    float(v)
                    if isinstance(v, (np.float16, np.float32, np.
                        float64))
                    else v
                )
                for k, v in pred.items()
            }
            for pred in predictions
        ]

        return all_predictions
```

Listing 25: Custom *postprocess_qa_predictions()* function used to transform the predictions of question-answering models to subtrings of original text..

A.2 AUTOMATIC PROMPT-TUNING USING GENERATIVE MODELS

PICO entity	Natural Language Prompts
intervention	- What is the intervention of the study? - What steps were taken as part of this research project that we want to test out? - For this particular trial, how did we intervene and change things compared to not doing anything at all? -How did we manipulate some aspect of the environment for our subjects/participants in order to see if it made any difference or produced noticeable effects? -When conducting this experimentation, did we modify or adjust certain conditions or variables, and if so, how? And what was our reasoning behind those decisions?
condition	- What is the condition of the study? - Which setting or circumstances apply to the current research investigation? - In which context or scenario are the participants experiencing or interacting within the scope of this study? - Given that these results are based on scientific inquiry, under what specific terms or parameters did we carry out this exploration? - Are there special situations or stipulations attached to this experiment that would impact the data gathered from the findings or conclusions drawn?
total-participants	- What is the number of total participants of the study? - What is the exact count of individuals involved in the entire experimental group of this investigation? - What is the sum of participants included in this research? - What is the size of the group of total participants in this study? - What is the full set of contributors across every category during this research.
age	- What is the age of the participants of the study? - How old were the individuals who took part in this research? - What was the age range of those involved in the experiment? - What is the age distribution of the subjects used for the investigation?- Was there any specific criteria regarding participant age for this study?
ethnicity	- What is the ethnicity of the participants of the study? - To which ethnic groups did those involved in the analysis belong? - Were any particular racial backgrounds represented among the investigated populations? - Can you provide details as to the diversity of ethnicities present within your sample set?" - What representation across distinct ethnic groups?"
intervention-participant	- What is the number of the intervention participants of the study? - How many individuals were assigned to receive the intervention treatment in the research project? - How many trial subjects received the experimental therapy as part of the experiment? - How many participants received the treatment or intervention? - What was the sample size of the research trial?"

Table 10: PICO entities and their corresponding prompts. The first prompt of each category is handcrafted, and the rest were automatically generated by a LLaMA model. First table.

PICO entity	Natural Language Prompts
location	- What is the location of the study? - Where was the study conducted? - Can you identify where the site of the study took place? - Whereabouts was the locale of the study being undertaken? - Where did the clinical study take place?
eligibility	- What is the eligibility criteria of the study? - Can you please describe who was screened and excluded from participation due to certain criteria? - Which specific features determined whether someone qualified or not? - Which subgroups of subjects ended up getting enrolled into the study, and why? - How was the process of identifying eligible patients for inclusion in the clinical trial?
control	- What is the control of the study? - How was the experiment controlled so that extraneous variables didn't affect the outcome? - Were there any conditions set in place during the experiment to minimize bias or confounding factors? - How did the investigators address potential confounders or third variables that could have influenced study results? - Which variables remained constant in order to isolate the effect of interest?
outcome	- What is the outcome that was measured on the study? - What are the conclusive observations of the study? - What is the outcome of the study? - What findings did the study produce? - Did the study achieve any significant conclusions or discoveries?"
outcome-Measure	-What is the outcome measure of the study? - In what respect did the study seek to evaluate the effects of the intervention? - By what means was the success of the research evaluated? - What is metric through which the impacts of the trial were determined? - Which measurement techniques were implemented?"
control-participant	- What is the number of the control participants of the study? - Disclose the subject count for the non-intervened group in the current experiment. - What is the quantity of participants enlisted for the placebo arm of the research? - What is the quantity of trial candidates who were not administered the novel medication. - How many test subjects did not receive the treatments or interventions.

Table 11: PICO entities and their corresponding prompts. The first prompt of each category is handcrafted, and the rest were automatically generated by a LLaMA model. Second table.

```
def qa2ner_predictions(predictions, dataset, nlp, hf_tokenizer,
    categories):
    print("Translating QA predictions to NER predictions")

    def group_qa_preds_same_abstract():
        predictions_grouped = {}
        current_id = 0

        for p in predictions:
            if p[3] == current_id and p[3] in predictions_grouped
                .keys():
                predictions_grouped[p[3]][0].append([p[0], p[1].
                    replace("-","_") ])

            if p[3] not in predictions_grouped.keys():
                current_id = p[3]
                predictions_grouped[int(p[3])] = [[[p[0], p[1].
                    replace("-","_") ]], p[4]]

        return predictions_grouped

    def mark_answers(prediction, categories):
        marked_abstract = prediction[1]
        pico_tags = []

        if prediction == []:
            return marked_abstract, pico_tags

        for pred in prediction[0]:
            if f"[{pred[1]}]" not in pico_tags:
                pico_tags.append(f"[{pred[1]}]")

            if pred[1] in categories:
                new_abstract = (
                    marked_abstract[: pred[0]["end"]]
                    + f" [{pred[1]}]"
                    + marked_abstract[pred[0]["end"]]
                    + marked_abstract[pred[0]["end"] + 1 :]
                )

                new_abstract = (
                    new_abstract[: pred[0]["start"]]
                    + f"[{pred[1]}] "
                    + new_abstract[pred[0]["start"]]
                    + new_abstract[pred[0]["start"] + 1 :]
                )
                marked_abstract = new_abstract

        return marked_abstract, pico_tags

    def spacy_tokenization(predictions, pico_tags):

```

```
        # Add special case rule for each pico tag
        for entity in pico_tags:
            special_case = [{ORTH: f"{entity}"}]
            nlp.tokenizer.add_special_case(f"{entity}",
                special_case)

        # Custom tokenizer
        text = predictions
        doc = nlp(text)

        # Flatten list & return
        return [token.text for token in doc]

    def hf_tokenization(pred_tokens, pico_tags):
        hf_tokenizer.add_tokens(pico_tags)
        inputs = hf_tokenizer(
            pred_tokens,
            return_tensors="pt",
            truncation=True,
            is_split_into_words=True,
            max_length=4096,
        )

        inputs.tokens()

        token_list = []

        current_entity = ""
        is_entity = False

        for token in inputs.tokens():
            if token[0] == "[" and token[-1] == "]" and is_entity
                 is False:
                current_entity = token[1:-1]
                is_entity = True
                continue

            elif token[0] == "[" and token[-1] == "]" and
                is_entity is True:
                current_entity = ""
                is_entity = False
                continue

            elif current_entity != "" and is_entity is True:
                token_list.append(current_entity)

            else:
                token_list.append("O")

        return token_list

    # Obtain grouped predictions
```

```
    grouped_predictions = group_qa_preds_same_abstract()

    # Order each list of predictions
    for key, elem in grouped_predictions.items():
        grouped_predictions[key] = elem
        grouped_predictions[key][0] = sorted(
            elem[0], key=lambda d: d[0]["start"], reverse=True
        )

    # Add empty predictions
    for record in dataset:
        if int(record["title"].split(" ")[0]) not in
            grouped_predictions.keys():
            grouped_predictions[int(record["title"].split(" ")
                [0])] = [[], record["context"]]

    # Order dictionary
    grouped_predictions = {key: grouped_predictions[key] for key
        in sorted(grouped_predictions)}

    predictions_tokens = []

    # For each prediction (grouped by abstract)
    for key, elem in tqdm(grouped_predictions.items()):

        # Mark the answer in the text
        marked_abstracts, pico_tags = mark_answers(elem,
            categories)

        # 2-step tokenization on the combined string
        # spaCy
        pred_tokens = spacy_tokenization(marked_abstracts,
            pico_tags)

        # HuggingFace
        predictions_tokens.append(hf_tokenization(pred_tokens,
            pico_tags))

    return predictions_tokens, grouped_predictions
```

Listing 26: Function to transform QA predictions to sequence-tagging predictions.

```
def qa2ner_references(dataset, predictions, nlp, hf_tokenizer,
    categories):
    print("Translating QA references to NER references")

    def group_qa_refs_same_abstract(dataset):
        references_grouped = {}
        current_id = 0

        for record in dataset:

```

```
            int_id = int(str(record["id"])[:8])

            if int_id == current_id and int_id in
                references_grouped.keys():
                references_grouped[int_id][0].append([record["
                    answers"], "&" + record["question"].replace("
                    -","_")])
            if int_id not in references_grouped.keys():
                current_id = int_id
                references_grouped[int_id] = [[[record["answers"
                    ], "&" + record["question"].replace("-","_")
                    ]], record["context"]]


        return references_grouped

    def mark_answers(reference, categories):

        marked_abstract = reference[1]
        pico_tags = []

        for pred in reference[0]:


            if f"[{pred[1]}]" not in pico_tags:
                pico_tags.append(f"[{pred[1]}]")

            if pred[1] in categories:

                text = pred[0]["text"][0]
                answer_start = pred[0]["answer_start"][0]
                answer_end = pred[0]["answer_start"][0] + len(
                    text)
                category = pred[1]

                new_abstract = (
                    marked_abstract[: answer_end]
                    + f" [{category}]"
                    + marked_abstract[answer_end]
                    + marked_abstract[answer_end + 1 :]
                )

                new_abstract = (
                    new_abstract[: answer_start]
                    + f"[{category}] "
                    + new_abstract[answer_start]
                    + new_abstract[answer_start + 1 :]
                )
                marked_abstract = new_abstract

        return marked_abstract, pico_tags

```

```
56      def spacy_tokenization(predictions, pico_tags):
57
58          # Add special case rule for each pico tag
59          for entity in pico_tags:
60              special_case = [{ORTH: f"{entity}"}]
61              nlp.tokenizer.add_special_case(f"{entity}",
                    special_case)
62
63          # Custom tokenizer
64          text = predictions
65          doc = nlp(text)
66
67          # Flatten list & return
68          return [token.text for token in doc]
69
70      def hf_tokenization(pred_tokens, pico_tags):
71          hf_tokenizer.add_tokens(pico_tags)
72          inputs = hf_tokenizer(
73              pred_tokens,
74              return_tensors="pt",
75              truncation=True,
76              is_split_into_words=True,
77              max_length=4096,
78          )
79
80          inputs.tokens()
81
82          token_list = []
83
84          current_entity = ""
85          is_entity = False
86
87          for token in inputs.tokens():
88              if token[0] == "[" and token[-1] == "]" and is_entity
                     is False:
89                  current_entity = token[1:-1]
90                  is_entity = True
91                  continue
92
93              elif token[0] == "[" and token[-1] == "]" and
                    is_entity is True:
94                  current_entity = ""
95                  is_entity = False
96                  continue
97
98              elif current_entity != "" and is_entity is True:
99                  token_list.append(current_entity)
100
101             else:
102                 token_list.append("O")
103
104         return token_list
```

```
105
106     # Obtain grouped predictions
107     grouped_predictions = group_qa_refs_same_abstract(dataset)
108
109     # Add empty predictions
110     for record in dataset:
111         if int(record["title"].split(" ")[0]) not in
                grouped_predictions.keys():
112             grouped_predictions[int(record["title"].split(" ")
                    [0])] = [[], record["context"]]
113
114     new_grouped_predictions = {}
115     # Erasing all the references that are not in predictions
116     for key, elem in grouped_predictions.items():
117         if key in predictions.keys():
118             new_grouped_predictions[key] = elem
119     grouped_predictions = new_grouped_predictions
120
121     # Order dictionary
122     grouped_predictions = {key: grouped_predictions[key] for key
            in sorted(grouped_predictions)}
123
124
125     reference_tokens = []
126
127     # For each prediction (grouped by abstract)
128     for key, elem in tqdm(grouped_predictions.items()):
129
130         # Mark the answer in the text
131         marked_abstracts, pico_tags = mark_answers(elem,
                categories)
132
133
134         # 2-step tokenization on the combined string
135         # spaCy
136         pred_tokens = spacy_tokenization(marked_abstracts,
                pico_tags)
137
138         # HuggingFace
139         reference_tokens.append(hf_tokenization(pred_tokens,
                pico_tags))
140
141
142     return reference_tokens
```

Listing 27: Function to transform QA references to sequence-tagging references.

BIBLIOGRAPHY

[1] Martín Abadi et al. "TensorFlow: A system for large-scale machine learning." In: (May 2016). URL: http://arxiv.org/abs/1605.08695.

[2] Adnan Akhundov, Dietrich Trautmann, and Georg Groh. "Sequence Labeling: A Practical Approach." In: (Aug. 2018). URL: http://arxiv.org/abs/1808.03926.

[3] Sultan Alrowili and Vijay Shanker. "BioM-Transformers: Building Large Biomedical Language Models with BERT, ALBERT and ELECTRA." In: *Proceedings of the 20th Workshop on Biomedical Language Processing*. Online: Association for Computational Linguistics, June 2021, pp. 221–227. URL: https://www.aclweb.org/anthology/2021.bionlp-1.24.

[4] Emily Alsentzer, John R. Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, and Matthew B. A. McDermott. *Publicly Available Clinical BERT Embeddings*. 2019. arXiv: 1904.03323 [cs.CL].

[5] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL].

[6] Emily M. Bender and Alexander Koller. "Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data." In: (July 2020), pp. 5185–5198. DOI: 10.18653/V1/2020.ACL-MAIN.463. URL: https://aclanthology.org/2020.acl-main.463.

[7] Daniele Bonadiman, Anjishnu Kumar, and Arpit Mittal. *Large Scale Question Paraphrase Retrieval with Smoothed Deep Metric Learning*. 2019. arXiv: 1905.12786 [cs.CL].

[8] Rohit Borah, Andrew W Brown, Patrice L Capers, and Kathryn A Kaiser. "Analysis of the time and workers needed to conduct systematic reviews of medical interventions using data from the PROSPERO registry." In: *Open* 7 (2017), p. 12545. DOI: 10.1136/bmjopen-2016. URL: www.R-project.org/;.

[9] Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. *Large Language Models in Machine Translation*. 2007, pp. 858–867.

[10] Tom B. Brown et al. "Language Models are Few-Shot Learners." In: (May 2020). URL: http://arxiv.org/abs/2005.14165.

[11] Thomas C. Chalmers, Harry Smith, Bradley Blackburn, Bernard Silverman, Biruta Schroeder, Dinah Reitman, and Alexander Ambroz. "A method for assessing the quality of a randomized control trial." In: *Controlled Clinical Trials* 2.1 (1981), pp. 31–49. ISSN: 0197-2456. DOI: https://doi.org/10.1016/0197-2456(81)90056-8. URL: https://www.sciencedirect.com/science/article/pii/0197245681900568.

[12] D J Cook, Cynthia Mulrow, and bhaynes@mcmaster.ca Haynes. "Systematic Reviews: Synthesis of Best Evidence for Clinical Decisions." In: *Annals of internal medicine* 126 (Apr. 1997), pp. 376–380. DOI: 10.7326/0003?4819?126?5?199703010?00006.

[13] José Cã, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. *Spanish Pre-Trained BERT Model and Evaluation Data*. URL: https://github.com/josecannete/spanish-corpora.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova Google, and A I Language. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proceedings of the 2019 Conference of the North* (2019), pp. 4171–4186. DOI: 10.18653/V1/N19-1423. URL: https://aclanthology.org/N19-1423.

[15] William Fedus, Barret Zoph, and Noam Shazeer. "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity." In: *Journal of Machine Learning Research* 23 (Jan. 2021), pp. 1–40. ISSN: 15337928. URL: https://arxiv.org/abs/2101.03961v3.

[16] Tianyu Gao, Adam Fisch, and Danqi Chen. "Making Pre-trained Language Models Better Few-shot Learners." In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3816–3830. DOI: 10.18653/v1/2021.acl-long.295. URL: https://aclanthology.org/2021.acl-long.295.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] Gordon Guyatt et al. "Evidence-Based Medicine: A New Approach to Teaching the Practice of Medicine." In: *JAMA* 268 (17 Nov. 1992), pp. 2420–2425. ISSN: 0098-7484. DOI: 10.1001/jama.1992.03490170092032. URL: https://doi.org/10.1001/jama.1992.03490170092032.

[19] Edward L. Hannan. "Randomized Clinical Trials and Observational Studies: Guidelines for Assessing Respective Strengths and Limitations." In: *JACC: Cardiovascular Interventions* 1.3

(2008), pp. 211–217. ISSN: 1936-8798. DOI: https://doi.org/10.1016/j.jcin.2008.01.008. URL: https://www.sciencedirect.com/science/article/pii/S1936879808001702.

[20] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2021. arXiv: 2006.03654 [cs.CL].

[21] *HuggingFace Transformer Tasks*. URL: https://huggingface.co/tasks.

[22] Kirsty James and Katharine A Rimes. "Mindfulness-based cognitive therapy versus pure cognitive behavioural self-help for perfectionism: A pilot randomised study." In: *Mindfulness* 9 (2018), pp. 801–814.

[23] Siddhartha R. Jonnalagadda, Pawan Goyal, and Mark D. Huffman. "Automating data extraction in systematic reviews: A systematic review." In: *Systematic Reviews* 4 (1 June 2015). ISSN: 20464053. DOI: 10.1186/s13643-015-0066-7.

[24] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. "Neural Architectures for Named Entity Recognition." In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 260–270. DOI: 10.18653/v1/N16-1030. URL: https://aclanthology.org/N16-1030.

[25] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut, and Google Research. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations." In: (Sept. 2019). URL: https://arxiv.org/abs/1909.11942v6.

[26] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. "Efficient BackProp." In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50. ISBN: 978-3-540-49430-0. DOI: 10.1007/3-540-49430-8_2. URL: https://doi.org/10.1007/3-540-49430-8_2.

[27] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. "BioBERT: a pre-trained biomedical language representation model for biomedical text mining." In: *Bioinformatics* 36.4 (2019). Ed. by Jonathan Wren, pp. 1234–1240. DOI: 10.1093/bioinformatics/btz682. URL: https://doi.org/10.1093\%2Fbioinformatics\%2Fbtz682.

[28] Yikuan Li, Ramsey M. Wehbe, Faraz S. Ahmad, Hanyin Wang, and Yuan Luo. *Clinical-Longformer and Clinical-BigBird: Transformers for long clinical sequences*. 2022. arXiv: 2201.11838 [cs.CL].

[29] Andy T. Liu, Wei Xiao, Henghui Zhu, Dejiao Zhang, Shang-Wen Li, and Andrew Arnold. "QaNER: Prompting Question Answering Models for Few-shot Named Entity Recognition." In: (Mar. 2022). URL: http://arxiv.org/abs/2203.01543.

[30] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing." In: *ACM Computing Surveys* 55 (9 July 2021), pp. 1–35. ISSN: 0360-0300. DOI: 10.48550/arxiv.2107.13586. URL: https://arxiv.org/abs/2107.13586v1.

[31] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." In: (July 2019). URL: http://arxiv.org/abs/1907.11692.

[32] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. "BioGPT: Generative Pre-trained Transformer for Biomedical Text Generation and Mining." In: (Oct. 2022). DOI: 10.1093/bib/bbac409. URL: http://arxiv.org/abs/2210.10341http://dx.doi.org/10.1093/bib/bbac409.

[33] Iain J. Marshall and Byron C. Wallace. *Toward systematic review automation: A practical guide to using machine learning tools in research synthesis*. July 2019. DOI: 10.1186/s13643-019-1074-9.

[34] Oren Melamud, Jacob Goldberger, and Ido Dagan. "context2vec: Learning Generic Context Embedding with Bidirectional LSTM." In: *CoNLL 2016 - 20th SIGNLL Conference on Computational Natural Language Learning, Proceedings* (2016), pp. 51–61. DOI: 10.18653/V1/K16-1006. URL: https://aclanthology.org/K16-1006.

[35] Faith Wavinya Mutinda, Kongmeng Liew, Shuntaro Yada, Shoko Wakamiya, and Eiji Aramaki. *PICO Corpus: A Publicly Available Corpus to Support Automatic Data Extraction from Biomedical Literature*. 2022. URL: https://aclanthology.org/2022.wiesp-1.4.

[36] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: (Dec. 2019). URL: http://arxiv.org/abs/1912.01703.

[37] Kate Pearce, Tiffany Zhan, Aneesh Komanduri, and Justin Zhan. "A Comparative Study of Transformer-Based Language Models on Extractive Question Answering." In: (Oct. 2021). URL: http://arxiv.org/abs/2110.03142.

[38] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* (2014), pp. 1532–1543. DOI: 10.3115/V1/D14-1162. URL: https://aclanthology.org/D14-1162.

[39] Ralph Peterli, Bettina Karin Wölnerhanssen, Diana Vetter, Philipp Nett, Markus Gass, Yves Borbély, Thomas Peters, Marc Schiesser, Bernd Schultes, Christoph Beglinger, et al. "Laparoscopic sleeve gastrectomy versus Roux-Y-gastric bypass for morbid obesity—3-year outcomes of the prospective randomized Swiss Multicenter Bypass Or Sleeve Study (SM-BOSS)." In: *Annals of surgery* 265.3 (2017), p. 466.

[40] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations." In: *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1 (Feb. 2018), pp. 2227–2237. DOI: 10.18653/v1/n18-1202. URL: https://arxiv.org/abs/1802.05365v2.

[41] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. *Language Models as Knowledge Bases?* 2019. arXiv: 1909.01066 [cs.CL].

[42] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. URL: https://github.com/codelucas/newspaper.

[43] Pranav Rajpurkar, Robin Jia, and Percy Liang. *Know What You Don't Know: Unanswerable Questions for SQuAD*. 2018. arXiv: 1806.03822 [cs.CL].

[44] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250 [cs.CL].

[45] David L Sackett. *Evidence-Based Medicine*. 1997, pp. 3–5.

[46] Erik F Sang and Fien De Meulder. "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition." In: *arXiv preprint cs/0306050* (2003).

[47] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." In: (Oct. 2019). URL: https://arxiv.org/abs/1910.01108v4.

[48] Timo Schick and Hinrich Schütze. "It's not just size that matters: Small language models are also few-shot learners." In: *arXiv preprint arXiv:2009.07118* (2020).

[49] Fabrizio Sebastiani. "Machine Learning in Automated Text Categorization." In: *ACM Comput. Surv.* 34.1 (2002), 1–47. ISSN: 0360-0300. DOI: 10.1145/505282.505283. URL: https://doi.org/10.1145/505282.505283.

[50] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. *Bidirectional Attention Flow for Machine Comprehension*. 2018. arXiv: 1611.01603 [cs.CL].

[51] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism." In: (Sept. 2019). URL: https://arxiv.org/abs/1909.08053v4.

[52] Peter Sleight. "The HOPE Study (Heart Outcomes Prevention Evaluation)." In: *Journal of the Renin-Angiotensin-Aldosterone System* 1.1 (2000), pp. 18–20. DOI: 10.3317/jraas.2000.002. eprint: https://doi.org/10.3317/jraas.2000.002. URL: https://doi.org/10.3317/jraas.2000.002.

[53] Ripple Talati, Diana M Sobieraj, Sagar S Makanji, Olivia J Phung, and Craig I Coleman. "The comparative efficacy of plant sterols and stanols on serum lipids: a systematic review and meta-analysis." In: *Journal of the American Dietetic Association* 110.5 (2010), pp. 719–726.

[54] Erik F. Tjong Kim Sang and Sabine Buchholz. "Introduction to the CoNLL-2000 Shared Task Chunking." In: *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*. 2000. URL: https://aclanthology.org/W00-0726.

[55] Erik F. Tjong Kim Sang and Fien De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition." In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*. 2003, pp. 142–147. URL: https://aclanthology.org/W03-0419.

[56] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[57] *Turing-NLG: A 17-billion-parameter language model by Microsoft - Microsoft Research*. URL: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.

[58] Stalin Varanasi, Saadullah Amin, and Günter Neumann. *AutoEQA: Auto-Encoding Questions for Extractive Question Answering*, pp. 4706–4712. URL: https://github.com/allenai/allennlp.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need." In: *Advances in Neural Information Processing Systems* 2017-December (June 2017), pp. 5999–6009. ISSN: 10495258. DOI: 10.48550/arxiv.1706.03762. URL: https://arxiv.org/abs/1706.03762v5.

[60] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2019. arXiv: 1804.07461 [cs.CL].

[61] Haohan Wang and Bhiksha Raj. "On the Origin of Deep Learning." In: (Feb. 2017). URL: http://arxiv.org/abs/1702.07800.

[62] Paul J. Werbos. "Generalization of backpropagation with application to a recurrent gas market model." In: *Neural Networks* 1.4 (1988), pp. 339–356. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90007-X. URL: https://www.sciencedirect.com/science/article/pii/089360808890007X.

[63] *What is Question Answering?* URL: https://huggingface.co/tasks/question-answering.

[64] Thomas Wolf et al. *Transformers: State-of-the-Art Natural Language Processing*. URL: https://github.com/huggingface/.

[65] Yi Yang and Arzoo Katiyar. "Simple and effective few-shot named entity recognition with structured nearest neighbor learning." In: *arXiv preprint arXiv:2010.02405* (2020).

[66] Tong Yu and Hong Zhu. *Hyper-Parameter Optimization: A Review of Algorithms and Applications*. 2020. arXiv: 2003.05689 [cs.LG].

[67] Zhuosheng Zhang, Junjie Yang, and Hai Zhao. *Retrospective Reader for Machine Reading Comprehension*. 2020. arXiv: 2001.09694 [cs.CL].

DECLARATION

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgement, the work presented is entirely my own.

Aarhus, June 2023

Ignacio Talavera Cepeda

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of June 9, 2023 (`classicthesis`).