Universidade do Minho
Escola de Engenharia

Henrique Manuel Palmeira Pereira

Avoiding Question-Answering Congestion on Health Services using Chatbots

December, 2021

Universidade do Minho
Escola de Engenharia

Henrique Manuel Palmeira Pereira

Avoiding Question-Answering Congestion on Health Services using Chatbots

Master's Dissertation
Integrated Master's in Informatics Engineering

Work supervised by
Professor Doutor Joaquim Macedo
Professora Doutora Olga Craveiro

December, 2021

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Braga, 28/12/2021

(Location) (Date)

[illegible]

(Henrique Manuel Palmeira Pereira)

Acknowledgements

First of all, I would like to thank both my supervisors, Professor Doutor Joaquim Macedo and Professora Doutora Olga Craveiro, for all the patience and support provided during this year.

To my parents, Mónica and Manuel, for always providing me everything I needed to complete my education and for making me the man I am today. Also, to my brother Eduardo for teasing me that he would finish his dissertation first than me.

To Sofia Cruz, my love, for bringing me up in my lowest moments and for always helping me find my motivation.

To my friends Braga, Caçador, Joãozinho, Pedro and Pedro, for sharing the same pains as me and for being the backbone of my university adventure.

To my Uphold colleagues from the Data team, mainly Yuliana, Guilherme, Zé and Mário, for letting me discuss my thesis with them and for always being ready to help.

To all my other friends that I do not mention personally, for all the good times that made this trip so beautiful.

Abstract

The proliferation of social networks presents a significant amount of fake news and fake information every day and every second. The COVID-19 pandemic confirms this situation. The general ignorance of this disease causes the spreading of misleading information, harming people's lives and governments' actions to contain it.

To fight this infodemic, the populations resorted to the health services' phone lines, congesting them with questions, most of them repeated among different individuals and locations. A chatbot for COVID-19-related questions would redirect this workload from the health services, mitigating such congestion. This chatbot should work for both the English and Portuguese languages.

This work provides a background overview about web crawlers, information processing and chatbot development, which are the three components of the application. A systematic literature review was done to provide an analysis of the existing literature on the mentioned thematics.

The application presented in this work consists of three main modules: a web crawler, using the *ACHE* crawler application, which downloads the web pages from the trustworthy sources; a text processor, that parses the web pages and indexes them according to their language to the respective ElasticSearch index; and a chatbot component, composed by a fine-tuned BERT model with the SQuAD 2.0 dataset and a web interface that queries the ElasticSearch indexes for the most relevant pages and extracts the answers to the given questions by the users. To comply with the English and Portuguese requirement, two sets of reliable sources were defined (one for each language) and a translated version of SQuAD 1.1 dataset was used to train the Portuguese BERT model. The chatbot queries the correct model using the web browser's defined language.

Our system was evaluated using a set of COVID-19 QA pairs extracted from the United Nations website, and the obtained results are described in this work. These were far from the desirable outcomes, so some improvements were applied to the crawler and to the ElasticSearch indexes. However the results were still not satisfactory, requiring a set of future modifications that are presented in this work.

Keywords: chatbot, COVID-19, information processing, natural language processing, web crawling

Resumo

Com a proliferação das redes sociais, um número significativo de *fake news* é disponibilizado às pessoas todos os dias, a cada segundo. Isto foi confirmado durante a pandemia da COVID-19, onde um desconhecimento geral da doença causou a difusão de informação enganosa, colocando em risco a vida das pessoas e as ações governamentais que visavam o controlo da doença.

Para combater esta infodemia, as populações recorreram às linhas telefónicas dos serviços de saúde nacionais, congestionando-as com questões muitas vezes repetidas. Com o intuito de mitigar este congestionamento, um *chatbot* para a COVID-19 ajudaria a redirecionar esta carga de trabalho dos serviços de saúde para a aplicação. Este *chatbot* deve suportar as línguas Portuguesa e Inglesa.

Este trabalho apresenta uma visão geral acerca de *web crawlers*, de processamento de informação e de desenvolvimento de *chatbots*. Uma revisão sistemática da literatura foi conduzida com o intuito de apresentar uma análise da literatura existente. A aplicação apresentada neste trabalho consiste em três componentes principais: um *web crawler*, usando a aplicação *ACHE*, que descarrega as páginas *web* das fontes confiáveis; um componente de processamento de texto, que processa as páginas e as indexa de acordo com a sua língua no respetivo índice de ElasticSearch; e um *chatbot*, composto por um modelo BERT treinado e refinado com o *dataset* SQuAD 2.0 e uma interface *web*, que pesquisa no ElasticSearch as páginas mais relevantes e extrai daí as respostas para as perguntas dos utilizadores. Para satisfazer o requisito das duas línguas, dois conjuntos de páginas confiáveis foram definidos (um para cada língua), e uma versão traduzida do SQuAD 1.1 foi utilizada para treinar o modelo BERT em Português. O *chatbot* questiona o modelo correto consoante a língua configurada no *browser* utilizado.

O sistema foi avaliado usando um conjunto real de perguntas e respostas sobre COVID-19, sendo apresentados neste trabalho os resultados obtidos. Estes ficaram longe do desejado, pelo que algumas melhorias foram aplicadas ao sistema. Porém, os resultados permaneceram ainda assim insatisfatórios, necessitando de um conjunto de futuras alterações que são apresentadas neste trabalho.

Palavras-chave: chatbot, COVID-19, processamento de informação, processamento de linguagem natural, web crawling

Contents

List of Figures

List of Tables

Listings

Acronyms

ACHE	Adaptive Crawler for Hidden-Web Entries
ANN	Artificial Neural Networks
BERT	Bidirectional Encoder Representation from Transformers
CBR	Case-based Reasoning
CDC	Centers for Disease Control and Prevention
COVID-19	Coronavirus Disease
CRF	Conditional Random Field
DNN	Deep Neural Network
DOM	Document Object Model
ECDC	European Center for Disease Prevention and Control
FAQ	Frequently Asked Questions
HITS	Hyperlink-Induced Topic Search
HMM	Hidden Markov Model
HTML	HyperText Markup Language
IoT	Internet of Things
IR	Information Retrieval
LDA	Latent Dirichlet Allocation
LSI	Latent Semantic Indexing
LSTM	Long Short Term Memory
MLM	Masked Language Modeling
MM	Multinomial Mixture

NHS	National Health Service
NLP	Natural Language Processing
NLU	Natural Language Understanding
NPMI	Normalized Pointwise Mutual Information
NSP	Next Sentence Prediction
PLSA	Probabilistic Latent Semantic Analysis
QA	Question Answering
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RQ	Research Question
SLR	Systematic Literature Review
SQuAD	Stanford Question Answering Dataset
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
UI	User Interface
URL	Uniform Resource Locator
VIPS	Vision-based Page Segmentation
VSM	Vector Space Model
WHO	World Health Organization
WWW	World Wide Web

Chapter 1

Introduction

This chapter introduces the purpose of this dissertation by providing the context, goals and outcomes on the topic of providing reliable information about COVID-19 through conversation agents and thus avoiding the question-answering congestion on Health Services.

1.1 Context and Motivation

In late 2019, an outbreak of the novel coronavirus SARS-CoV-2, which started in Hubei Province of the People's Republic of China, spread to many other countries in the world. In January 2020, the WHO (World Health Organization) Emergency Committee declared a global health emergency, given the growth of cases internationally (Velavan and Meyer, 2020).

Yuki et al. (2020) state that the outbreak of SARS-CoV-2 originally started via a zoonotic transmission associated with the seafood market in Wuhan, China. Later it was recognized that human-to-human transmission played a major role in the subsequent outbreak. The disease caused by the aforementioned virus was named COVID-19 (Coronavirus Disease 19) and it affects mostly the respiratory system. Symptoms of the disease include fever, dry cough, dyspnea (short of breath), headaches, vomiting and diarrhea. However, the initial clinical sign of the SARS-CoV-2-related disease which allowed case detection was pneumonia.

The transmission of the virus from human to human occurs through coughing, sneezing, and the spread of respiratory droplets or aerosols, as well as through indirect contact (contaminated objects and airborne contagion) (Lotfi et al., 2020).

However, the general ignorance and initial disregard of the disease led to its wide-spreading, incapacitating health systems to treat all the diseased, growing this way the number of fatalities related to the COVID-19. In Figure 1.1 (extracted from *WHO Coronavirus (COVID-19) Dashboard*), the evolution of the numbers of cases and deaths worldwide is presented.

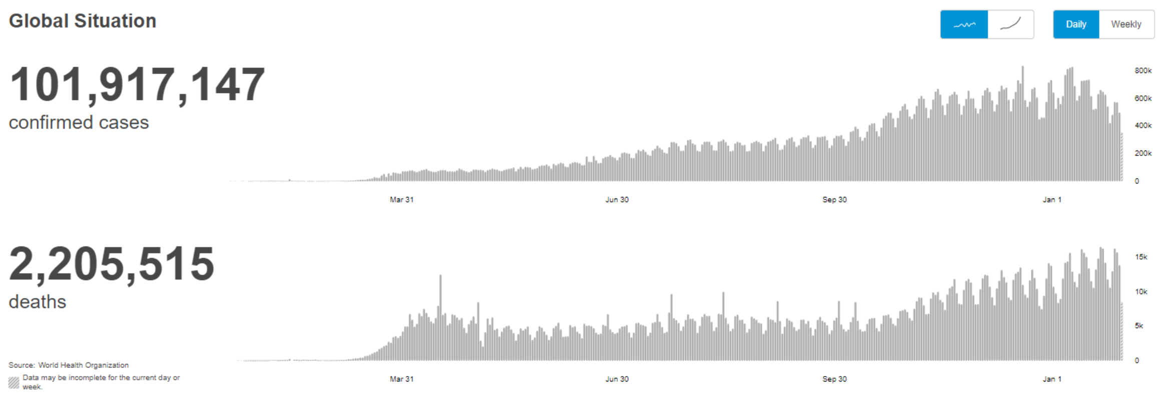


Figure 1.1: Evolution of COVID-19 cases and deaths worldwide (*WHO Coronavirus (COVID-19) Dashboard*)

With the proliferation of social networks, an increasing amount of fake news and fake information is being presented to people every day at every second. Between 2012 and 2017, 40% of the medical news shared on social media contained misinformation (Waszak et al., 2018). This phenomenon could be well confirmed during the COVID-19 pandemic (Naeem and Bhatti, 2020), where misleading information spread like wildfire. In order to clarify doubts and to verify the information read on the internet, people can contact the Health Services (e.g. SNS24, Portugal's Health Services' contact center), thus congesting them and leading to the inability to treat more urgent medical issues.

Chatbots can be used to relieve this congestion, allowing people to easily get an answer for their questions without the intervention of another human agent, usually a doctor or a nurse (Dale, 2016). However, chatbots need to be fed with reliable and trustworthy information from renowned sources, in order to mitigate the fake information dissemination. This way, the proposed work would be composed of three parts:

1. Reliable Information Gathering: textual information will be retrieved by a web crawler through reliable sources;
2. Information Processing: information collected by the crawler must be treated using text segmenters and classifiers;
3. Chatbot Development: implementation of an intelligent conversation agent that provides answers to the users' questions.

By grouping all the three parts, an application without any previous data can answer the given questions, parsing the text accordingly and providing the right reliable information that was gathered or redirecting to a more complete source.

Odhiambo and Okungu (2020) identified the need for trusted public health information sources as one of the top 15 COVID-19 research topics and areas, as well as the creation of datasets related to the disease. This work is thus an interesting contribution to both topics, providing an application that provides trustworthy information and creating a dataset for that purpose.

1.2 Objectives

The purpose of this research is to set the foundations for the implementation of a Chatbot fed by reliable information in order to answer people's questions about the COVID-19 disease.

There are several web crawling approaches and tools, and this work means presenting a review of them, while also mentioning some of the available tools to extract information from web pages.

The same applies to the information processing step: several methodologies for text segmentation, classification and summarization are available in the literature.

When it comes to chatbots, there are applications using different algorithms. Some make use of machine learning and deep learning, generating responses based on an intelligent algorithm, while others rely on predefined rules to fetch the answers for each question.

In order to review the existing modern approaches to each of the mentioned thematics, a Systematic Literature Review (SLR) will be conducted, allowing the identification of relevant works in the field and the reproducibility of the study. The SLR will be applied to each of the application's parts, stated previously.

Before this review, a background overview needs to be done, helping to set the research questions addressed by the SLR.

The application to be developed should fulfill the following basic requirements:

- The application should be available as a web or mobile application.
- The application should be a chatbot, providing answers that are grammatically correct and coherent.
- Given users' questions about COVID-19, the application should answer them correctly and ask the user for feedback.
- The knowledge base should be constructed from reliable and trustworthy sources of information.
- The application should work with both English and Portuguese languages.

1.3 Achievements

The application developed in this work was a chatbot made available to the users as a web application and with its knowledge base built upon information extracted from reliable sources like governmental or organizational web pages, e.g. the World Health Organization and the Portuguese Health Ministry websites.

The application works in both English and Portuguese. This is done by having two sets of reliable sources, from which the pages will be downloaded, thus building an English and a Portuguese knowledge bases. Also, a second BERT model was trained using a translated version of the dataset SQuAD 1.1. The correct version of the BERT model is used according to the defined web browser's language used to connect to the web interface.

However, there are some flaws and drawbacks when it comes to the quality of the answers provided. Frequently, they are incomplete or incorrect, misleading the users. This is due to many possible factors, but the ones we could identify directly were the fact that the model was extracting the answers from questions in a FAQ web page, instead of the answers themselves. On the other hand, they are grammatically correct and coherent since they are extracted directly from the original text.

In conclusion, most of the requirements were achieved, although the answers' quality was not the expected and required, as mentioned before. The only requirement that was not developed was the request for user feedback, as it was not deemed an essential feature.

1.4 Document Structure

The present work is structured as follows:

- After the present introductory chapter, comes the State of the Art, where an overview of the thematics (web crawling, information processing and chatbot application) is presented (Chapter 2), followed by the conducted SLR (Chapter 3);
- Then, the Architecture and Solution Design of the proposed application (Chapter 4), where the system overview of the application and the architectures for its components;
- This is followed by the Implementation steps (Chapter 5), where the development of the application and each of its modules are described, as well as the multiple issues faced and how we overcame them;
- With the application already implemented, the Tests and Tuning made to it in order to document the outcomes and improve the results are shown (Chapter 6);
- At last, the Conclusions of this work are presented (Chapter 7), making an overview of the whole work and documenting some improvements that could be developed as future work.

Chapter 2

Background

This chapter sets the foundations for the wanted application, by providing an overview of the different thematics contemplated in this work. Therefore, a list of reliable sources is presented, along with the first investigation on web crawlers, information processing and chatbot development. This chapter is a preliminary study to set the basis of the SLR.

2.1 COVID-19 Reliable Sources

Given the fact that we are currently facing an infodemic, as said by Cinelli et al. (2020), where misinformation about COVID-19 is damaging not only people's lives but also government actions to fight the pandemic.

As an example, when CNN anticipated the Lombardy (Italy) lockdown and its inhabitants crowded the public transports to escape the region. This happened before it was officially communicated and potentially increasing the spreading of the virus.

It is very important to use the information provided from accredited and certified sources, in order to make sure it is reliable and trustworthy, as well as being scientifically comproved.

To aid the identification of reliable sources and fighting the infodemic, Shahi and Nandini (2020) developed a dataset to help identifying fake news related to COVID-19, due to the disease being the biggest challenge fact-checkers have ever faced (Brennen et al., 2020).

On the other hand, the recommended information sources are the WHO and the Centers for Disease Control and Prevention (CDC), according to Carver and Phillips (2020), as well as governmental and public health organizations in general, according to Rochwerg et al. (2020).

Therefore, a list of reliable websites was built both for English and Portuguese languages.

English sources

1. WHO: www.who.int/emergencies/diseases/novel-coronavirus-2019
2. CDC: www.cdc.gov/coronavirus/2019-ncov
3. NHS (UK): www.nhs.uk/conditions/coronavirus-covid-19
4. ECDC: www.ecdc.europa.eu/en/coronavirus

Portuguese sources

1. Portuguese Health Ministry: covid19.min-saude.pt
2. Portuguese Government: covid19estamoson.gov.pt

2.2 Web Crawling

In the modern era of technology, the Internet has experienced explosive growth, with huge amounts of information being stored and sought each second. Without a proper methodology of indexing and searching, it would be extremely inefficient and complicated to obtain the information needed. The users would need to know the specific storing address of the documents they wanted or navigate from link to link until finding what they wanted. To solve this issue, search engines were developed to ease the process of fetching the required data with a few clicks.

2.2.1 Information Retrieval

The problem of obtaining information from a huge data source lies under the scope of Information Retrieval (IR), the activity of obtaining information from a number of informational resources. Saini and Arora (2016) state that IR can be divided in two approaches: the traditional and the automated, as seen in Figure 2.1.

The traditional IR systems are index-term based search models, which are divided into three primary categories, according to the survey conducted by Dong et al. (2008):

- theoretic: boolean, case-based reasoning (CBR), fuzzy set and extended boolean models
- algebraic: vector space (VSM), generalized vector space, latent semantic indexing (LSI) and neural network models (ANN)
- probabilistic: inference network, brief network and probabilistic models

To decrease the information overload caused by the tremendous explosion in the amount of unstructured data, both internal, corporate document collections, and the immense and growing number of document sources on the Internet, the need for effective methods of automated IR has grown in importance, as sustained by Greengrass (2000), since the traditional IR methodologies were usually applied within small, controlled and non-linked collections of documents. These automated approaches have been modeled after the traditional search systems and are used in:

- web crawling: downloads web pages for further processing by search engines for indexing the visited pages
- indexing: allows content extraction
- querying languages: including query syntaxes such as boolean queries, natural language queries, phrasal queries, proximity queries and patterns matching
- data mining: extracts useful and meaningful data, uncovering hidden relationships among it
- machine learning: helps solve complex and challenging problems through statistical natural language processing and information retrieval

Search engines can be described as important tools that allow users to take advantage of automated information retrieval.

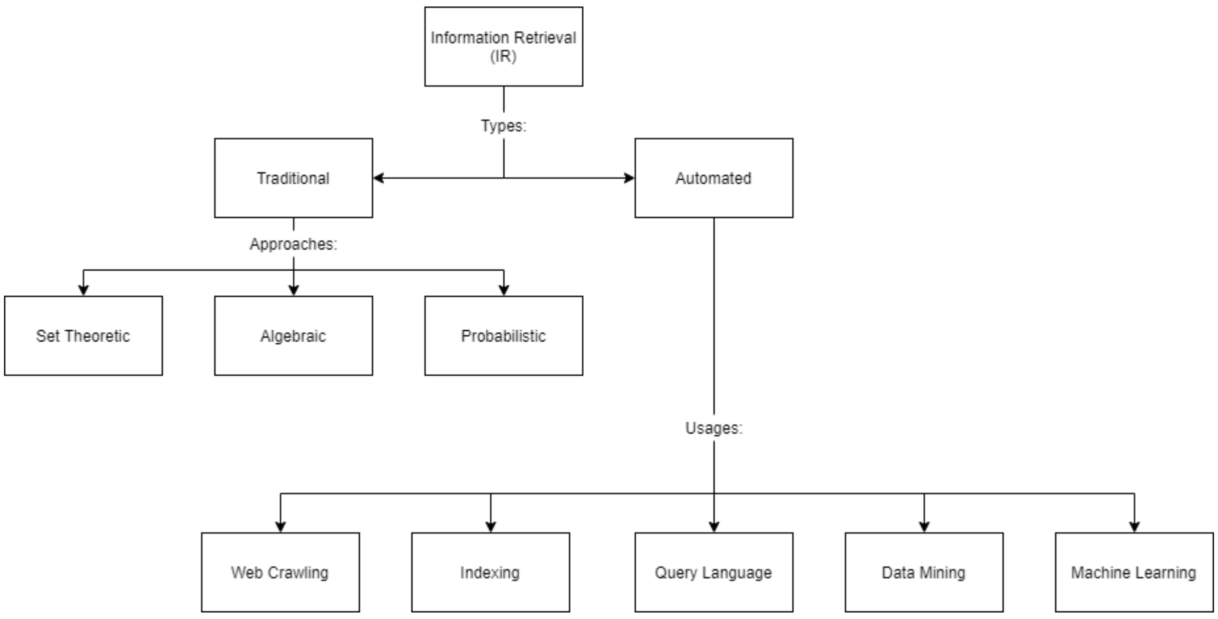


Figure 2.1: Categorization of information retrieval approaches (adapted from Saini and Arora (2016))

2.2.2 Search Engines

As stated previously, a boom of information has flooded the World Wide Web (WWW), and without the search engines the process of finding information is similar to seeking a needle in a haystack (Kumar et al., 2017), so they became an essential and indispensable part of our digital lives. A search engine has three primary components:

- Web crawler: an application that keeps traversing the web and gathering the numerous sources and information
- Indexer: which, as the name indicates, indexes the data gathered by the crawler and allows the data to be correctly delivered for a given query

- Searching-ranking algorithm: that makes it possible to deliver the best results for the users' queries, ordered by relevance.

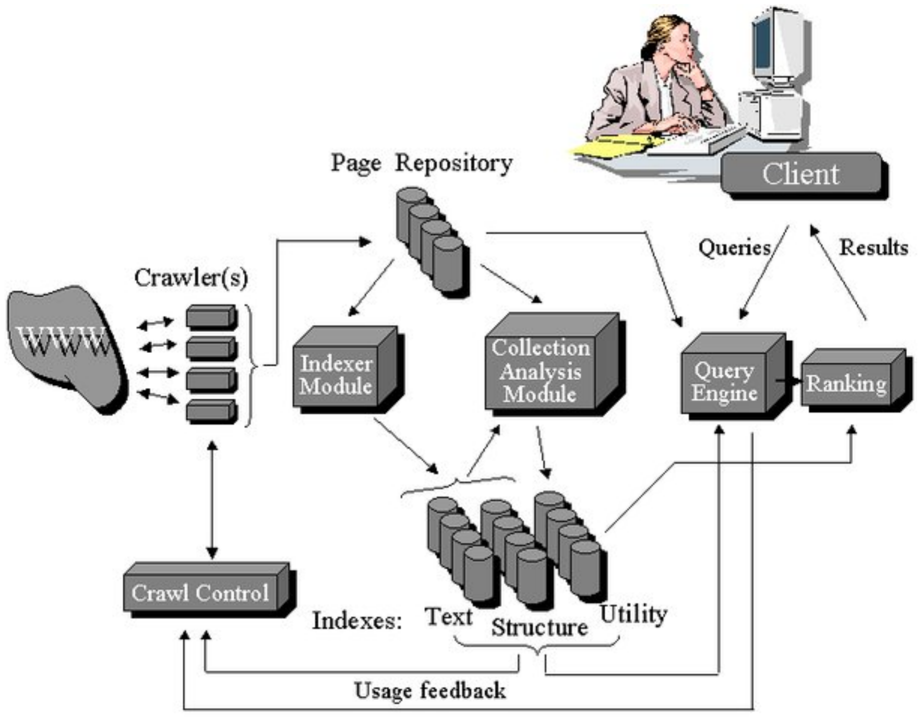


Figure 2.2: General search engine architecture (Arasu et al. (2001))

However, Arasu et al. (2001) presents a more complex and complete general architecture, seen in Figure 2.2.

Every engine has its foundation on a crawler, that browses the web from a given start set of URLs and fetches the URLs inside the retrieved pages.

Then, the URLs are passed to the crawl control module, which decides what links to visit next and passing them again to the crawler.

The crawlers also provide the downloaded web pages to the page repository. The indexer and collection analysis modules then generate a lookup table from the words existent in each page of the page repository.

The query engine is responsible for processing the users' search requests, relying heavily on the indexes generated by the previously mentioned modules to provide the right results, which are ordered by relevance through the ranking module.

Since the focus of this work is the development of an application that, given a set of URLs representing a reliable source database, can fetch the maximum of related and relevant information about COVID-19, we will focus on the crawler and crawl control modules.

We will merge both modules and refer to them as Web Crawler from now on.

2.2.3 Web Crawler

Gupta and Anand (2015) provided a clear and concise definition of web crawling.

It is the automated traversal of the web to collect all the useful informative pages, effectively and efficiently in order to gather information about link structure interconnecting those informative pages.

Usually, search engines require web crawlers (also known as spiders) to discover pages continuously, in an attempt to try to answer all the users' queries.

This approach is called by Kumar et al. (2017) as universal or broad crawlers, which keep on following links endlessly and getting all the pages encountered, without limiting them to a certain topic or domain.

Besides broad crawlers, Kumar et al. (2017) state that there was no standard taxonomy for web crawlers in the literature at that moment, but that we could nevertheless divide them in several types and purposes: preferential, hidden web, mobile and continuous/incremental crawlers.

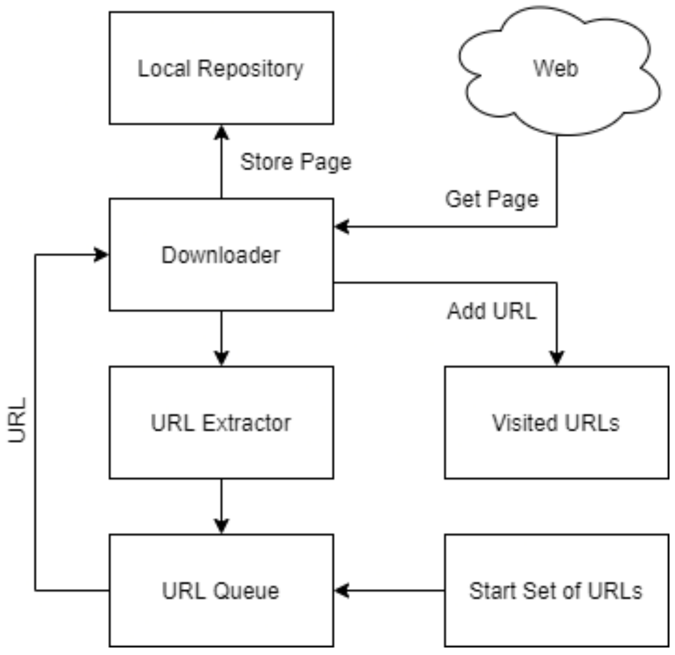


Figure 2.3: Basic Web Crawler architecture and behaviour

All the same, they share a basic common architecture, with differences between them, but with the same foundations. A web crawler is composed of:

- Start set of URLs - this is the seed of the crawling process, which means the crawler will start downloading them and extracting the URLs from these set and adding them to the URL list
- Downloader - which downloads the pages and adds them to the local repository as well as marking the URL as visited
- URL extractor - a component that given a web page extracts the URLs present in it and adds them to the queue
- URL queue - contains the URLs of the pages yet to be crawled
- Local repository - stores the downloaded pages, to be later processed by another application (or module) outside the crawler.

By integrating all the previous parts, we obtain a functional web crawler that only stops when all the URLs in the queue have been visited. Figure 2.3 exemplifies the mentioned architecture and flow.

In the next subsection, the several types of web crawlers that were already mentioned will be further explained.

2.2.4 Types of Web Crawlers

The taxonomy of web crawlers proposed by Kumar et al. (2017) is graphically presented in Figure 2.4.

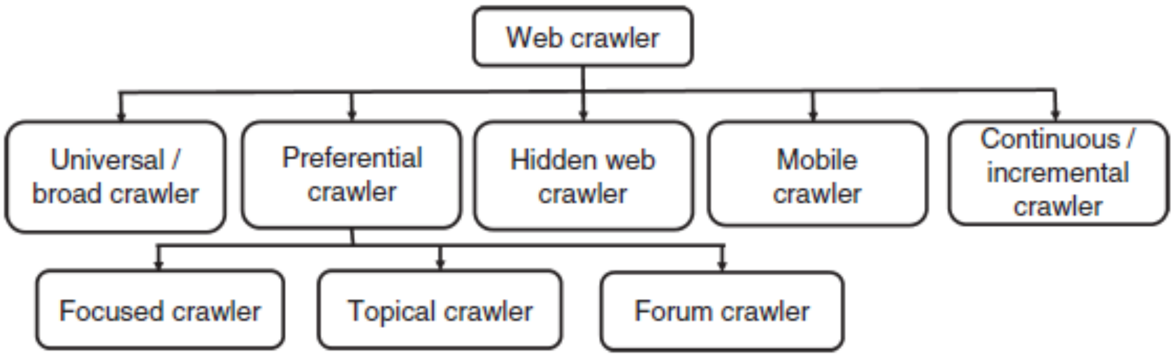


Figure 2.4: Taxonomy of Web Crawlers (Kumar et al. (2017))

The mentioned types are described below.

2.2.4.1 Universal / Broad Crawler

The broad crawlers have already been pointed in this work as the standard type of crawlers for the usual search engines. They scatter the web in order to index as many pages as they can, not discarding any page due to any reason.

2.2.4.2 Preferential Crawler

This type of crawler, as opposed to universal crawlers, does not crawl every URL they come across. Instead, that only happens when the web page is related to a given topic, subject or condition submitted by the user. Kumar et al. (2017) also identify three subtypes of the preferential crawler:

- Focused crawler: works similarly to the topical version, but assumes also the existence of a dataset of web pages labeled as relevant and non-relevant;
- Topical crawler: used to fetch information on a specified topic;
- Forum crawler: used to crawl only online forum content.

To make this possible, there are several approaches to execute the filtering. The survey conducted by Kumar et al. (2017) identified the following, ordered by frequency:

1. **Using soft computing techniques**: machine learning algorithms are used to classify a web page and determine if it belongs to the wanted topic. For example, neural networks are used to learn the page's characteristics and decide its relevance, but many other algorithms and techniques are employed, such as clustering (e.g. formal concept analysis), genetic algorithms, ontology-based or Hidden Markov Models (HMM).

2. **Application-based**: crawlers developed for a specific task or application, e.g. crawling medical data for sentiment analysis or searching for malicious YouTube videos.

3. **Based on link, text and URL**: these crawlers use the text of the web page and in the URL for deciding its relevance. This methodology allows the URLs to be discarded without downloading the page, by using only its URL. For example, a study is mentioned to use link distance instead of probability or relevance score. Also, some of these methodologies allow the classification to be made without the whole text of the web page, by partitioning it in blocks, or even using the text around a link to speculate if it is relevant or not.

4. **Based on context graphs, decision trees and DOM**: here the Web is visualized as a graph or decision tree where every page represents a node. This methodology links nodes with similar interest to each other, thus allowing the identification of relevant pages through the taxonomy of topics.

5. **Semantic crawling**: crawlers in this category exploit the semantics of the Web content and use some ontology heuristics, by calculating a web page relevance using domain knowledge related to the wanted topic.

6. **Learnable crawling**: with this approach, the crawlers need to maintain a continuously updated knowledge base, using either supervised or unsupervised algorithms for the classification.

7. **Topic specific**: this kind of crawler is used to build a repository of web pages on a given topic, using a topic similarity to determine the category that best fits a page.

8. **Parallel and distributed**: given the large size of available web documents, using a parallel and distributed crawler allows a faster and lighter processing, by dividing the web in segments and assigning each one to one of the parallel agents. By distributing the computing resources, each agent will have less overload and classify the pages more efficiently, which means more pages crawled in less time.

9. **Language classification-based**: these crawlers are used to construct a repository of web pages on a specific language or topic corpora.

10. **TF-IDF- and rule-based**: TF-IDF is a metric that indicates a word's importance in a document for a collection of documents and rule-based crawlers use linkage statistics among topics, by being trained with a predefined topic taxonomy to generate the rules with a probability score.

11. **Vertical search engines**: these special search engines differ from the normal ones in the fact that the results obtained are selected from a smaller set of websites.

12. **Based on query, metadata and keyword**: as it indicates, crawlers that follow this methodology use metadata present in the web page to classify its relevance, using queries' keywords provided by the users for training and relevance feedback.

13. **Location- and geographical-based**: the goal of these crawlers is to gather the web pages relevant to the location (e.g. city-state pairs) provided by the user. To achieve this goal, crawlers use the geospatial information hidden in the page's data.

14. **Incremental and revisit policy-based**: this methodology tries to tackle the crawling difficulty created by the Web's dynamism and continuously changing state, proposing a revisit policy that programs the crawler to recrawl a web page after a specified period of time if it detects any change on its text or structure and act accordingly, processing it again.

On the other hand, Gupta and Anand (2015) make a simpler division and identification of preferential crawling approaches. In their study, they identified priority-based, structure-based, context-based, learning-based and other crawlers (this last category includes all the approaches that do not fit in none of the others).

Priority-based crawlers store the extracted links from a web page in a priority queue, according to their relevance.

Structure-based crawlers are divided in three classes: the ones using division score and link score and the ones using a combination of content and link similarity. Link score is calculated on the basis of division score and average relevance score of parent pages of a particular link. Division score means how many topic keywords belong to the division in which the particular link belongs.

Context-based crawlers are comparable to the previous approach, with the difference that here the user context and environment are also taken into account.

At last, **learning-based crawlers** use datasets (composed of four relevance related attributes: URL words, anchor text, parent page and surrounding text) to train classification machine learning algorithms (like Naive Bayes), thus predicting the relevance of the unvisited URLs.

Ganguly and Raich (2014) divides the focused crawlers in different categories, mentioning **ontology-based**, **reinforcement learning**, **intelligent** and **concept graph** focused crawling.

Despite the existing differences regarding the classification and division of approaches, a basic common architecture for focused crawling is proposed by Joe Dhanith and Surendiran (2019), as seen in Figure 2.5.

2.2.4.3 Hidden Web Crawler

The hidden web is the part of the web that is not reachable by following the links in a website, but only through some kind of form (e.g. login, query or search interfaces). Thus, a traditional web crawler cannot

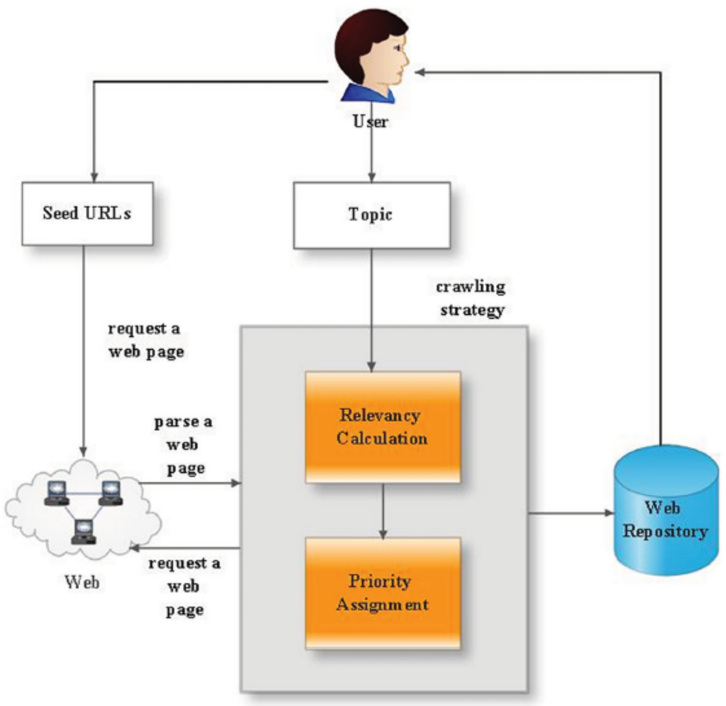


Figure 2.5: Basic Architecture for a Focused Web Crawler (Joe Dhanith and Surendiran (2019))

usually access it, so special efforts are needed to do so and the heterogeneous and dynamic nature of the web makes it a tough challenge. This is where hidden web crawlers come into play. According to Kumar et al. (2017), the used approaches (ordered by frequency) are: keyword query-based, form-based, revisit policy and incremental, attribute and label extraction, labeled value set-based, and domain- or topic-specific.

2.2.4.4 Mobile Crawler

Mobile crawlers tackle the limitations of traditional web crawlers, eliminating the centralized data access, web page filtering and indexing. These types of crawlers can move themselves to the web server to download information and contents available on the server, therefore bringing ”code to data”instead of the usual ”data to code”. Like with other types, studies around mobile crawlers can be divided into categories such as freshness and revisit policy-based, user feedback-based, agent-based and ontology-based.

2.2.4.5 Continuous / Incremental Crawler

At last, Kumar et al. (2017) specified the continuous/incremental crawler type. Given that the Web is dynamic and the web pages' data is frequently changing, this type of crawler is used to maintain the search engines' index database. The challenge with incremental crawlers is the trade-off between resource consumption and web page freshness, since keeping the information about all the billions of existing pages updated requires a tremendous computational effort and availability.

2.2.5 Selection of the Web Crawler's Type

Since the present work's web crawler objective is to obtain web pages related to COVID-19 from a selected set of reliable sources, the most adequate type of web crawler is the preferential one, excluding the forum crawler. Generally, topical crawlers are included in the focused crawler category.

The start set of our crawler is restricted to a few official sources, given the newness of the topic. The COVID-19 pandemic started in 2020, but the disease was discovered in 2019.

In addition, we intend to use a small number of reliable sources. So, we don't need the use of a parallel crawler.

As we intend to process the web pages' text content in a different module, the mobile crawler was also discarded. Also, a hidden web crawler is not necessary since the information to be fetched is usually obtained through links displayed in the web pages in order to allow people to easily access it and to fight misinformation and fake news, an issue that prejudiced the health and lives of those who followed and believed in the information wrongly spread on the internet, e.g. social networks.

The focused web crawler leads to a more efficient web page processing, once the reduced number of pages gathered (and the trustable guarantee that it belongs to the specified topic) simplifies the information processing process and thus provides better and more related answers to the users' queries in this work's chatbot.

With the wide diversity of methodologies and approaches regarding focused web crawlers, a systematic literature review was conducted (see Chapter 3) to investigate the most appropriate approach for this work's purpose, as well as surveying which tools are available to develop and deploy it.

An important point to take into account is that the trustworthy information sources about COVID-19 are mostly governmental and taking into account that around 40% of the Internet traffic and consumption of bandwidth is attributed to web crawlers (Badawi et al., 2013), an effort needs to be made as a way of not overloading these web pages (and shutting down their servers, risking peoples' access to them). Koster (1994) defined a set of policies that must be followed, which are made explicit next.

2.2.6 Web Crawling Policies

As stated, there is a set of policies that must be followed in order to keep the servers working and not crashing them or creating a bottleneck in their services (Koster, 1994).

- **Politeness policy**: a crawler should not hamper any website with the requests, as servers may be overloaded as they have to handle the requests of the viewers of the site as well as the crawler's requests. To lighten the load there are solutions such as setting an interval between requests, as well as respecting the robots exclusion policy. This is defined by the websites' administrators in the *robots.txt* file, which sets the pages not reachable by the crawler.

- **Parallelization policy**: this policy is aimed at controlling the access to the web pages made by multiple web crawlers, with the goal of maximizing its download rate. Thus, the URLs discovered in the crawling process should follow the policy which states they should be assigned to different threads running in parallel.

- **Revisit policy**: used to minimize the cost associated with updating outdated web pages by the crawler, which needs to revisit them in order to keep the indexes of a search engine up-to-date. This revisiting should be uniform or proportional, following the appropriate policy for each.

- **Robustness policy**: a crawler must be immune to any kind of malicious behaviours perpetrated by web servers.

2.2.7 Web Crawling Challenges

In addition to the need to follow the policies aforementioned, there are other challenges faced when developing a web crawler.

As said before, the Web is very dynamic and is constantly changing, while also having inconsistent data structures, as websites are built with no sense of universal standard or norm. Despite that, there are many web pages poorly formatted, and even with errors. Thus, an extra effort is needed to parse the information contained in the downloaded web pages.

In the next section, an overview of the needed information processing is made, including mainly web page parsing and relevant data extraction, laying, therefore, the foundations for this work's application module responsible for such processing.

2.3 Information Processing

Having the relevant web pages downloaded, the need for extracting the relevant data arises. Therefore, a module that processes the web pages, discarding the unwanted metadata associated with the documents' structure, segmenting the text in phrases or blocks of text with similar meanings and then classifying them, before feeding the information to the chatbot module. The next subsections provide an overview on the topic, divided by phases.

2.3.1 HTML Parsing

The process of extracting data from web documents is also called web scraping. Parvez et al. (2018) states that given the dynamic nature of the web, it would be very difficult to scrape the web pages, so automation is used whenever possible, such as with web crawling. Parvez et al. (2018) divided a web scraper application in two components: the already describer web crawler and a data extractor (Figure 2.6).

Given the unstructured nature of a web document, the data extractor aims at extracting meaningful data from these unstructured documents. There are several tools and techniques available for this step of the processing, which Parvez et al. (2018) have listed summarily:

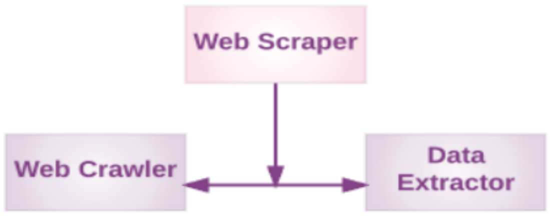


Figure 2.6: Web Scraper Main Components (Parvez et al. (2018))

- Human copy and paste: most common and traditional approach to extract information from the web sources but requires tremendous effort for large datasets.
- HTML parser: allows fast and real-time parsing of web pages, is widely used because of its simple design, processing speed and ability to handle real-world HTML.
- HTTP programming: with socket programming, HTTP requests are sent to the remote servers, assuring this way that the exact data from web pages is retrieved.
- Tree-based techniques: web pages are generally in a semi-structured form leading to a labeled ordered rooted tree usually called Document Object Model (DOM), depicting web pages as a hierarchical structure (exemplified in Figure 2.7). This DOM tree is used in some data extraction techniques, addressing the elements in the tree using XPath (a query language to select specific nodes) or with tree edit distance matching algorithms.
- Web wrapper: procedure of extracting structured data from unstructured (or semi-structured) data sources. Can follow approaches based on regular expressions, logic or machine learning algorithms.

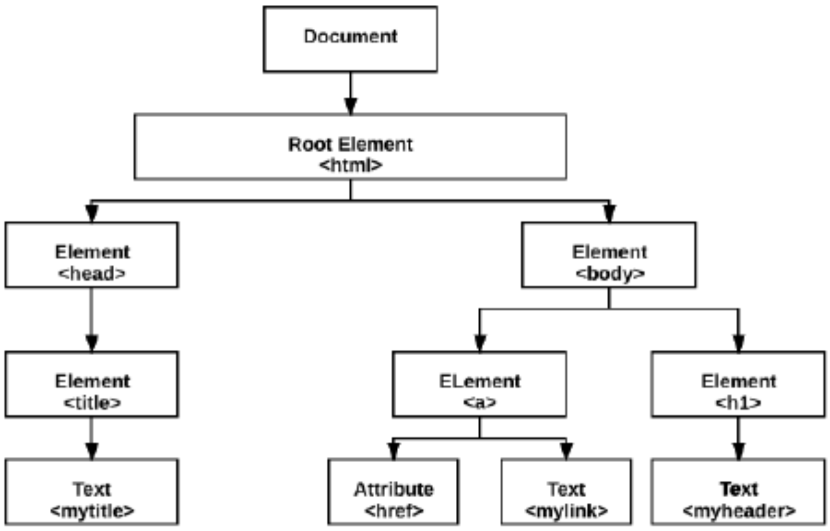


Figure 2.7: DOM tree example (Parvez et al. (2018))

We will focus on the HTML parser, since it plainly and simply provides the functionalities to satisfy the requirement of cleaning the HTML documents. Parvez et al. (2018) described two HTML parsing tools, one for Java called JSoup[1] and another for Python called BeautifulSoup[2], the latter being one of the most widely used tools for data extraction. It provides services like cleaning and parsing the extracted document due to its ease of use and because of many of the NLP applications being written in Python (it has lots of reputable third-package libraries for that purpose (e.g. NLTK, spaCy) and allows the implementation and deployment of many machine learning algorithms).

Also, BeautifulSoup is thoroughly documented by Richardson (2017), which led as well to the decision of choosing BeautifulSoup for the first component of this work's data extractor.

After removing the unwanted information from the web documents, we need to break it into text chunks, grouped by their meaning and central idea. Thus, text segmentation followed by text classification is needed and will be discussed next.

2.3.2 Text Segmentation

Text segmentation is the method of splitting a text into smaller parts, where each segment has a relevant meaning (Pak and Teh, 2018). It can be useful for emotion extraction, sentiment mining, opinion mining, topic identification, language detection and information retrieval.

Pak and Teh (2018) stated in their review that there are different types of segments, including topic, sentence and word, and that these types were selected based on their analyzing targets and specifications. They also discovered that the most frequent type is word segmentation, as it can include the character segmentation problem associated with the Asian languages, where, unlike English or Portuguese, words are not explicitly delimited by whitespaces (Huang et al., 2003).

Given the objective of grouping the web page texts according to their meaning, the most appropriate method is topic segmentation or sentence segmentation followed by its classification. This work will focus on English and Portuguese. Generally sentence segmentation for those languages would be done using rather simple regular expressions, dividing the text by the dot character (”.”), interrogation (”?”) and exclamation (”!”) marks. This may require some tweaks for extraordinary situations like dialogs. The topic segmentation would be much more complex, as described next.

In their review, Pak and Teh (2018) identified that the second most frequent methodology for text segmentation was topic-based. This type of segmentation can be extremely useful when it comes to mitigating information overload when a whole document is presented, by presenting only the relevant parts.

The review identified different approaches for topic segmentation: orthographic division with similarity scores (Osman and Yearwood, 2007), probabilistic latent semantic analysis (PLSA) (Brants et al., 2002), unsupervised algorithms like TextTiling, DotPlotting and C99 (Flejter et al., 2007) or using Latent Dirichlet Allocation (LDA) and Multinomial Mixture (MM) (Misra et al., 2011).

[1]https://jsoup.org/

[2]https://beautiful-soup-4.readthedocs.io/en/latest

2.3.3 Text Classification

Text classification is the process of assigning text into two or more categories (Miner et al., 2012). The goal of text classification is to extract no more than the category from the text, being usually applied to documents. The basic approach followed in text classification is deriving a set of features and applying an algorithm to process and use these features to select the most likely category. An example of a text classification application is spam filtering.

Text classification algorithms usually employ a statistical model to assign labels, but also existing rule-based approaches.

For the feature creation step, according to Miner et al. (2012), most text classification problems require a text preprocessing, where the stopwords are stripped, the text is tokenized and stemmed, in order to use words (often called terms or tokens) as the primary feature, putting them in a standardized form before the classification. Tokenization is the process of dividing up raw text into discrete words, while stemming is the process of normalizing word forms. Another challenge is the choice of which text to use: documents may contain unstructured text, titles, abstracts or even metadata. There are other features like the document structure or even its length. Also, most of the text classification algorithms assume that the text has been converted to a vector of features.

Regarding the classification algorithms, Miner et al. (2012) presents two of the most popular, both being efficient for high-dimensional data and being among the most accurate for text classification: Naive Bayes and Maximum Entropy. These classifiers need to be trained in order to set the numerical parameters of the model (feature weights and thresholds). Once the training is finished, they can be used to classify new and previously unseen text. However, if the training set is not correctly constructed, the classification will not be precise and correct.

Mironczuk and Protasiewicz (2018) present in their review a flowchart of text classification, seen in Figure 2.8. The classification process starts with the data acquisition from various text sources and, from there, a dataset representing a physical or business process is obtained. The next step is to preprocess the dataset to generate a representation required by the selected learning method. Features are constructed and weighted with the feature representation algorithm, being the number of features reduced at last by the feature selection algorithm. Then, the classification model is developed and trained, allowing new inputs to be classified according to their features.

In the review, Mironczuk and Protasiewicz (2018) described different methodologies for each of the text classification phases, summarizing extensively the existing literature (including the most recent studies) on the matter.

2.4 Chatbot Application

The chatbot is the application that can give the correct answers based on correct information.

Question-answering (QA) systems enable the process of retrieving precise answers to natural language

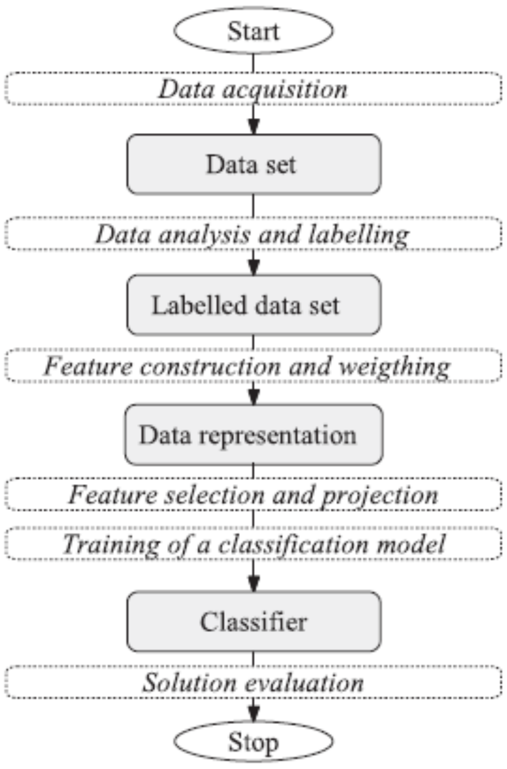

Start
Data acquisition
Data set
Data analysis and labelling
Labelled data set
Feature construction and weigthing
Data representation
Feature selection and projection
Training of a classification model
Classifier
Solution evaluation
Stop

Figure 2.8: Text Classification Flowchart (Mironczuk and Protasiewicz (2018))

questions (Andrenucci and Sneiders, 2005). These systems were introduced between late 1960s and early 1970s and the rise of WWW allowed the QA systems growth and evolution, with the need for user-friendly search engines.

Along with QA systems, chatbots are quickly gaining a relevant position in the contemporary paradigm of intelligent systems. They allow an easy interaction between users and information systems. Among the most popular chatbots and conversational agents are Amazon's Alexa, Microsoft's Cortana and Google's Google Assistant (Nithuna and Laseena, 2020). These applications can not only answer users' enquiries by searching the web and returning the most popular results but also make a call to the wanted person or set up an alarm. Therefore, we can divide chatbots into different types, as specified next.

2.4.1 Chatbot Classification

Barbosa et al. (2020) reviewed the chatbot taxonomy, and based on Nimavat and Champaneria (2017) identified the following criteria of chatbot classification (presented in Figure 2.9):

- **Knowledge Domain**: being either closed-domain, meaning they are trained and focused on a particular area of expertise (e.g. a restaurant booking bot), or open-domain agents, which can replicate a conversation talking about general topics (acting as a chit-chat bot).

- **Service Provided**: differ on emotional proximity to the user: interpersonal chatbots give information to the user (e.g. booking assistants or FAQ bots); intrapersonal agents exist in a personal area, acting as companions with a unique personality to the user (e.g. calendar managers or behavioural therapist bot); and inter-agents can be defined by Internet of things (IoT) communication, where two or more systems have their services linked and integrated.

- **Goals**: information based (retrieving information stored from querying a database), conversation based (replicating a human being with continuous conversation) and task based (where their actions are under a specific task following predefined events).
- **Response Generation Method**: intelligent systems handle the method of processing inputs and designing responses using Natural Language Understanding (NLU) to comprehend the users' queries (sustained by a self-learning algorithm), while rule-based systems use parsing and pattern matching methods to create rigid answers (with limited and fixed outcomes) and hybrid systems combine the two previous approaches, using rules to manage the conversation flow and machine learning to provide the responses.

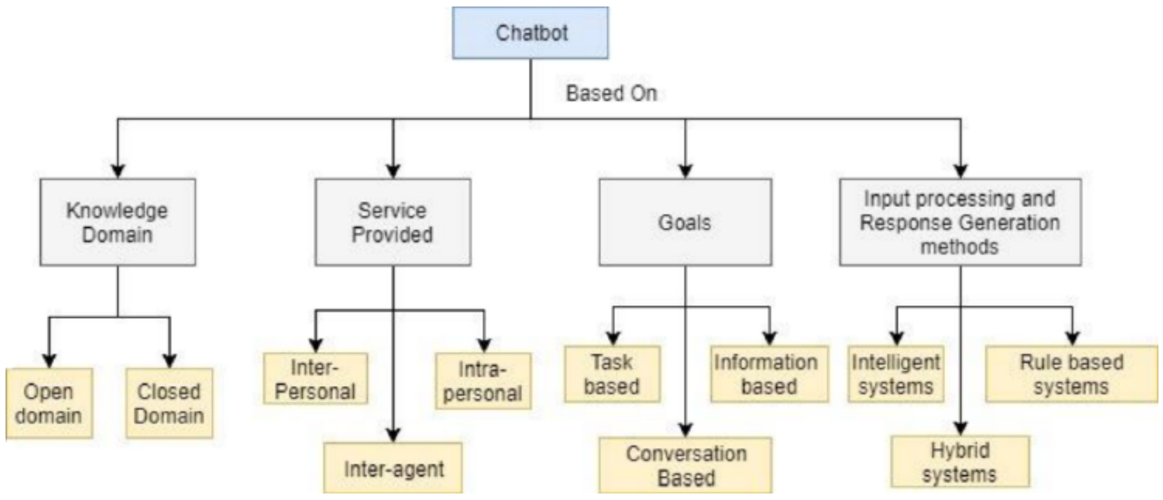


Figure 2.9: Chatbot Classification Criteria (Nimavat and Champaneria (2017))

But, Barbosa et al. (2020) concluded based on recent studies that following an overall chatbot classification can be subjective and deprecated to the scope of a specific use, since the chatbot field is very dynamic, mainly due to the emerge of new technologies, and so proposed a new chatbot classification, as seen in Figure 2.10, where they are classified according to their knowledge domain (broad or restrict) and to their conversation design (rule-based or artificial intelligence).

2.4.2 Approaches to Chatbot Development

Following the classification presented by Barbosa et al. (2020), for either broad or restrict, the chatbot design can be based on rules or artificial intelligence. For the second methodology, chatbots can be retrieval-based or generative-based.

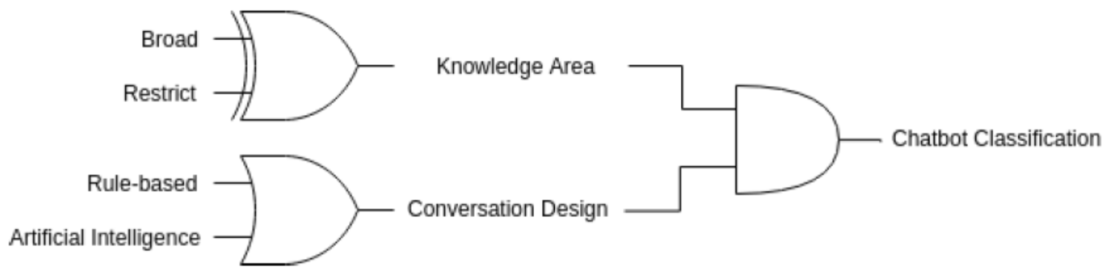


Figure 2.10: Chatbot Classification (Barbosa et al. (2020))

2.4.2.1 Rule-Based Approach

Chatbots answer questions based on specific rules in which they are trained. However, they cannot provide a correct answer if the input does not match the predefined patterns. Therefore, these bots are straightforward to implement, but it is almost impossible and very time consuming to develop rules for every scenario.

Rule-based chatbots are a good approach for simple input applications, but fail on processing complex queries, compared to the artificial intelligence approaches.

2.4.2.2 Retrieval-Based Approach

Chatbots are trained on a set of questions and their likely outcomes, identifying the most accurate answers from a repository of all possible answers for each question. This identification can be based on a rule basis or machine learning classifiers. However, these bots can not generate new responses nor process syntactically incorrect phrases, as they are solely based on predefined answers.

Retrieval-based chatbots can be used when the data is limited and the knowledge area is restricted to a few conversation scenarios.

2.4.2.3 Generative-Based Approach

Chatbots can generate new answers, as they are not dependent on a predefined response repository, making them intelligent systems. The responses are produced by interpreting the question word by word. On the other hand, they are prone to errors and wrong answers, as they take into account spelling and grammar.

Generative-based chatbots outperform rule-based models as they can adapt to complex and unforeseen scenarios, once they are trained using large amounts of processed data.

2.4.3 Selection of Chatbot Development Approach

Regarding the knowledge area, this work's application will have a strict knowledge base, focusing only on the COVID-19 pandemic.

Regarding the conversation design, a retrieval-based approach is not feasible, as a dataset of questions and answers does not exist. The data used to provide responses to the users will rely solely on the information crawled and processed by the application from the selected reliable sources.

Also, a rule-based approach is not adequate as well, given the fact that COVID-19 is always being updated with new scientific facts and knowledge, outdating the previously released information about the disease. Therefore, a new set of rules would be needed to be able to answer the users' queries every time an update was made. This goes against the objective of the present work of having the latest available reliable information.

So, the best approach would be following a generative-based approach, which should allow the chatbot to answer most of the questions inputted, whether they are simpler or complex, hence achieving the goal of bringing COVID-19 correct information to every user.

In order to understand what is the best algorithm and tool for the wanted application, this thematic will also be focused on the SLR already mentioned.

Chapter 3

Systematic Literature Review

This chapter describes the methodology followed to perform the literature review, describing each of the steps, from the Research Questions' definition to the Data Extraction phase.

In the end, a synthesis of the selected studies is presented.

3.1 Methodology

Using a systematic approach to review the existing literature represents an efficient way to identify relevant works to a defined set of research questions. This approach also allows studies to be easily peer-reviewed and replicated (Keele et al., 2007), since a SLR should follow a strict methodology defined previously, describing the exact way the relevant studies were selected.

Instead of following the SLR method described by Keele et al. (2007), this work first tried to follow the one described by Wohlin (2014), where the guidelines for snowballing a SLR are defined. Rather than using databases and research queries to gather the studies needed to answer the research questions, this method has its basis on an initial set of studies. Then, through Backward Snowballing (the works cited by a given paper) and Forward Snowballing (the works that cite a given paper) more studies are referenced.

However, starting with a start set of 27 papers, after the first iteration of backward snowballing more than 700 papers were obtained and given the lack of automated tools to aid the process of analyzing the papers for inclusion or exclusion, the method was rejected. In fact, given the wideness of this work's scope (having three different thematics), analyzing the huge quantity of papers gotten through the snowballing procedure was consuming too much time and efforts while providing comparable results with the classic SLR approach (Badampudi et al., 2015; Jalali and Wohlin, 2012).

Having decided to use the classic SLR approach (Keele et al., 2007), the steps to carry it out can be graphically visualized in Figure 3.1.

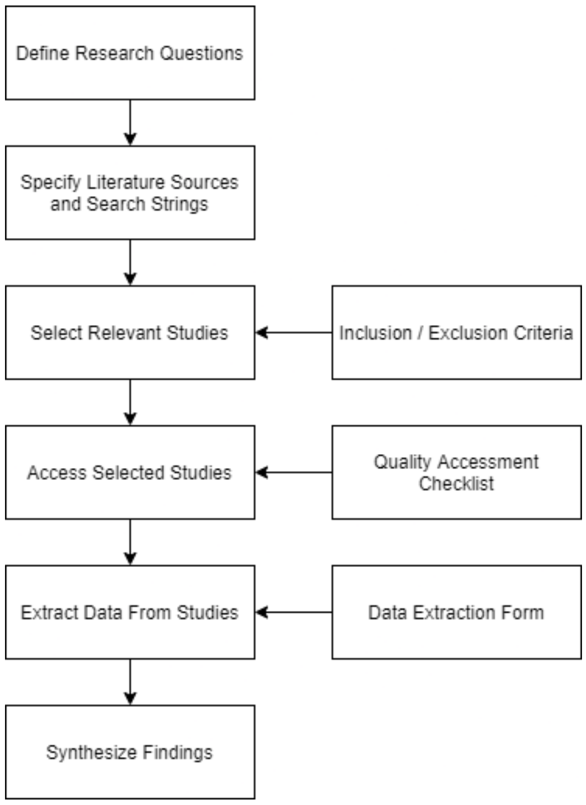

Define Research Questions
Specify Literature Sources and Search Strings
Select Relevant Studies
Inclusion / Exclusion Criteria
Access Selected Studies
Quality Accessment Checklist
Extract Data From Studies
Data Extraction Form
Synthesize Findings

Figure 3.1: Steps to conduct a SLR (adapted from Amara et al. (2016))

3.2 Research Questions

As stated previously, the SLR was executed on web crawling and chatbot development. So, the research questions will be divided into two groups. However, if the chatbot development findings revealed the need for an advanced text processing module, the review would be extended to also cover that thematic.

3.2.1 Web Crawling

RQ1: Which technique should be used to crawl the defined sources?

Motivation: With the exponential increase of web sites and data sources and the continuous research of new search engine algorithms to improve the process of finding information, many strategies for the collection of data have been developed. With such a diversity, there are various types of web crawling approaches, each one focusing a different problem to be solved.

Despite the inexistence of an official taxonomy for web crawlers, they can be divided in various types (Kumar et al., 2017). Since we want to gather information from a previously defined set of sources (websites) we want to use a focused web crawler, that starts crawling from the specified seeds. There are many techniques to perform the focused crawling and therefore this RQ is aimed at defining which one is the best for our work.

RQ2: Which open-source or free tools are available for focused web crawling?

Motivation: This study aims to do a small survey on open-source tools to allow the implementation of focused crawlers and its deployment. In this work, a continuous information retrieval is essential to provide the chatbot with fresh data. For research purposes, an open-source tool is preferred due to community support, the continuous improvement and the open code to analyze the implementation (Bonaccorsi and Rossi, 2003; Scacchi, 2007).

3.2.2 Chatbot Development

RQ3: Which generative deep learning algorithms can be used for the chatbot application?

Motivation: One of the earliest NLP applications was a chatbot named Eliza, developed in 1960, so chatbots are not a new and unknown topic Dale (2016). With decades of research and development, there are many algorithms and tools that allow software engineers to build and deploy such applications.

For this work, a generative-based approach is wanted to provide the more correct and precise answers to each asked question. These generative approaches are usually implemented using deep learning algorithms. This research question aims at discovering which of the mentioned algorithms are adequate for this work. Unsupervised approaches are preferable due to the unnecessary existence of a pre-built dataset. It is also expectable to find tools that can aid the implementation of the algorithm.

3.3 Literature Sources and Search Strings

The literature sources used in this SLR are some of the major online databases available: ACM, IEEE Xplore, ScienceDirect, Scopus and Web of Science.

Having the list of sources defined, a search string will be constructed regarding each part of this study, following the approach by Keele et al. (2007):

1. Identify major search terms;
2. Check keywords of already analyzed studies to find more search terms;
3. Identify alternative spellings or synonyms of the chosen major terms;
4. Construct search string by joining the terms: boolean ORs are used to join alternative spellings/synonyms and ANDs to join major search terms.

For the reliable information gathering part, the major terms were identified, easily identified in Table 3.1.

From the identified terms, the following search strings were built:

Table 3.1: Search Terms and Alternative Spellings and Synonyms

Thematic	Terms	Alternative terms and synonyms
Web Crawling	Focused Web Crawler	Focused Crawler, Focused Crawling, Focused Web Crawling
	Techniques	Algorithms, Tools
Chatbot Development	Chatbot	Chatterbot
	Deep Learning	Machine Learning
	Generative	Generative-based
	Algorithm	Tool, Model

- (”Focused Web Crawler” **OR** ”Focused Crawler” **OR** ”Focused Crawling” **OR** ”Focused Web Crawling”) **AND** (”Techniques” **OR** ”Algorithms” **OR** ”Tools”)
- (”Chatbot” **OR** ”Chatterbot”) **AND** (”Deep Learning” **OR** ”Machine Learning”) **AND** (”Generative” **OR** ”Generative-Based”) **AND** (”Algorithm” **OR** ”Tool” **OR** ”Model”)

3.4 Inclusion and Exclusion Criteria

In order to select the appropriate studies for inclusion in the review, both inclusion and exclusion criteria were specified.

Following the approach of the authors in Amara et al. (2016), the inclusion criteria were obtained:

- If a study has a journal and a conference version available, only the journal version is kept;
- If a study has several versions published, only the most recent is kept;
- If a study exists in more than one source, only one copy is included.

To exclude ineligible studies, exclusion criteria were also defined:

- Studies that do not consider any of the thematics of this work;
- Studies that only focus on ethical or legal aspects of the thematics;
- Studies that are not peer-reviewed;
- Studies that are not written in English;
- Studies prior to 2010;
- Studies where the file cannot be obtained.

Table 3.2: Distribution of found and selected studies (Web Crawling)

Source	Studies Found	Studies Selected
ACM	202	13
IEEE Xplore	42	19
ScienceDirect	134	16
Scopus	220	29
Web of Science	69	9
Total	667	86

3.5 Studies Selection

Using the stated literature sources and the Web Crawling search string defined in section 3.3, 667 studies were found (as seen in Table 3.2).

In sum, 86 studies were selected after removing the duplicates, with an efficiency rate of 6.44% for ACM (13 studies), 45.24% for IEEE Xplore (19 studies), 11.94% for ScienceDirect (16 studies), 13.18% for Scopus (29 studies) and 13.04% for Web Of Science (9 studies).

Using the Chatbot Development search string, 170 studies were found (as seen in Table 3.3).

Table 3.3: Distribution of found and selected studies (Chatbot Development)

Source	Studies Found	Studies Selected
ACM	74	8
IEEE Xplore	6	3
ScienceDirect	65	8
Scopus	18	2
Web of Science	7	0
Total	170	21

After removing the duplicates and the articles without any available or free readable document, 21 studies were selected, with an efficiency rate of 10.81% for ACM (8 studies), 50% for IEEE Xplore (3 studies), 12.31% for ScienceDirect (8 studies), 11.11% for Scopus (2 studies) and 0% for Web Of Science (0 studies).

3.6 Quality Assessment

After selecting the studies for this review, a quality assessment checklist (Table 3.4) was built in order to clearly and systematically classify them. Studies with a score minor than 3 were considered ineligible for this literature review and were excluded.

With this minimum score required, we excluded from this review:

- Web Crawling: 10 studies out of 86 (11.63%)

Table 3.4: Quality Assessment Checklist

ID	Question	Score
QA1	Is the study published in a recognized journal or scientific event proceeding?	- Journals (JCR Ranking): Q1: 2 Q2: 1.5 Q3: 1- Q4: 0.5 No JCR: 0 - Conferences/Workshops (CORE Ranking): A: 1.5 B: 1 C: 0.5 No CORE: 0
QA2	Is there a clear statement of the aim of research?	Yes: 1 Partially: 0.5 No: 0
QA3	Is the experimental procedure carefully explained?	Yes: 1 Partially: 0.5 No: 0
QA4	Are the findings clearly stated and presented?	Yes: 1 Partially: 0.5 No: 0
QA5	Was the paper cited by other researchers?	Yes: 1 No: 0

- Chatbot Development: 2 studies out of 21 (9.52%)

The results of the quality assessment are detailed in Appendix A.

3.7 Data Extraction

To aid the data extraction process, a form that lists the key information to be collected from the gathered studies was built, helping to get a general view of the results. This form is described in Table 3.5.

3.8 Data Synthesis

Synthesising the data collected aims to summarise and report the relevant results of the analysed research. With this, this phase answers each one of the research questions established from the extracted data above and identifies any possible research gaps and recommendations.

Table 3.5: Data Extraction Form

Data ID	Data
D01	Study Identifier
D02	Authors
D03	Year of Publication
D04	Title
D05	Type (Journal, conference/workshop proceedings)
D06	Quality Assessment Score
D07	Thematic (Reliable Information Gathering, Information Processing or Chatbot Development)
D08	Tool (if applicable)
D09	Algorithm (if applicable)

3.8.1 Studies Overview

This overview will be divided in three parts, each addressing one of the areas related to the search queries defined: web crawling, text segmentation and classification and chatbot development.

3.8.1.1 Web Crawling

Regarding the Web Crawling part, we were able to produce the graphs seen on Figure 3.2 (year of the study's publication and its type: journal or conference), Figure 3.3 (language or tool on which the crawlers mentioned in the studies were developed) and Figure 3.4 (approach followed to crawl the web pages).

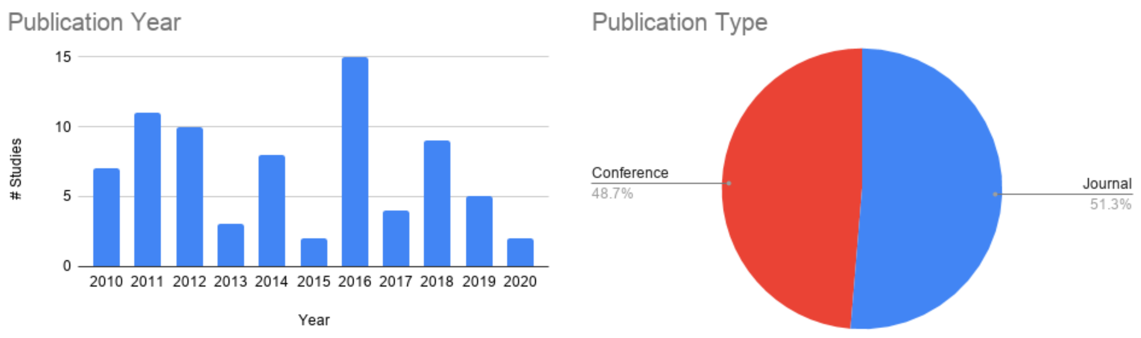


Figure 3.2: Web Crawling Studies: Year and Type of Publication

From the 86 studies that were selected, 11 were reviews or surveys on the matter, so these are not contemplated in the graphs regarding the language/tool and approach.

As seen on Figure 3.3, most of the studies do not mention the language or tool used to develop the crawler. However, when it is mentioned, we can observe that the most used language is Java (18 studies cite it or cite a tool developed in it: Nutch, ACHE, Heritrix, crawler4j and BUbiNG). The second most relevant is Python (4 studies cite it directly and 1 cites Scrapy which is built on it). The relevance of Java is due to its multithreading feature, allowing higher scalability and processing power, which results in more web pages crawled in a lower amount of time. Python can have its place here explained by its ease of use and lots of third-party packages that allow programmers to do almost everything (from NLP tools and machine learning algorithms and frameworks to web crawling applications).

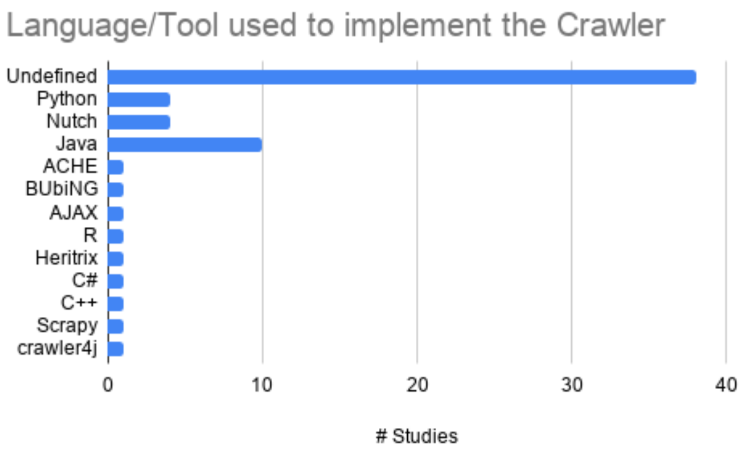


Figure 3.3: Web Crawling Studies: Language or Tool

In regard to the algorithm used for the web pages' classification or URL priority, the most used are Naive Bayes (a probabilistic machine learning algorithm, mostly used in NLP for text classification, due to its high scalability) and the TF-IDF metric (a numerical statistic that is intended to reflect how important a word is to a document in a collection), which is used in conjunction with other algorithms like HITS or LSI.

Nonetheless, many different algorithms are mentioned, each for a different purpose and approach. For example, some studies use machine learning algorithms to classify the web pages' content or URL while others use ontologies to provide the most relevant keywords or terms and then use VSM to compare and then filter the web pages related to the given ontology. Besides these two methods, other two are also mentioned in the studies: clickstream-based algorithms (that classify the web pages on a user-guided basis from previous actions) and clustering-based (which groups web page or URL features into categories, therefore being able to verify if they belong to a certain topic or not).

When it comes to the crawling approach, from Figure 3.4 we concluded that the most used is based on the web page's content (metadata, HTML tags and text), representing almost 33% of the studies' approaches. Some studies also combine the content with the URL for the classification, while others use only the URL. These three types represent circa 75% of the studies. However, there are also studies which combine the common approaches with anchor text and context, providing a seemingly more contextualized and focused way of classification. A minority set of studies take "unorthodox"approaches, like classifying the web pages based on context graphs, specific HTML elements or even on only HTML tags or terms.

3.8.1.2 Chatbot Development

Regarding the Chatbot Development part, we were able to produce the graphs seen on Figure 3.5 (year of the study's publication and its type: journal or conference), Figure 3.6 (language or tool on which the

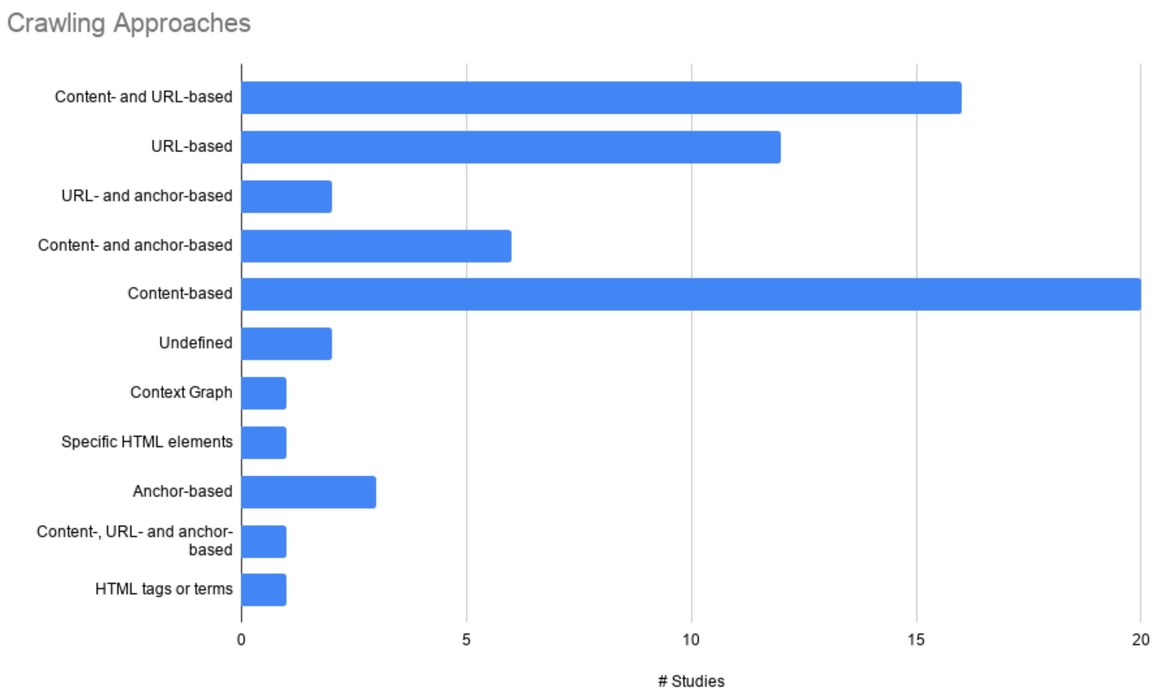


Figure 3.4: Web Crawling Studies: Approach

chatbot or its algorithm was developed), Figure 3.7 (algorithm/approach implemented in the chatbots) and Figure 3.8 (dataset type used for generating the answers).

From the 21 studies that were selected, 5 were reviews or surveys on the matter, so these are not contemplated in the graphs regarding the language/tool, algorithm and approach.

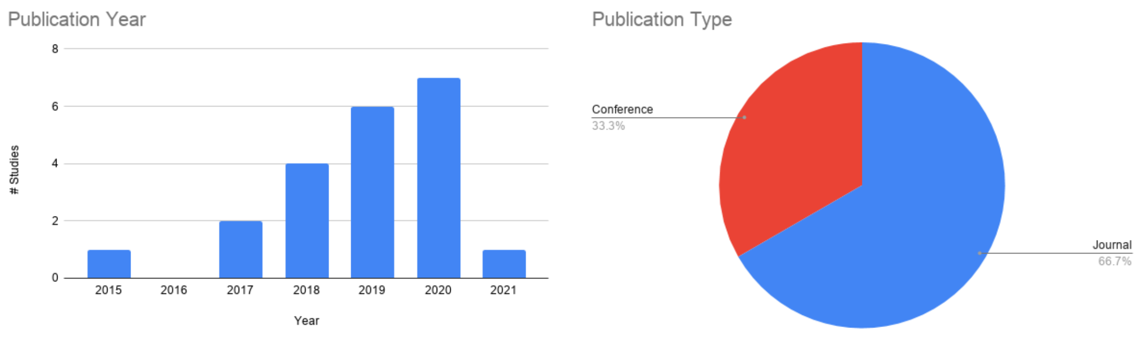


Figure 3.5: Chatbot Development Studies: Year and Type of Publication

After analyzing Figure 3.5, it is interesting to point out that even though having a time range for article inclusion from 2010 to 2021, the great majority of selected studies are very recent (the most prominent year was 2020).

Despite more than 30% of the studies not specifying the language or tool used to implement the chatbot, in Figure 3.6 we can observe that most of the other 70% of the studies used Machine Learning and Deep Learning framework based on Python, like Tensorflow (which was the most used), Keras and Pytorch. Theano and NLTK are also referred to as auxiliary tools in some processes. Google Assistant and

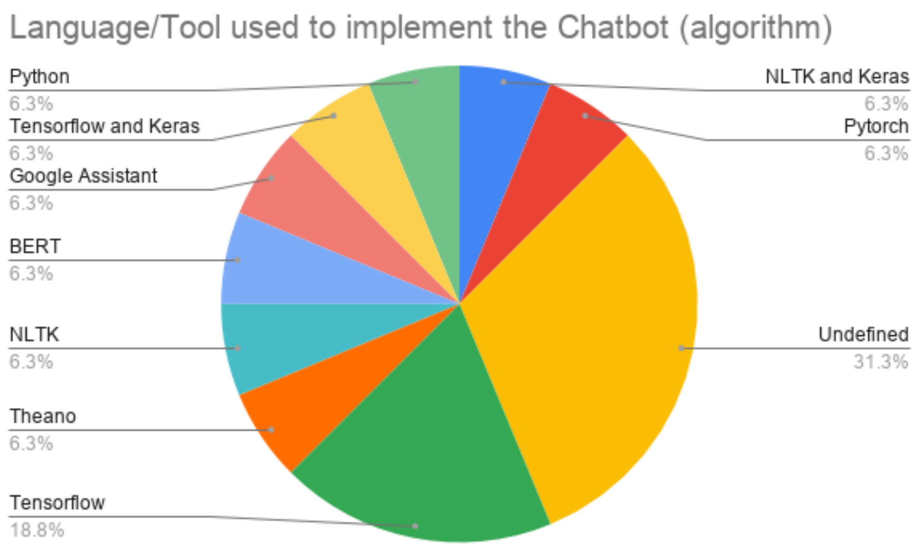


Figure 3.6: Chatbot Development Studies: Language/Tool

BERT (Google's Bidirectional Encoder Representations from Transformers) are used as complementary solutions for the already mentioned frameworks.

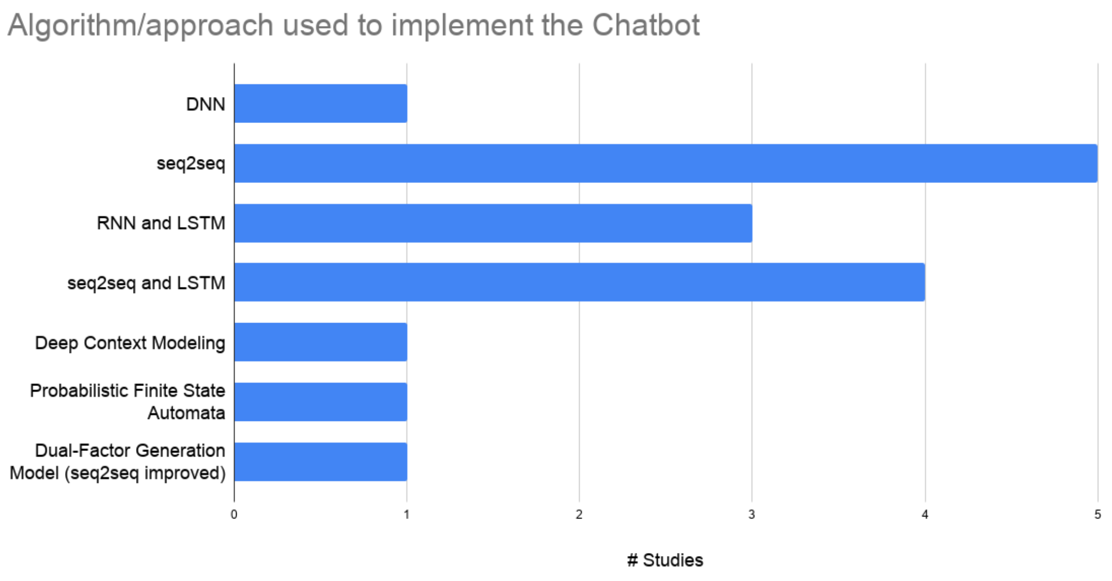


Figure 3.7: Chatbot Development Studies: Algorithm/Approach

Concerning the algorithms or approaches used, most of the selected studies were based on Artificial Neural Networks, in their variances of Deep Neural Networks (DNN) and Recurrent Neural Networks (RNN). In the 16 studies, 9 mentioned the use of the seq2seq (sequence to sequence) approach, which implements a RNN where the context for each item is the output from the previous one, leading to a conversation contextualization and follow-up. 7 studies also mentioned the use of LSTM (Long Short Term Memory), which is an RNN architecture used for deep learning. This distribution can be seen on Figure 3.7.

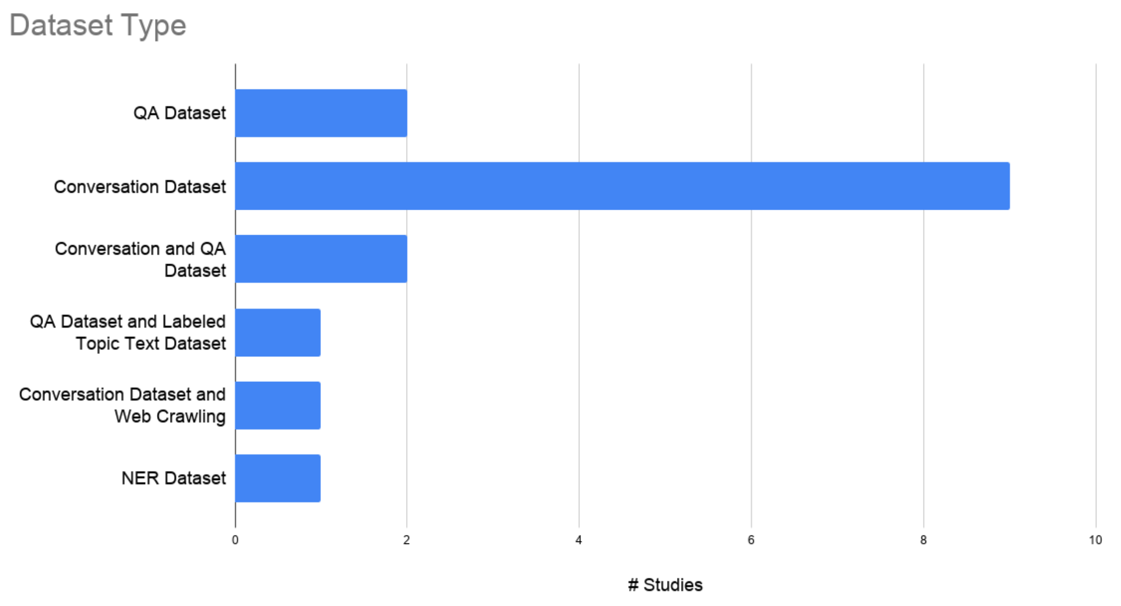


Figure 3.8: Chatbot Development Studies: Dataset Type

When it comes to the type of dataset used to feed the chatbots' algorithms, they can be divided into two major types: conversational datasets (e.g. based on movie transcriptions) and QA datasets (e.g. FAQs, forums). Based on Figure 3.8, we can conclude that more than a half of the studies used the first type of datasets, either alone or together with other types of datasets (like QA or web pages crawled from the internet).

3.8.2 Research Questions Findings

In this subsection, the research questions defined previously will be answered with the information retrieved from the selected literature.

RQ1: Which technique should be used to crawl the defined sources?

Unlike Kumar et al. (2017), we divided the studies found regarding focused crawling in categories based on the elements of the web page used for its classification of topic relevance. So, we found studies that classified them based only on its content (text), which was the most frequent method, based on the URL only and based on the anchor text and link context. Also, there are studies that combine two or more of the above. Along with these most prominent methodologies, we also found studies that used context graphs and metadata like HTML tags or terms.

While the approaches based on the page's content require its download, the approaches based on the URL alone can filter them before it is downloaded, leading to less processing effort.

Besides its category, most studies follow different approaches. The most common is the approach using soft computing, applying the Naive Bayes algorithm. However, there are many other approaches found, such as the combination of TF-IDF and other algorithms (like Latent Semantic Indexing, PageRank or even Naive Bayes), ontology-based (e.g. using semantic relevance), clickstream-based or using context graphs.

Starting with the content-based methodologies, Wang et al. (2018) applied to news-related websites the Latent Dirichlet Allocation (LDA) algorithm to classify different news through their titles' keywords. LDA is an unsupervised machine learning used for topic modelling that allows users to cluster a collection of documents so that more similar documents are grouped together and less similar documents are put into different categories.

Rheinländer et al. (2016) used the Naive Bayes algorithm to classify the main text of a web page, which was previously extracted and converted to a bag-of-words. Also using Naive Bayes, Wang et al. (2010) combines it with TF-IDF which is used to compute the weight of each word in the link context in order to calculate its crawling priority, while Amalia et al. (2016) suggested the use of Larger-Sites-First algorithm for the same purpose in the context of crawling health related articles. However, the use of the Larger-Sites-First did not bring a significant increase of related crawled pages. Saleh et al. (2017) implemented an optimized Naive Bayes algorithm, through the exclusion of outliers with a Support Vector Machine (SVM) optimization using a genetic algorithm and the selection of the most informative examples from the available ones.

When it comes to the URL-based classification, all the proposed crawlers differed in their approaches. It goes from a simple regex filtering process (Khalil and Fakir, 2017) to the usage of online incremental learning described by Singh et al. (2012), which sees URL-based topic classification as a stateless Reinforcement Learning (RL) problem. On the other hand, Rajalakshmi and Aravindan (2013) developed a crawler with the page classification being done with SVM and Maximum Entropy classifiers on character n-gram based features extracted from the URL, comparing both methodologies and obtaining similar results for both.

Zheng (2011) combined the genetic and ant algorithms, taking the advantages of the two algorithms to overcome their shortcomings. Feng et al. (2010) proposes a focused crawler with an algorithm called Navigational Rank, which computes each link's capability of leading to target pages by considering both the target and non-target pages it leads to using a Multinomial Naive Bayes as a classifier, but stating it can be used with any classification algorithm. Another approach is presented by Suebchua et al. (2018), which uses the already downloaded pages to estimate the relevance of another through the neighbouring feature.

Hernández et al. (2016) suggested a custom algorithm, using an unsupervised methodology based exclusively on URL features and being agnostic to variables like domain and language. Zhang and Lu (2010) focused on a semi-supervised clustering-based approach, with the usage of fuzzy class memberships and Q-values.

Considering only anchor text and link context in the topic relevance calculation, Zheng and Qian (2016) followed a keyword extraction approach based on the TF-IDF metric. Naghibi and Rahmani (2012) uses the vision-based page segmentation (VIPS) algorithm, which allows the web page to be divided into blocks using DOM in addition to some visual clues of the page's layout to detect related parts of the page, and then considering the text of a block as the context text of the links contained there. At last, a clustering-based approach is presented by Liu and Peng (2014), where a new kind of data structure called CFu-tree is used to speed up the process of hierarchical clustering.

As mentioned above, there are also studies combining these approaches. Using a domain ontology, Bedi et al. (2012) proposed a focused crawler that uses semantic similarity based on the Vector Space Model (VSM) derived from the web page's content text and URL, while a similar approach is followed by Du et al. (2015) but using the content and anchor texts instead. Torkestani (2012) also used the VSM, but combined it with a learning automata, developing this way an adaptive crawler that uses content text and the URL.

This last combination of content and URL is also present in the following studies: Deng (2020) (semantic similarity methodology using TF-IDF and HITS algorithm), Mali and Meshram (2011) and Tan and Mitra (2010) (both using the algorithm implemented in Google search engine PageRank, while the latter uses also a clustering approach combining it with TF-IDF), Taylan et al. (2011) and Pawar et al. (2016) (using a Naive Bayes classifier), Liu and Milios (2012) (that proposes two crawlers: one using Hidden Markov Models (HMM) and other using Conditional Random Fields (CRFs), both probabilistic methodologies) and Goyal et al. (2016) and Yan and Pan (2018) (both adopting the genetic algorithm, but the second presented some improvements to it).

Yet, regarding the use of context and anchor texts, Joe Dhanith and Surendiran (2019) suggested an ontology-based crawler using Normalized Pointwise Mutual Information (NPMI) and Resnik based semantic similarity algorithm, in an effort to mitigate the problems created when applying TF-IDF weights: the creation of severe deviations from the priorities of unvisited web pages and the fact that the similarity calculation only happens if the word occurs in the web page.

Using only the URL and the anchor and context texts is also possible, as demonstrated by Dahiwale et al. (2014) and Venu et al. (2016) (the first using a Naive Bayes classifier and the second the HITS algorithm for unsupervised domain ontology learning).

An approach using all of the above (URL, content text, anchor text and link context) is described by Xu et al. (2019), which uses the Mutation Improving Particle Swarm Optimization Algorithm, belonging to the learning focused crawler category. Seyfi et al. (2016) proposes an hierarchical structure called T-Graph to define the priority score for each unvisited link using specific HTML elements of a page to predict the topical focus of all the pages that have an unvisited link within the current page.

Each study has its advantages and disadvantages: there are simpler approaches, but most require a labeled dataset of previously collected web pages, while there are more complex algorithms but without the need of said dataset. Most of the proposed crawlers present a topic classification accuracy of over 80% which already represents a significant decrease in terms of number of pages to be later processed. However, the desired approach must use an unsupervised algorithm or another methodology which excludes the need of a pre-processed labeled dataset.

After analyzing COVID-19 related government websites, they all include some reference to it on the URL, because the link structures are organized in a centralized way so that people can easily access all the information about the disease in a simplified interaction. Thus, an URL-based crawler should be enough to filter the relevant pages. However, if the results are not satisfactory at the time of the implementation, another contextualization should be added to the algorithm, like including link context as a feature into the classification.

Another important point is the need to crawl international sources in English, but also national entities' websites in Portuguese, so the algorithm should be language-agnostic. This represents an obstacle to semantic approaches or ontology-based, given that at least two datasets would be needed, one for each language, with the increased difficulty of later expanding to other languages. At first, a regex approach will be followed, given the universality of the terms "COVID"and "coronavirus". However, if that is not enough to filter the adequate web pages, the tool proposed by Hernández et al. (2016) seems to be a good option, since it is unsupervised and language-agnostic.

RQ2: Which open-source or free tools are available for focused web crawling?

Although most of the studies either did not specify which tool they used for the development of the web crawler or developed an application with the wanted algorithm from root, the remaining studies mentioned only free and open-source frameworks:

- Nutch[1] - Nutch is a software project included in the Apache Foundation. It is coded entirely in Java and has a highly modular architecture. Nutch allows developers to create plugins for several purposes, e.g. media-type parsing, data retrieval, querying and clustering. Nutch is one of the most used and well-established web crawlers due to its high scalability, since it is built on Hadoop data structures, improving the performance of batch processing. However, Nutch alone is not a focused web crawler, but can be modified to work that way with the use of plugins. Nutch is used in Almuhareb (2016), Hassan et al. (2017), Rheinländer et al. (2016) and Pasari et al. (2016).

- ACHE[2] - ACHE is a focused web crawler developed in the Visualization, Imaging, and Data Analysis Center at New York University, which allows the collection of web pages that belong to a given domain or that contain user-defined rules. ACHE is built in Java and allows web page classification and URL prioritization. This tool is mentioned in Santos et al. (2016).

- BUbiNG[3] - BUbiNG is a web crawler tool developed by Boldi et al. (2018), in the Laboratory of Web Algorithmics from University of Milan. It is also built in Java and is a high-speed distributed crawler that coordinates autonomously to download data from the web.

- crawler4j[4] - crawler4j is an open source web crawler for Java which provides a simple interface for crawling the Web. The setup is very simple when comparing to other crawlers: developers define

[1]https://github.com/apache/nutch
[2]https://github.com/ViDA-NYU/ache
[3]https://github.com/LAW-Unimi/BUbiNG
[4]https://github.com/yasserg/crawler4j

two functions, *shouldVisit* (which sets if a given URL from a web page should be visited or not) and *visit* (which sets the action triggered when a web page is successfully downloaded). With the first function, a focused crawler can be deployed. Even though Crawler4j does not support distributed operations it can be scaled up by adding multiple threads. So, the performance and efficiency of crawler4j can be improved by increasing the number of crawler threads and reducing the politeness delay. It is used by Bedi et al. (2012).

- Heritrix[5] - Heritrix is another web crawler built in Java, but mainly designed for web archiving and it is maintained by The Internet Archive. Heritrix is extensible and a web-scale crawler. The definition of rules for the crawling process can be done, allowing thus focused crawling, although it is not as straightforward as other tools. Heritrix is used by Hao et al. (2011).

- RCrawler[6] - RCrawler is a R package developed by Khalil and Fakir (2017), that automatically traverses and parses all web pages of a website, and extracts all data you need from them at once with a single command. It also allows developers to build a network representation of a website's internal and external hyperlinks, aiding them to better understand a website's structure. Also, focused crawling is possible through URL and content-type filtering. Besides that, the main features of RCrawler are multi-threaded crawling, content extraction, and duplicate content detection.

- Scrapy[7] - Scrapy is one of the most used web crawlers, being built in Python. It allows crawling in a fast, simple, yet extensible way. However, Scrapy defaults are optimized for crawling specific sites, and only allows focused crawling by itself through URL filtering. This web crawler is cited in Xie and Xia (2014).

An overall comparison of the aforementioned tools is presented in Table 3.6.

Table 3.6: Web Crawling Tools Comparison

Tool	Language	Focused Crawler	Continuous Crawling	Parallel Crawling	Extensible	Content-based Crawling	URL-based Crawling	Mixed Crawling	Maintenance
Apache Nutch	Java	Partially	No	Yes	Yes	Yes	Yes	Yes	Yes
ACHE	Java	Yes	Yes	No	No	Yes	Yes	Yes	Yes
BUbiNG	Java	No	Yes	Yes	No	No	No	No	Yes
crawler4j	Java	Yes	Yes	Partially	No	No	Yes	No	Partially
Heritrix	Java	Partially	Yes	Yes	Yes	No	Yes	No	Yes
RCrawler	R	Partially	No	Yes	No	No	Yes	No	Partially
Scrapy	Python	Partially	No	Yes	Yes	No	No	No	Yes

After analyzing all the presented web crawlers, ACHE was the selected tool, since it has many features considered relevant to the scope of the project: continuous crawling, web page classifiers and URL prioritization to crawl only topic-related data (regarding COVID-19) and the indexing of the crawled pages with Elasticsearch. Also, the data processing step was not taken into account in this step as it can easily be

[5]https://github.com/internetarchive/heritrix3
[6]https://github.com/salimk/Rcrawler
[7]https://github.com/scrapy/scrapy

done in a module apart from the crawler itself. Another point that weighted favourably to ACHE was the fact that it is built in Java and allows fast and scalable processing (due to multi-threading).

RQ3: Which generative deep learning algorithms can be used for the chatbot application?

Neither of the studies selected followed the approach of having raw text extracted from the web as input for the intelligent self-learning algorithms used in the chatbots: they either used conversational datasets (like Whatsapp conversations in Chandra and Suyanto (2019) or English-French translation task from the WMT'14 in Nguyen and Shcherbakov (2018)), QA datasets (e.g. QA pairs from the Kaggle healthcare services competition in Vamsi et al. (2020) or Yahoo! Answers dataset in Gao and Ren (2019)) or a combination of both (Cornell movie dialogue and a custom-built insurance QA datasets in Nuruzzaman and Hussain (2020)).

Since in this work we do not want to start from a pre-built labelled conversational dataset or FAQ documents, none of the presented solutions can be followed "as is". However, we were able to identify some indirect alternatives which can be used to solve the problem of creating a chatbot based on raw textual data:

- Nuruzzaman and Hussain (2020) presented in their study a review of the existing dialog-based chatbot approaches, where they identified one that could be applied in this work (although they were not the main subject of their study). It consists of a search-engine based model, by applying indexing and search software (like Apache Lucene[8]) to the gathered web documents and then, given the users' queries, obtain the most relevant answer. Following this methodology, the web pages could be segmented in either sentences or paragraphs (using topic segmentation) and then indexed one by one. The multi-lingual requirement here would not be a problem, as the indexes would allow precise searches in either languages (English and Portuguese). However, a conversational dialog would not be possible with this approach *per-se*: an extra component would be needed in the application in order to allow responding to "chit-chat"interactions (e.g. greetings).

- The second alternative is based on the approach presented by Li et al. (2021). It uses BERT, Google's pre-trained language model. In our case, the model could be trained on QA datasets like SQuAD (Stanford Question Answering Dataset[9]), but there is the need for two different models: one for each language (English and Portuguese). Also, the textual data crawled from the web sources would be fed to the model and the answers to the users' questions would be retrieved from that data. The downside is the grammar inconsistency in the generated answers, as they are parts directly retrieved from the text. This approach is also partially applied by Barbosa et al. (2020) in their work.

- Another applicable approach identified was found in the study conducted by Adamopoulou and Moussiades (2020). It consists in using the Knowledge Connector BETA functionality of Dialogflow[10].

[8]https://lucene.apache.org/
[9]https://rajpurkar.github.io/SQuAD-explorer/
[10]https://cloud.google.com/dialogflow

Dialogflow (formerly known as api.ai) is a natural language understanding platform owned by Google, that allows the creation of conversational user interfaces. It supports more than 30 different languages, but it is mainly rule-based. On the other hand, the Knowledge Connectors functionality parses documents to find automated responses, without the need of pre-defined rules. One of its advantages is the fact that it keeps learning with the usage of the tool, providing better answers each time. The cons are that it is not entirely free and the functionality that is compatible with our use case is only on the BETA phase.

- Also, a fourth alternative is to combine both the search-engine based model and the BERT model. The raw textual data can be separated in paragraphs, which are then indexed by the search-engine tool and lately retrieved the best match to serve as the context paragraph where BERT will extract the answer to the provided user query. This way, a more precise answer is given and BERT computes on a smaller input.

After analyzing these four possibilities, we decided to follow the last described approach: search-engine based model combined with BERT, which allows less text processing and the generation of precise and concise answers. This model can be extended to other thematics easily, by changing the input textual data. However, a separation of processes will be needed to differentiate English and Portuguese sources/questions.

3.9 Limitations

The conducted SLR provides a solid overview of the existing literature around focused web crawlers and the associated chatbot implementation. However, there are some limitations to this research approach.

Without having a clear and precise search query (when there is not a deep understanding of the thematics to be researched), a significant number of studies will be found. As many of them can be found irrelevant after a deep analysis, it is difficult to retain the focus.

Also, filtering them with a Quality Assessment can exclude interesting studies that can provide a disruptive and innovative solution to the problem.

3.10 Conclusion

This systematic literature review presents an overview on how a chatbot application can be built based on crawled textual data from reliable sources, which can be very useful to fight misinformation dissemination (in this case, regarding COVID-19).

Firstly, a list of web crawling applications/tools was presented, as well as a concise comparison between them. Also on this topic, the existing approaches for focused crawling were discussed and ultimately a pair of tool and approach was defined by the authors as the go-to solution.

Secondly, four alternatives were identified for the development of the chatbot itself, fed by the crawled data. These alternatives were either indirectly extracted from the found studies or proposed by the authors, as there was not a direct approach for the problem being addressed in this study. All the studies found relied on a pre-build dataset of conversational text or QA pairs.

However, there might be the need for an additional review regarding text processing and text summarization. After analyzing the results presented in Chapter 6, future work is required to remove irrelevant textual information on the downloaded web pages.

Chapter 4

Architecture and Solution Design

This chapter presents the solution design for work, describing the architectures of the whole system and of its components: web crawler, text processor and chatbot application.

4.1 System Overview

With the state of the art described, a model of the system's architecture could be designed, consisting of three main components: Web Crawler, Text Processor and Chatbot Application.

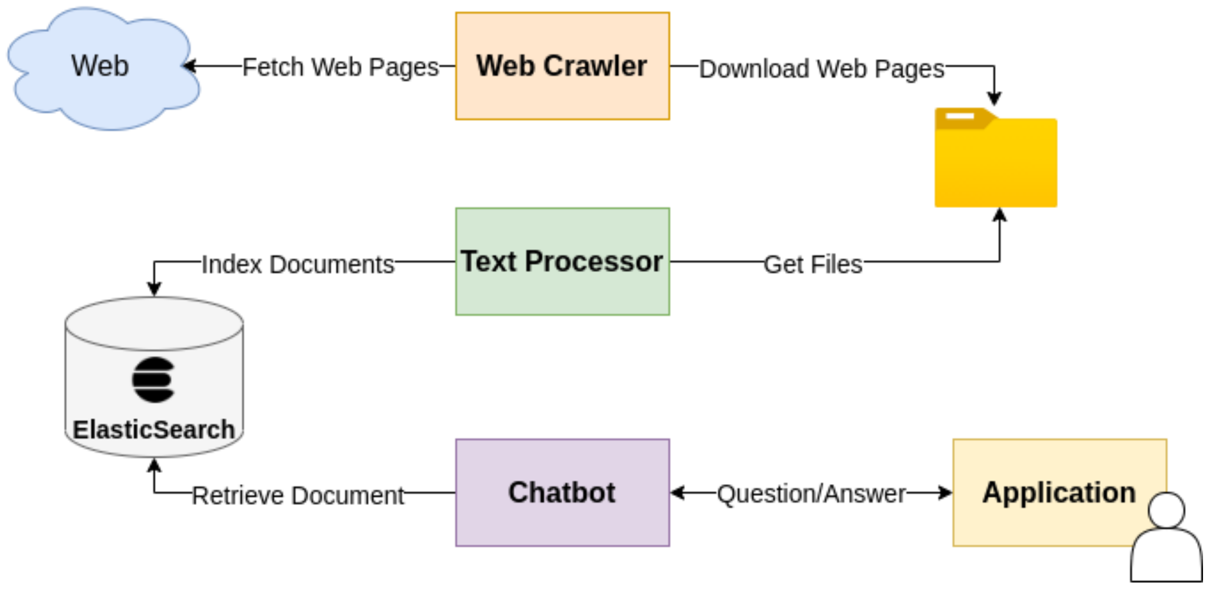


Figure 4.1: System Overview: High-Level Architecture

As seen on Figure 4.1, the system's behaviour starts with the Web Crawler fetching the relevant web pages from the Internet and then storing them on the local file system. The Text Processor parses the downloaded web pages, removing the HTML metadata and other irrelevant data and indexes the text in

ElasticSearch. The Chatbot Application interprets the users' questions, queries ElasticSearch for the most relevant document and then generates the answer using a trained BERT model, sending the response to the users as the last step.

Two flowcharts were developed to help understanding the flow of the information across the system: the first depicts the gathering and storage of the information (Web Crawler and Text Processor: Figure 4.2), while the second shows the fetching and generation of the answer to a given question (Chatbot Application: Figure 4.3).

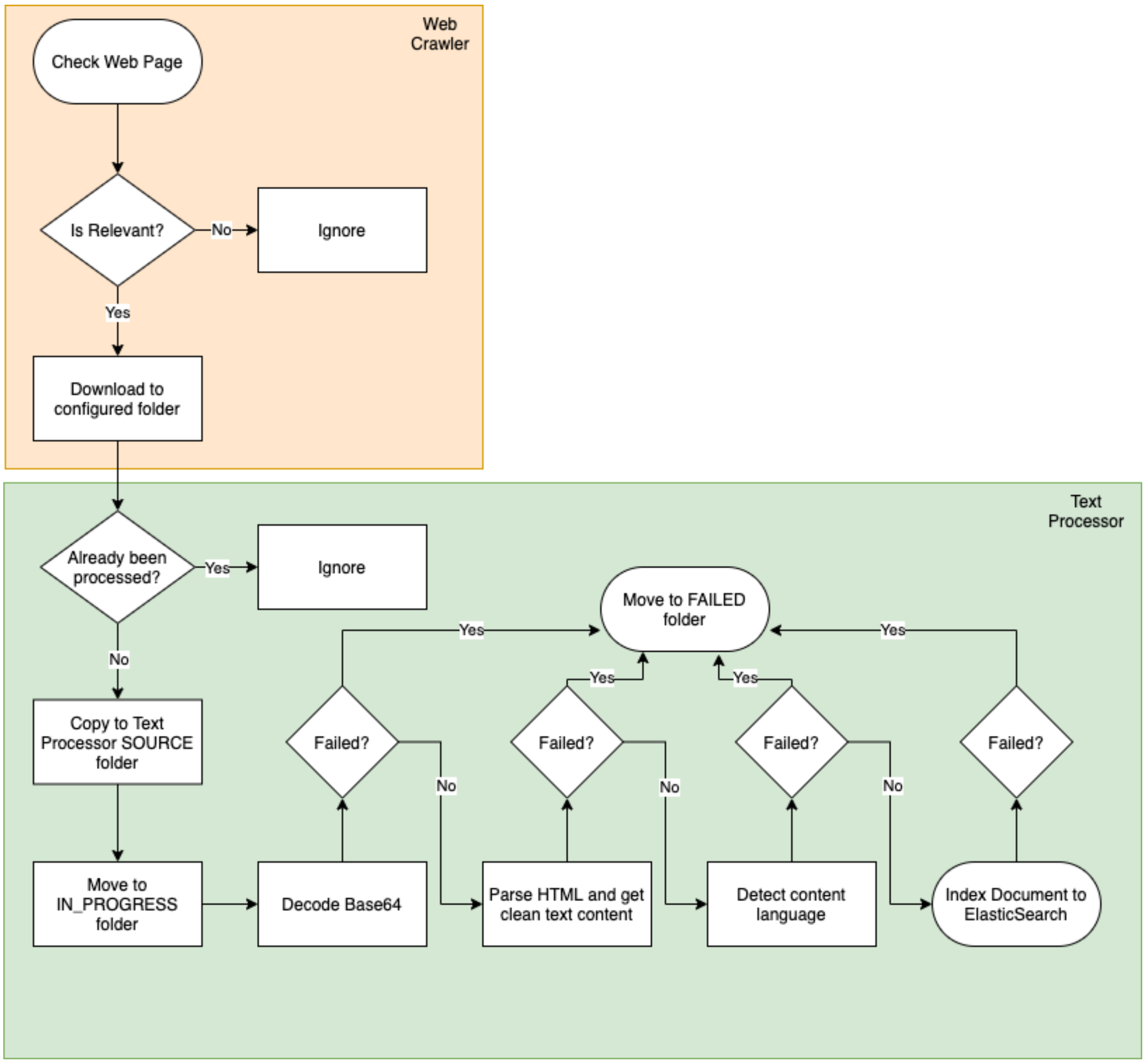


Figure 4.2: Information Flowcharts: Web Crawler + Text Processor

The architecture for each of the system's components is presented in detail in the next sections.

4.2 Web Crawler

The Web Crawler component is responsible for downloading the relevant pages from the reliable websites (which are passed to the web crawler as the start set of URLs to be crawled) and then storing them to

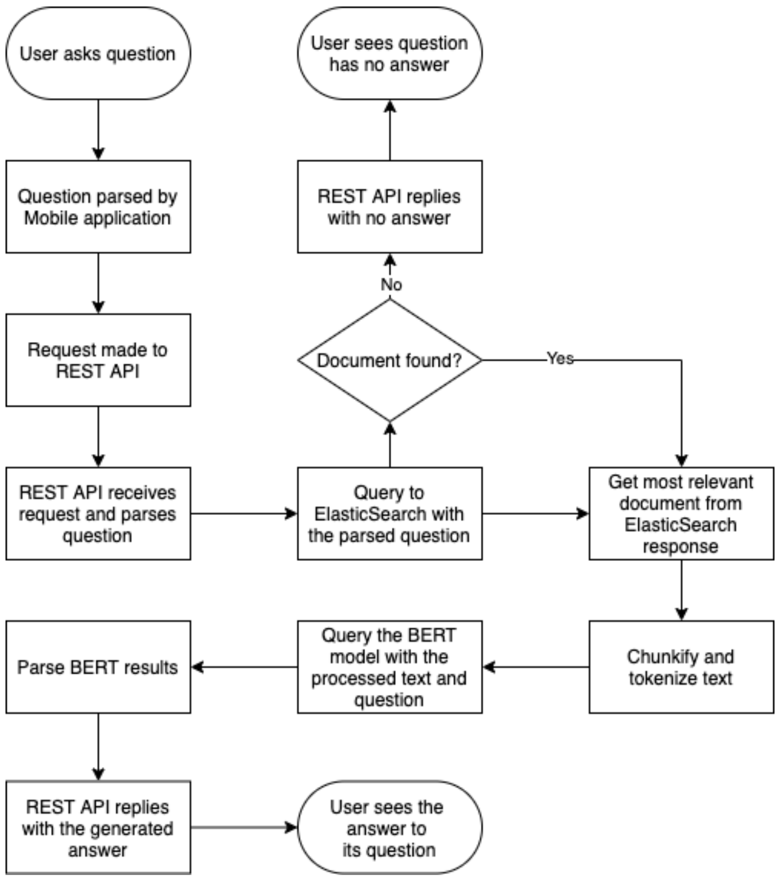

User asks question
Question parsed by Mobile application
Request made to REST API
REST API receives request and parses question
Query to ElasticSearch with the parsed question
Document found?
No
Yes
REST API replies with no answer
User sees question has no answer
Get most relevant document from ElasticSearch response
Chunkify and tokenize text
Query the BERT model with the processed text and question
Parse BERT results
REST API replies with the generated answer
User sees the answer to its question

Figure 4.3: Information Flowcharts: Chatbot Application

make them available for the Text Processor component in order to allow its information extraction.

As stated in section 3.8.2, the selected tool for web crawling is ACHE. It is developed in Java, allowing multi-threading and thus more efficient processing, with more pages crawled in a reduced amount of time. Also, it comes with multiple useful features, like the easy setup of the page classifier, in order to allow the focused crawling of the web pages.

ACHE can be deployed locally using a Docker container, with bound volumes connecting to the local file system (thus sharing files like the configurations and the output downloaded pages).

The ACHE crawler was developed to run with a set of parameters. The seeds file contains the list of websites that are going to be crawled. The crawler configuration file which is a YAML file with configurations like the output format and the crawling strategy. The filtering model is described also in a YAML file and it defines which pages are considered relevant or not. This filtering model can either be a regex validation on the web pages' URL, body or both, or a Machine Learning model previously trained. The configurations and filtering model is further discussed in Chapter 5.

In Figure 4.4, the Web Crawler's architecture is graphically presented.

The deployed crawler fetches the web pages, classifies them as relevant or not (following the given page classifier) and, if so, downloads and stores them in the local file system's defined folder.

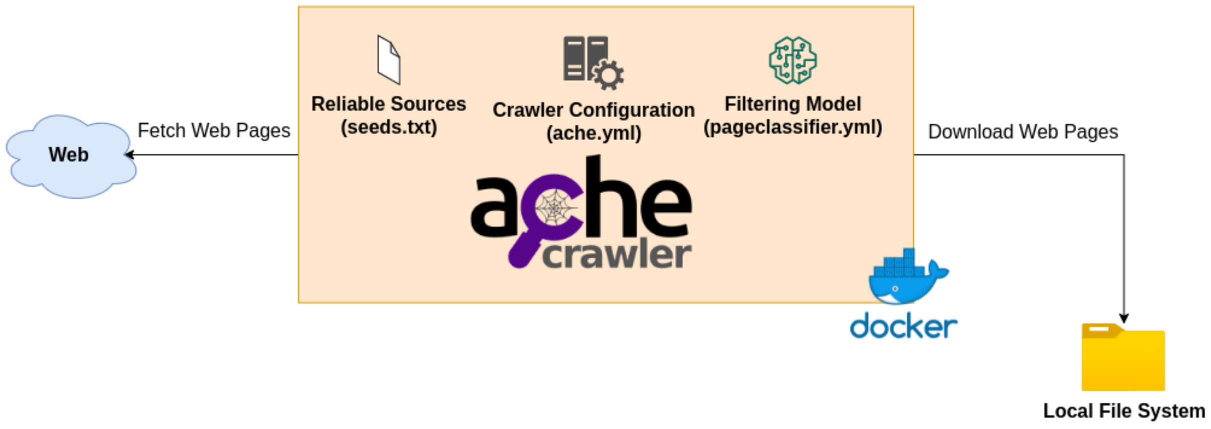

Web
Fetch Web Pages
Reliable Sources (seeds.txt)
Crawler Configuration (ache.yml)
Filtering Model (pageclassifier.yml)
ache crawler
docker
Download Web Pages
Local File System

Figure 4.4: Web Crawler: Low-Level Architecture

4.3 Text Processor

The Text Processor is responsible for extracting relevant textual information from the crawled web pages. The architecture for this component is presented on Figure 4.5.

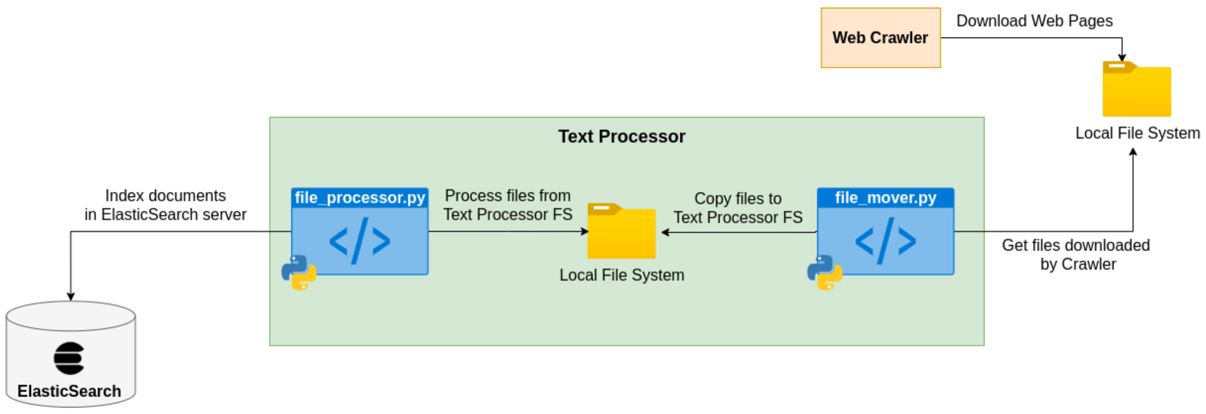

Web Crawler
Download Web Pages
Local File System
Text Processor
Index documents in ElasticSearch server
file_processor.py
Process files from Text Processor FS
Local File System
Copy files to Text Processor FS
file_mover.py
Get files downloaded by Crawler
ElasticSearch

Figure 4.5: Text Processor: Low-Level Architecture

In order to do what it is supposed to, this component is composed by two submodules:

- The File Mover, which fetches the downloaded web pages by the crawler and moves them to the Text Processor working area. This area is used to better manage file processing, keeping a consistent pipeline of successfully processed files and those that finished with errors.

- The File Processor, which does the text processing, consisting in parsing the files, decoding the Base64 strings (ACHE's predefined HTML output encoding) and extracting the textual content from the HTML (removing the tags and other metadata irrelevant for the chatbot). When the processing is complete, this submodule stores the generated document in an ElasticSearch server, which indexes it and allows a near real-time search of the documents. Thus, the bridge between the Text Processor's processed files and the chatbot application is built using ElasticSearch.

4.4 Chatbot Application

The Chatbot is the application that allows the interaction with users. It parses the users' questions and tries to answer it through a RESTful API. An overview of this component's architecture can be seen on Figure 4.6.

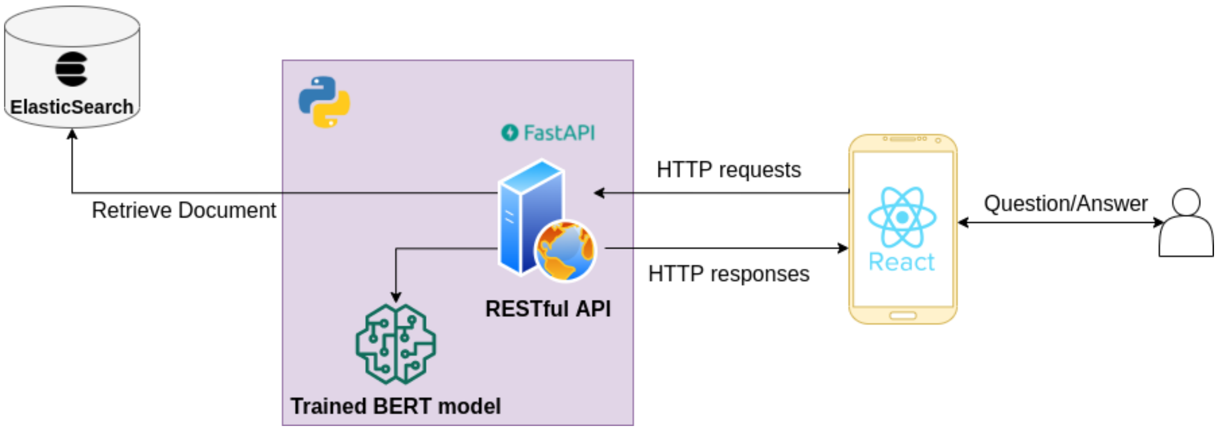


Figure 4.6: Chatbot Application: Low-Level Architecture

The user can interact with the application through a web application, for easier and simple access, given the fact that almost everyone nowadays possesses a device with internet connection. This application can be developed in React, allowing a straightforward development and little time to market. Also, React is one of the most popular frontend development frameworks, having a huge community and lots of documentation and support.

The application interprets the questions and passes them to the backend module, which will be presented as a RESTful API. This API can be developed using FastAPI. This framework allows the development of APIs in an agile way, with simple definitions and access management. Being based on Python, it can also be useful as there are many packages available for machine and deep learning.

The API will process the questions, query the ElasticSearch server for the most relevant document and provide it to the trained BERT model, which will then extract the answer to the given question. At last, the API returns it to the application, which displays the "conversation"on the users' phone.

BERT is the machine learning model that will be used to generate the answers for the given questions, using a context paragraph for the purpose. BERT is an open source framework for Natural Language Processing which is used to help understand the meaning of ambiguous language in text by using surrounding text to establish context (Lutkevich, 2020). BERT was trained using text from the Wikipedia (in one language or many, depending on the model) and can be fine-tuned using other text-based datasets, e.g. SQuAD (Stanford Question Answering Dataset).

BERT is based on a deep learning model called Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text (Horev, 2018). A Transformer model is composed by two distinct mechanisms: an encoder that reads the text input and a decoder that produces a prediction for the task. BERT is bidirectional in the sense that a sequence of words is read at once instead of sequentially, allowing the model to learn the context of a word based on all of its surrounding. Also, since BERT's objective is to generate a language model, only the encoder is necessary, thus its name of "Bidirectional Encoder Representations from Transformers".

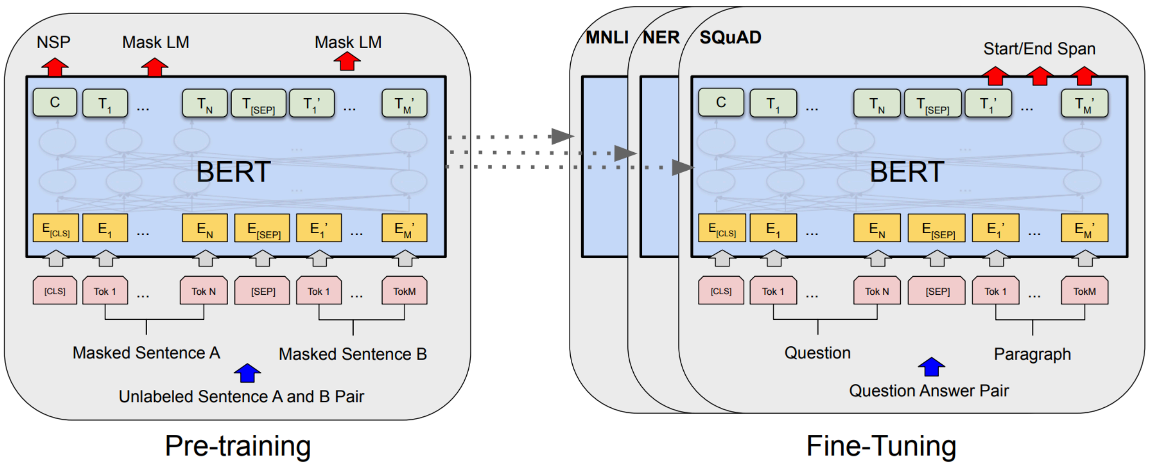


Figure 4.7: Overall Pre-training and Fine-tuning Procedures for BERT (Devlin et al., 2018)

BERT was first introduced by Devlin et al. (2018) and his colleagues from Google. In his work, the process of pre-training and fine-tuning BERT for question answering is described and can be observed in Figure 4.7. Apart from output layers, the same architectures are used in both pre-training and fine-tuning and are composed of two unsupervised tasks, the first being Masked Language Modeling (MLM) and the second Next Sentence Prediction (NSP).

Masked LM consists of giving BERT a sentence and optimizing the weights inside BERT to output the same sentence on the other side. While MLM teaches BERT to understand relationships between words, NSP teaches BERT to understand longer-term dependencies across sentences. NSP consists of giving BERT two sentences and it infers if one is subsequent to the other. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. The model calculates the probability of one sentence following the other using *softmax*.

Before entering the model, the inputs are processed as seen on Figure 4.8:

1. A "`[CLS]`"token is inserted at the beginning of the first sentence and a "`[SEP]`"token is inserted at the end of each sentence.

2. A sentence embedding indicating Sentence A or Sentence B is added to each token.

3. A positional embedding is added to each token to indicate its position in the sequence.

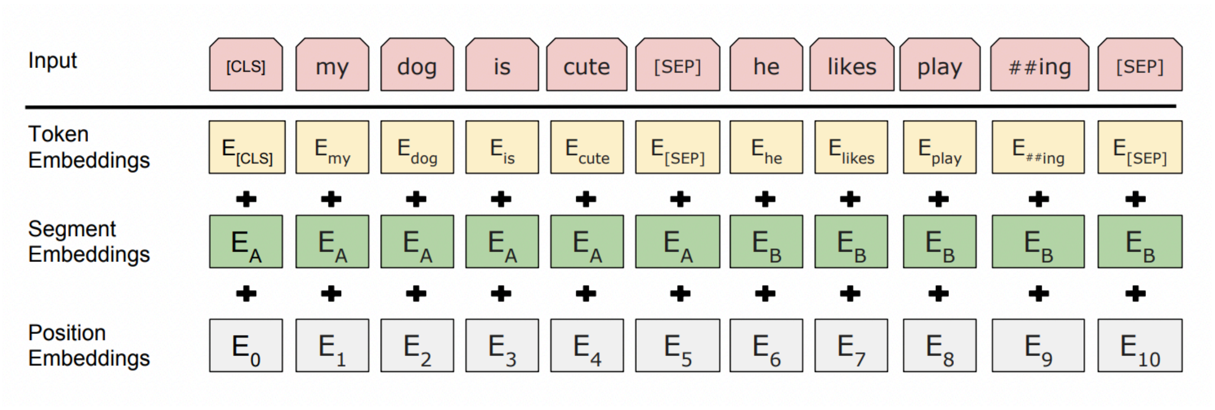

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 4.8: BERT Input Representation (Devlin et al., 2018)

Training BERT is done by predicting 15% of the tokens in the input, where the tokens are pre-processed by replacing 80% of the tokens with a "`[MASK]`"token, 10% with a random word and 10% with the original word.

Due to its great performance, by October 2020 almost every single English-based query was processed by BERT (Schwartz, 2020).

Chapter 5

System Implementation

This chapter describes the implementation process of the whole system, describing the issues that were faced and how we solved them. We focused on a bottom-up development, starting with the Web Crawler, then the Text Processor, and at last, the Chatbot Application (REST API and Web Application).

5.1 Web Crawler

Given that our whole system is based on the reliable information gathered, we started the implementation by developing the Web Crawler. The code is available in a public GitHub repository[1].

As said, we used the ACHE crawler to provide this functionality. In order to easen the use of the Web Crawler, we structured the project as follows:

- **/configs** - this folder will contain a subfolder relative to each project (in this case, we only have one called *qavid*. Each project should contain three files: `ache.yml`, `seeds.txt` and `settings.env`. The first one will contain the specifications used to run ACHE, the second the seed URLs used by the crawler to start fetching the pages and the last the environment variables used to allow a more customizable execution.

- **/models** - similarly to *configs*, the *models* folder will contain a subfolder for each project. Then, each project must contain the model used to classify web pages as relevant or not. This configuration should be present in a file called *pageclassifier.yml*.

- **/scripts** - The *scripts* folder contains Shell scripts used to install, start and stop the Crawler's execution. These are helpful given the fact that makes the connection through Docker abstract and direct.

[1]https://github.com/hpereira98/qavid19_crawler

5.1.1 Scripts

The ACHE Crawler can be installed using Docker, Gradle or Conda. Given our familiarity with Docker, and given the fact that it mainly eliminates the execution discrepancies between different machines and different operating systems and their versioning, we followed that approach. In order to facilitate the installation and usage of the tool, we developed scripts that assist this process.

To install the ACHE, one just needs to run `install_ache.sh`. This script verifies the dependencies (JAVA version greater or equal than 8 and Docker), and, if they are already installed, proceeds with the ACHE installation fetching the *latest* version of the Docker image[2]vidanyu/ache and then building the container.

To start the ACHE crawler once it is installed, one can run the script `start_crawler.sh <project> <classification_model>`, where *project* is the name of the crawling project to be ran (in our case, it will be *qavid19*) and *classification_model* is the name of the folder which contains the *pageclassifier.yml* (*url_regex* in our case). These parameters will be used to fetch the correct configuration files, either for the crawling execution and for the classification model. After loading the environment variables needed, the script will try to run the already built ACHE container.

Listing 5.1: Run ACHE crawler on Docker container

```
nohup sudo docker run \
   -v ${CONFIGS_FOLDER}:/configs/${CRAWLER_APPLICATION} \
   -v ${DATA_FOLDER}:/data/${CRAWLER_APPLICATION} \
   -v ${MODELS_FOLDER}/${CLASSIFICATION_MODEL}:/models/${CRAWLER_APPLICATION}/${
       ↪ CLASSIFICATION_MODEL} \
   -p 8080:8080 \
   --name ${DOCKER_CONTAINER_NAME} \
   vidanyu/ache startCrawl \
   -c /configs/${CRAWLER_APPLICATION}/ \
   -s /configs/${CRAWLER_APPLICATION}/seeds.txt \
   -o /data/${CRAWLER_APPLICATION}/ \
   -m /models/${CRAWLER_APPLICATION}/${CLASSIFICATION_MODEL}/ \
   &>${OUTPUT_LOG_FILE} &
```

As seen on Listing 5.1, it will create shared volumes on the folders configs, data and models, allowing files to be shared between our local machine and the Docker container. Also, we are setting the name to DOCKER_CONTAINER_NAME, which translates to *<project>_crawler* (in our case, *qavid19_crawler*). The other flags are setting the application, model and seeds configuration files locations. An important aspect is that this is executed using `nohup`, which makes the Docker container running in the background, while logging the output to the file specified as OUTPUT_LOG_FILE.

ACHE stores execution metadata, allowing us to stop and continue the crawler from the same state when it was stopped. In order to stop the container, one can use the script `stop_crawler.sh <project>`, which stops the Docker container called *<project>_crawler*.

[2]https://hub.docker.com/r/vidanyu/ache

However, if you want to fully restart your crawler, by removing the cached metadata and the downloaded pages, you can run `reset_crawler.sh <project>`. This will remove the Docker container called *<project>_crawler* and the downloaded web pages.

5.1.2 ACHE Configurations

ACHE is very rich when it comes to functionalities and customization. It allows us to chose the output file format, the classification model we want to use to classify relevant pages, data compression, limit pages to crawl (and per domain), hard focus (if the crawler should only crawl the seed domains), link classifier strategy and the crawler's agent, for example.

These configurations are all provided to the crawler through the *ache.yml* YAML file. For our *QAVID19* crawler, we set the configurations as seen on Appendix B:

- Regarding Target Storage, we are storing files as JSON, hashing the filenames and only storing pages classified as relevant. We are not hard focusing and not using a bipartite crawler. To limit the crawler execution, we set the maximum number of pages to visit as 1.000.000.000, to guarantee that we get the most of the seed web pages.

- Regarding Link Storage, we are setting the limit of pages from each domain to 1.000.000.000, and allowing the crawler to follow only forward links. The crawler can also fetch pages outside of the provided seed list, given that they are mentioned in the seeds. The crawler is using a random strategy when it comes to the order the pages are crawled, grouped with a Top K Selector. Also, we are configuring the crawler to recrawl sitemaps every 6 hours (360 minutes) in order to find new web pages.

- Regarding Crawler Manager, we set the user agent information like name (ACHE), URL (ACHE's GitHub project) and email (a80261@alunos.uminho.pt). We also set the number of threads used to download the web pages (100), as well as the maximum number of download retries (3). Another important aspect of this configuration is that our crawler only accepts the following MIME types:

  - *text/xml*
  - *text/html*
  - *text/plain*
  - *application/x-asp*
  - *application/xhtml+xml*
  - *application/vnd.wap.xhtml+xml*

  This way we guarantee we are not downloading binary data like PDF files or images.

Another required configuration we had to provide to ACHE in order for it to perform as a Focused Crawler was the URL seeds, which are the crawler's starting point. Thus, having the defined seed URLs defined in Section 2.1, we created the *seeds.txt* file containing what is seen on Listing 5.2.

Listing 5.2: ACHE Seed URL file

```
https://covid19.min-saude.pt
https://covid19estamoson.gov.pt
https://www.who.int/emergencies/diseases/novel-coronavirus-2019
https://www.cdc.gov/coronavirus/2019-ncov/index.html
https://www.nhs.uk/conditions/coronavirus-covid-19/
https://www.ecdc.europa.eu/en/coronavirus
```

5.1.3 Models

Having the base of the Web Crawler configured, we needed to configure the page classifier model, in order to filter the relevant pages. As specified in the RQ1 answer in Section 3.8.2, given the fact that most COVID-19 related pages are highlighted at the moment, they have most of the times a reference to it on the URL itself, thus a URL Regex filter should be enough to only obtain the relevant pages.

Following the ACHE documentation[3], we were able to define our classifier model in the file *pageclassifier.yml*. Listing 5.3 represents the mentioned file. As expected, we are filtering pages whose URLs contain the words ”covid”, ”corona”or ”coronavirus”.

Listing 5.3: ACHE Page Classifier Model

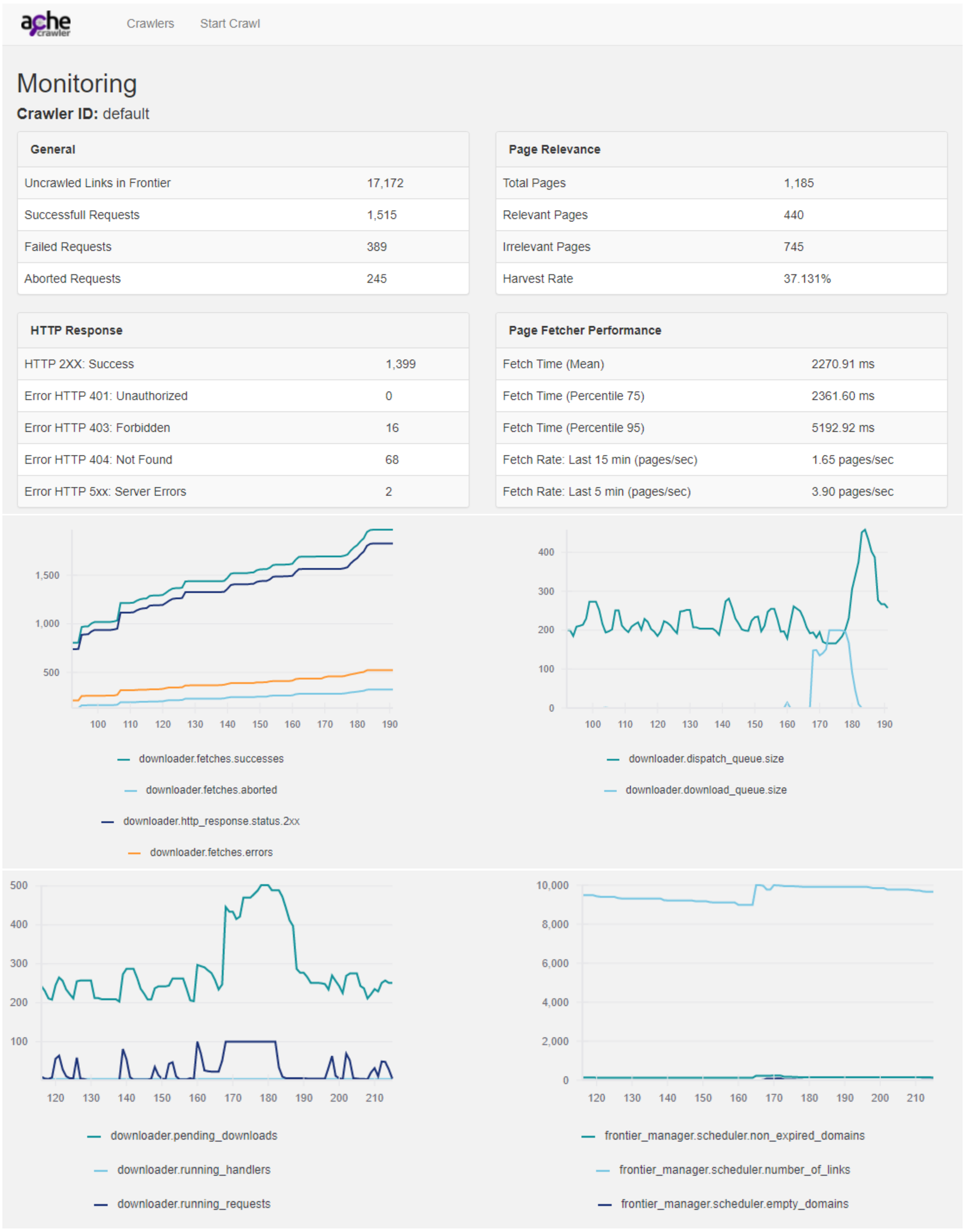
```
type: url_regex
parameters:
   regular_expressions: [
      ”.*covid.*”,
      ”.*corona(virus)?.*”,
   ]
```

5.1.4 Execution

ACHE Crawler provides a monitoring tool, which allows us to check the crawling progress and some statistics, like the percentage of relevant pages found, the number of failed downloads and the distribution of HTTP requests' answer codes, for example (as seen on Figure 5.1).

The downloaded files are JSON files, containing metadata along with the HTML data encoded in Base64. The metadata is composed by the URL, the content type, the response headers, the date it was fetched, the relevance score and other HTML related information (which is not very relevant for the purpose of this thesis).

[3]https://ache.readthedocs.io/en/latest/page-classifiers.htmlurl-regex


ache crawler
Crawlers
Start Crawl
100
110
120
130
140
150
160
170
180
190
200
210
500
1,000
1,500
0
100
200
300
400
500
2,000
4,000
6,000
8,000
10,000
downloader.fetches.successes
downloader.fetches.aborted
downloader.http_response.status.2xx
downloader.fetches.errors
downloader.dispatch_queue.size
downloader.download_queue.size
downloader.pending_downloads
downloader.running_handlers
downloader.running_requests
frontier_manager.scheduler.non_expired_domains
frontier_manager.scheduler.number_of_links
frontier_manager.scheduler.empty_domains

Monitoring

Crawler ID: default

General	
Uncrawled Links in Frontier	17,172
Successfull Requests	1,515
Failed Requests	389
Aborted Requests	245

Page Relevance	
Total Pages	1,185
Relevant Pages	440
Irrelevant Pages	745
Harvest Rate	37.131%

HTTP Response	
HTTP 2XX: Success	1,399
Error HTTP 401: Unauthorized	0
Error HTTP 403: Forbidden	16
Error HTTP 404: Not Found	68
Error HTTP 5xx: Server Errors	2

Page Fetcher Performance	
Fetch Time (Mean)	2270.91 ms
Fetch Time (Percentile 75)	2361.60 ms
Fetch Time (Percentile 95)	5192.92 ms
Fetch Rate: Last 15 min (pages/sec)	1.65 pages/sec
Fetch Rate: Last 5 min (pages/sec)	3.90 pages/sec

Figure 5.1: ACHE Crawler: Monitoring Tool

On a first run of the crawler, we deactivated the recrawling functionality and left it running for circa 4 hours. In this period of time, nearly 7.5GB of data was downloaded, which is translated to 34 484 web pages. From these, 219 pages were in Portuguese and 28086 were in English (which were the languages of the seed URLs). Despite having the seeds in two languages, we were able to fetch 2102 pages in French, 845 in Spanish, 462 in German and 882 in Indonesian. However, many pages were also found in other languages, while 30 we were not able to detect (either due to no text features in the web page or to text breaking characters not compliant with *utf-8*). We could conclude that without the hard-focus functionality, we were able to crawl a significant number of pages, but regarding the Portuguese language, the results were not very satisfactory. This could be overcame by keeping the crawler running for longer periods of time (in order to fetch more data in Portuguese) or by disabling the hard-focus functionality (only fetching data from the seeds, assuring we only kept pages in the language of the seed websites) or by limiting the crawler to follow links within a given ”hops”from the seeds (reducing the number of pages found in other languages but possibly shortening the information gathered).

This data will be used as the test basis for the chatbot application as it seems sufficient for the purpose. However, when it comes to properly testing the application, we may need to run the crawler for a longer period of time or apply the mentioned changes to the crawler’s configurations, allowing it to gather as much data as it is possible.

With the relevant web pages downloaded, and the mechanism to keep this process running developed, we were able to feed the next module of our system, the Text Processor.

5.2 Text Processor

Following the approach described in the Solution Design (Section 4.3), this module was divided in two independent parts: the File Mover and the File Processor, which are going to be described in the next sections. Both parts were developed using Python.

5.2.1 File Mover

The first part of the Text Processor module is responsible for copying the data downloaded by the Web Crawler into a specified folder. This is useful to guarantee that downloaded data is not processed more than once, which could create entropy and slow down the process.

The File Mover uses an auxiliary file called `processed_files.txt`, where it records the names of the files it has already copied.

In order to make sure the downloaded files are being processed continuously, the script is executed in an infinite cycle, executing every 2 hours and sleeping in the meanwhile.

5.2.2 File Processor

The second part of the Text Processor is the more important one, since it is responsible for parsing the text from the downloaded web pages and indexing it to ElasticSearch.

File Processor needs a set of 3 folders in order to execute:

- **IN_PROGRESS** - the script moves files from the *SOURCE* folder (File Mover's output directory) to *IN_PROGRESS* every two hours and then processes them;
- **FINISHED** - folder that contains the files successfully processed;
- **FAILED** - folder that contains the files where the processing failed.

This folder structure allows us to have the File Processor running continuously, while also allowing the failed files to be processed again.

The File Processor works as follows, after moving files into *IN_PROGRESS*:

1. First, each file is loaded into a Python dictionary, to facilitate data parsing.
2. Then, we decode the Base64 information in the dictionary's *content* field. This is done using the code in Listing 5.4.

Listing 5.4: Decode Base64 text

```
def decode_base64(str):
    base64_bytes = str.encode('utf-8')
    message_bytes = base64.b64decode(base64_bytes)
    message = message_bytes.decode('utf-8')
    return message
```

3. Next, the decoded text is parsed using a cleaning engine to extract the text only from the HTML page, thus removing its tags and irrelevant metadata. This is done using the Python library `html2text`. This engine was configured to ignore links, images, emphasis and tables.
4. Having the text cleansed and extracted from the web page, its language is detected using the Python library `langdetect`, that identifies the content's language seamlessly.
5. The language is then used to set the ElasticSearch index used to store the parsed text. This index follows, in this case, the format *covid_<language>*. This means that a page in Portuguese will be indexed inside *covid_pt* and an English page will be indexed in *covid_en*.
6. With the index defined, the page is actually indexed to ElasticSearch using their official Python API library. The web page is placed in the mentioned index, having as its ID the web page's URL encoded as Base64. Its body is composed of the parsed text content.
7. If this whole process is completed with success, the original file is moved to the *FINISHED* folder, else it is moved to *FAILED*.

5.3 ElasticSearch

ElasticSearch is the bridge between the Text Processor and the Chatbot application itself. It allows the chatbot to request the most relevant page for the given page, providing it as a context for the BERT model to extract the best possible answer.

In a first stage, the default configurations were used. While developing, a local instance was created, but it can easily be scaled to the Cloud (having the budget).

These default configurations include the following:

- The instance runs on `localhost:9200`.
- Before indexing, words are lowercased and splitted and punctuation is removed.
- Stopwords are not removed.
- ElasticSearch creates 5 primary shards and one replica for each index.

5.4 Chatbot Application

The Chatbot Application is the ”final”module of this system. It is responsible for making it available to the end users, by presenting an interactive and easy to use UI and by providing answers to the asked questions.

This module is composed by three main parts: the Web application (which is what the end user sees, it is the UI used to communicate with the application), the REST API (that receives the questions in the form of HTTP requests and responds to them, carrying the answer to the given interrogation) and the Chatbot Models (trained BERT models in each available language: English and Portuguese in this works' context). Next, each of these parts is described.

5.4.1 REST API

The REST API was developed in Python, using the FastAPI framework. The API is responsible for receiving the chatbot's questions and for providing the correct answer. As its purpose is solely the communication between the BERT models and the web application, it contains a very reduced set of endpoints. At the moment, it only contains two: one for checking the ElasticSearch instance status and another for receiving and answering the users' questions.

The API is structured in two modules, one for the Machine Learning models and another for the API itself (composed by the endpoint definition and connections to the defined services, e.g. ElasticSearch and the ML models).

When the API receives a question, one of the arguments must be the language the question is in. This is necessary for the API to request the right BERT model. Otherwise, given the duality of languages in this project, the answers would not be extracted successfully. Then, a query is made to ElasticSearch to the correct index (taking language into consideration), and thus fetching the most relevant web page for the provided query's words. The mentioned query is done as a match query in the *text* field of the index, in the format *"query": "match": "text": <query>*. This allows the API to get the context needed for the BERT engine to work properly and extract the answer. In order to communicate with the BERT model, the Python library *transformers* was used, along with *PyTorch*, the framework used to train the models.

Given the fact that the context may very well exceed most BERT models token limit (512), the approach defined in the Cloudera's Fast Forward Blog on NLP for Question Answering (*Building a QA system with Bert on Wikipedia*): the context is divided into chunks, which are then tokenized and provided the BERT model, which tries to extract answers from these chunks. However, this can create answer discrepancies if an answer can be extracted from an irrelevant chunk. For example:

```
Question: When was Barack Obama born?
Top wiki result: <WikipediaPage 'Barack Obama Sr.'>
Answer: 18 June 1936 / February 2 , 1961 /
```

Here, we can observe that for the Wikipedia page on Barack Obama Sr., the engine could extract two answers for the question "When was Barack Obama born?". This is because the model obtained an answer in the paragraph describing Barack Obama Sr. and another one in the paragraph describing its son, the former President of the United States of America. The code used for this mechanism is represented in Listing C.1 from Appendix C, which was adapted from the aforementioned blog.

To overcome this issue, we needed to follow a windowing approach, which would go over all the chunks but keeping some tokens from the previous chunk, thus preventing the context from being broken. This was achieved using the *pipeline*[4] functionality of HuggingFace's Transformers. However, as expected, the processing time is longer, but with the outcome of extracting a better answer, avoiding the context data being disrupted. On Listing C.2, the code developed for this approach is presented.

5.4.2 Chatbot Models

The Chatbot machine learning models (fine-tuned BERT in English and Portuguese) are the pieces that do the "magic"of providing the right answers to the provided questions.

BERT is used for many purposes, one of them being Question-Answering. However, it does not offer this functionality out-of-the-box. BERT models are made available with a diverse set of pre-training. In this work, we used the multilingual cased BERT model, which was pre-trained using the whole Wikipedia database in several languages. Despite being a much heavier model than the rest, it brings the advantage of allowing it to be trained in most of the languages, widening the range of our system if we later wanted to add more supported languages to it. These pre-trained models are made available publicly in the **google-research/bert** GitHub Repository.

[4] https://huggingface.co/transformers/main_classes/pipelines.html

Having the pre-trained model, we needed to fine-tune them using a QA dataset, enabling the model to extract answers from a context paragraph. For this, we trained our models using the SQuAD dataset[5], which contains 100000 questions and was developed by Rajpurkar et al. (2016). The SQuAD datasets are very complete, and contain context paragraphs, questions related to it and the correct answers. For the English model, we trained it using the second version of the dataset, SQuAD2.0. This dataset also contains 50000 unanswerable questions, making the model even more robust. However, we could not find a Portuguese version of this dataset. We tried to translate it, but the costs were too high, as no free platform was found and the Google Translator API presented several limitations, which did not allow the execution. Therefore, we used the SQuAD1.1 dataset, translated by the Google translation API and made available in the GitHub repository *nunorc/squad-v1.1-pt*. However, this dataset also presents some limitations: as it was automatically translated, and the dataset contains the starting position of the answer in the context paragraph, due to the differences between English and Portuguese, the starting position could not be validated. Thus, in these cases, the starting position was set to 0.

With the datasets and the pre-trained BERT model, we started to fine-tune the latter. This was done using the Python script provided by the Google Research team who developed BERT, called *run_squad.py*. This script, which was adapted from Tensorflow to PyTorch by HuggingFace (we decided to use this one given the fact we were also using PyTorch in the REST API), fine-tunes the given BERT model using a version of the SQuAD dataset. As said, the selected model was the multilingual cased BERT, which is called `bert-base-multilingual-cased`.

Given the fact that the multilingual cased BERT model was significantly heavy, the fine-tuning was slow. Lacking GPUs and local machines which could enhance the performance of the models' training, we had to set up a Virtual Machine within Google Cloud Platform (using the trial voucher of 300$). This VM also had no GPU, but since it was running no other significant process, we were able to reduce the execution time (from nearly 15 days on a local machine to 7 days). The VM was a *n1-standard-4* instance, with 4 vCPUs and 15 GB of memory.

The English model was trained with the specifications seen on Listing 5.5, while the Portuguese model with the ones seen on Listing 5.6.

Listing 5.5: Run BERT Fine-tuning (EN)

```
python3 run_squad.py \
   --model_type bert \
   --model_name_or_path bert-base-multilingual-cased \
   --output_dir models/bert/ \
   --data_dir data/squad_en \
   --overwrite_output_dir --overwrite_cache \
   --do_train \
   --train_file train-v2.0.json \
   --version_2_with_negative \
   --do_eval \
```

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

```
   --predict_file dev-v2.0.json \
   --per_gpu_train_batch_size 2 \
   --learning_rate 3e-5 \
   --num_train_epochs 2 \
   --max_seq_length 384 \
   --doc_stride 128 \
   --threads 10 \
   --save_steps 5000
```

Listing 5.6: Run BERT Fine-tuning (PT)

```
python3 run_squad.py \
   --model_type bert \
   --model_name_or_path bert-base-multilingual-cased \
   --output_dir models/bert_pt/ \
   --data_dir data/squad_pt \
   --overwrite_output_dir --overwrite_cache \
   --do_train \
   --train_file train-v1.1.json \
   --do_eval \
   --predict_file dev-v1.1.json \
   --per_gpu_train_batch_size 2 \
   --learning_rate 3e-5 \
   --num_train_epochs 2 \
   --max_seq_length 384 \
   --doc_stride 128 \
   --threads 10 \
   --save_steps 5000
```

The English model (trained on SQuAD2.0) took nearly 7 days to complete the training, while the Portuguese model (trained on SQuAD1.1) took circa 4 days to complete (since the dataset was smaller and less complex. These models trained for QA can be evaluated with two metrics: Exact Match and F1.

Exact match is a strict metric which states if for each question-answer pair, the characters of the model's prediction exactly match the characters of at least one of the True Answers.

F1 is a common metric used in classification models but also widely used for QA, calculated with precision and recall. Precision is the ratio of the number of shared words to the total number of words in the prediction, and recall is the ratio of the number of shared words to the total number of words in the provided answer (in this case, True Answer). F1, as in Wikipedia (2021), is calculated using the following formula:

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Table 5.1, Table 5.2 and Table 5.3 were obtained from the fine-tuning outputs, with the intent of helping us evaluate and analyze the performance of the BERT models. In these tables, the *total* values refer to the number of question-answer pairs used for that metric in the *dev* set.

Model	bert-base-multilingual-cased	bert-base-multilingual-cased
SQuAD Version	2.0 (original EN)	1.1 (translated to PT)
exact	73.19969679103849	50.74739829706717
f1	76.20200818956586	68.16061545510546
total	11873	10570

Table 5.1: BERT: Scores averaged over all examples in the *dev* set

In Table 5.1, the Exact Match for the English model was circa 73.2 while the Portuguese one got circa 50.75, with a significant loss compared to the first model. This can be due to the Portuguese translated dataset missing the correct answer positions. Regarding the F1 score, the English model got circa 76.2, where the Portuguese model got circa 68.16. We concluded that the results of the English model were satisfactory, compared with the results generally obtained with BERT and SQuAD 2.0 (presented in the dataset's web page). When it comes to the Portuguese model, despite having a low Exact Match, it can still provide a rather precise answer. If a version of the SQuAD dataset was available for the Portuguese language (with the correct answer positions and correct translation to European Portuguese), we believe the results would be better.

Model	bert-base-multilingual-cased	bert-base-multilingual-cased
SQuAD Version	2.0 (original EN)	1.1 (translated to PT)
HasAns_exact	68.35357624831309	50.74739829706717
HasAns_f1	74.36680891273862	68.16061545510546
HasAns_total	5928	10570

Table 5.2: BERT: Scores averaged over only positive examples (have answers)

Table 5.2 presents the results obtained from the question-answer pairs in the *dev* set which effectively had an answer. Since the SQuAD version 1.1 only had positive examples, the results for the Portuguese model are the same as the ones on Table 5.1 (this can be concluded also by the same *total* value). On the other hand, the English results were worse than the ones in Table 5.1, but still presenting an overall good performance regarding the F1 score (only 2 points lower, while the Exact Match value lowered by nearly 5 points). The total examples used in the English model were almost half of the dataset.

Model	bert-base-multilingual-cased	bert-base-multilingual-cased
SQuAD Version	2.0 (original EN)	1.1 (translated to PT)
NoAns_exact	78.03195962994113	-
NoAns_f1	78.03195962994113	-
NoAns_total	5945	-

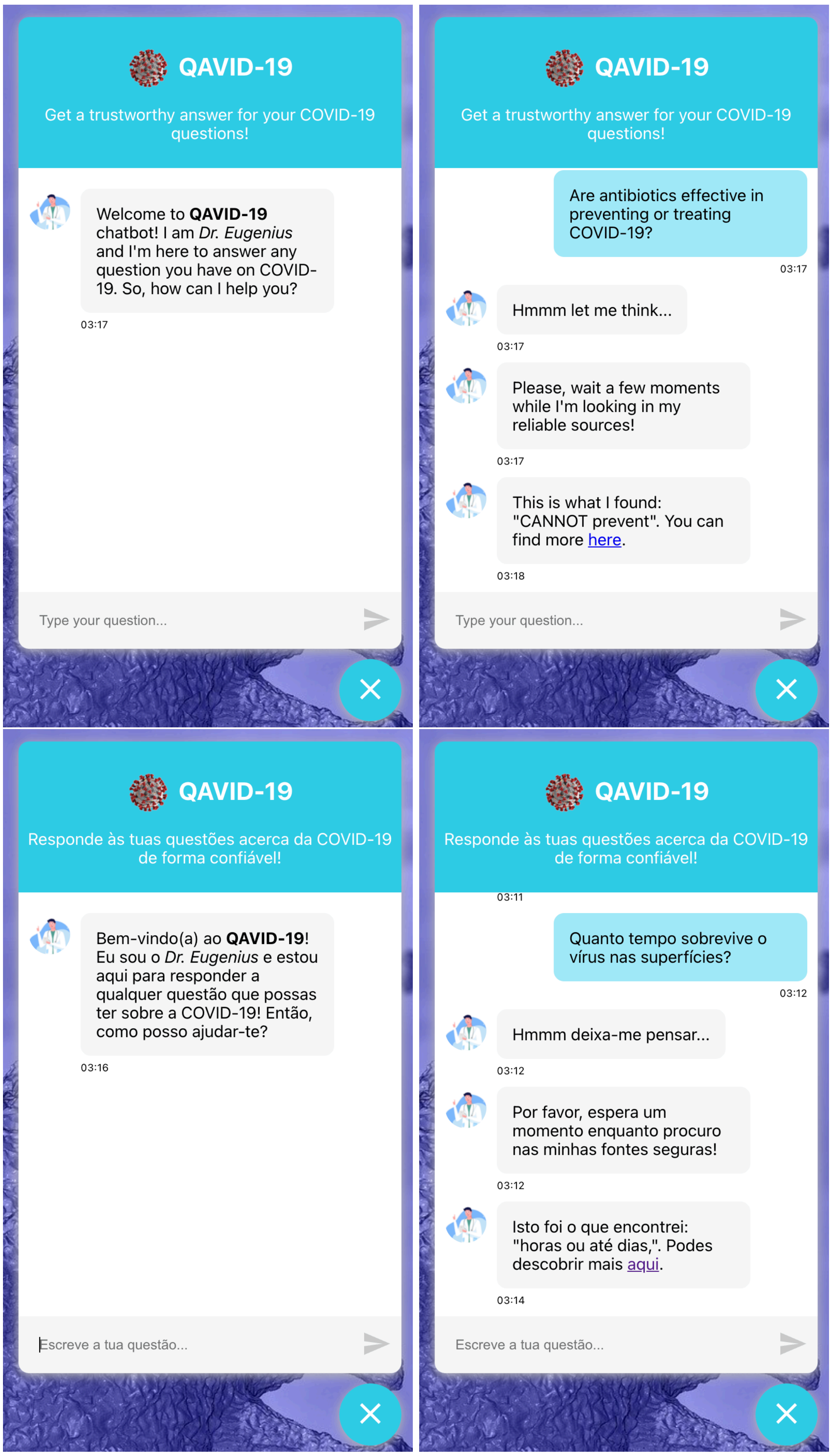
Table 5.3: BERT: Scores averaged over only negative examples (no answers)

When it comes to negative examples (question-answer pairs with no answer), Table 5.3 showed us that the English model improved its performance on the other half of the dataset, increasing the Exact Match value and F1 score to both 78.03. The Portuguese model had no outputs for these metrics due to the fact that the SQuAD version 1.1 does not provide negative examples.

5.4.3 Web Application

The web application was developed using React. It is a simple application (which can be used on any device with internet connection and a browser), containing a chat to allow users to interact with the chatbot.

The application first tries to check the user's device language (it uses English by default). Then, a UI is presented as shown in Figure 5.2. The user interacts with a virtual "doctor"called *Dr. Eugenius*, which is actually our chatbot. Any questions written in the application are transmitted to the REST API, which provides the answers. These are then printed on the chat-like application screen.



QAVID-19

Get a trustworthy answer for your COVID-19 questions!

Welcome to **QAVID-19** chatbot! I am *Dr. Eugenius* and I'm here to answer any question you have on COVID-19. So, how can I help you?

03:17

Type your question...

QAVID-19

Get a trustworthy answer for your COVID-19 questions!

Are antibiotics effective in preventing or treating COVID-19?

03:17

Hmmm let me think...

03:17

Please, wait a few moments while I'm looking in my reliable sources!

03:17

This is what I found: "CANNOT prevent". You can find more here.

03:18

Type your question...

QAVID-19

Responde às tuas questões acerca da COVID-19 de forma confiável!

Bem-vindo(a) ao **QAVID-19**! Eu sou o *Dr. Eugenius* e estou aqui para responder a qualquer questão que possas ter sobre a COVID-19! Então, como posso ajudar-te?

03:16

Escreve a tua questão...

QAVID-19

Responde às tuas questões acerca da COVID-19 de forma confiável!

03:11

Quanto tempo sobrevive o vírus nas superfícies?

03:12

Hmmm deixa-me pensar...

03:12

Por favor, espera um momento enquanto procuro nas minhas fontes seguras!

03:12

Isto foi o que encontrei: "horas ou até dias,". Podes descobrir mais aqui.

03:14

Escreve a tua questão...

Figure 5.2: Web Application: UI (English and Portuguese)

Chapter 6

Testing and Tuning

This chapter presents the results obtained from the Chatbot application and the refinements developed and applied to achieve better results.

6.1 COVID-19 QA Test Set

In order to test our application, a set of questions related to COVID-19 was obtained both for Portuguese and English languages from legit and trustworthy sources. This way, we were able to compare the official responses with the ones obtained using the developed chatbot.

The English FAQs is a subset of ten questions from the ones present on the United Nations' website (*Coronavirus disease (COVID-19) FAQs*):

ENQ1 How does COVID-19 spread?

ENQ2 What are the symptoms of COVID-19?

ENQ3 How do I know if it is COVID-19 or just the common cold?

ENQ4 Can the virus that causes COVID-19 be transmitted through the air?

ENQ5 What can I do to protect myself and prevent the spread of disease?

ENQ6 Should I wear a mask while exercising?

ENQ7 Are the symptoms of COVID-19 different in children than in adults?

ENQ8 Are antibiotics effective in preventing or treating COVID-19?

ENQ9 Can humans become infected with COVID-19 from an animal source?

ENQ10 How long does the virus survive on surfaces?

The Portuguese question set, in order to guarantee consistency, is the same as the English one. It is presented, as well as the answers to each of the aforementioned questions, on Appendix D.

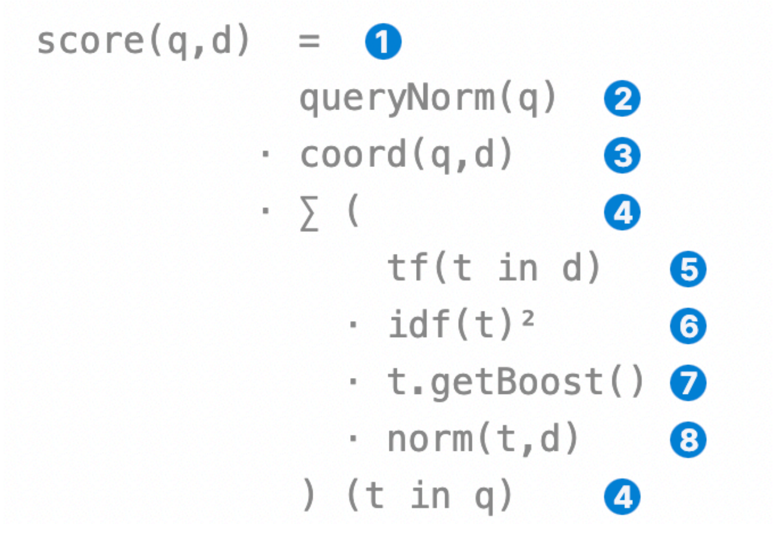
Before running our application with these question sets, we could identify some questions that could create some issues, e.g. the questions that do not mention COVID-19 or coronavirus can be answered with information written on the context of other diseases or viruses. Also, despite not being part of this question set, composed questions (more than one sentence) can also be tricky to the application.

6.2 Results

In this section, the results obtained are presented for each of the application's configurations.

In order to try to get the best answer, the top three documents were fetched from ElasticSearch (except in Section 6.2.3, where only the most relevant document is retrieved), and the answer with the best score is the one presented, as well as the BERT execution time for that document/answer pair. However, this approach led to a significant overall response time from 4 to 8 minutes for each request on the English QA Set (on the Portuguese QA Set the average was between 8 and 12 minutes).

Also, the tables with the results in the following sections present the ElasticSearch score obtained for the chosen document with the given query. The ElasticSearch scoring function is described in *Lucene's Practical Scoring Function* and it's calculated as seen on Figure 6.1.



```
score(q,d)  =  ❶
               queryNorm(q)  ❷
             · coord(q,d)    ❸
             · ∑ (           ❹
                   tf(t in d)    ❺
                 · idf(t)²       ❻
                 · t.getBoost()  ❼
                 · norm(t,d)     ❽
               ) (t in q)    ❹
```

Figure 6.1: ElasticSearch Scoring Function (*Lucene's Practical Scoring Function*)

Breaking the formula in Figure 6.1 into parts, we can see that:

1. *score(q,d)* is the relevance score of document *d* for query *q*;
2. *queryNorm(q)* is the query normalization factor (which is an attempt to normalize a query so that the results from one query may be compared with the results of another);

3. *coord(q,d)* is the coordination factor (which is used to reward documents that contain a higher percentage of the query terms);
4. The sum of the weights for each term *t* in the query *q* for document *d*
5. *tf(t in d)* is the term frequency for term *t* in document *d* (calculated with the square root of the number of times the term appears in the document);
6. *idf(t)* is the inverse document frequency for term *t* (the logarithm of the number of documents in the index, divided by the number of documents that contain the term);
7. *t.getBoost()* is the boost that has been applied to the query (parameter defined at search time to give one query clause more importance than another);
8. *norm(t,d)* is the field-length norm, combined with the index-time field-level boost, if any (calculated with the inverse square root of the number of terms in the field).

This score is merely used by ElasticSearch to calculate the most relevant documents indexed, it does not indicate if the answer to the given query is present in the document. So, the scores in the tables below are only indicative and cannot be used to draw conclusions on the answer obtained. However, if the scores were too low, we could deduct that no relevant documents could be retrieved from the indexes.

6.2.1 No Tuning (Raw Indexing)

Firstly, we ran the application with no tuning whatsoever. This means that the ElasticSearch indexes do not have any special configuration other than the defaults and the contexts (web pages) are used as they were indexed, without any further treatment.

Question ID	Answer	Answer Score	Execution Time (s)	ES Score
ENQ1	”more easily than SARS”	0.68192	116.753583	6.968636
ENQ2	”Coronavirus Disease”	0.99244	103.691477	5.240504
ENQ3	”missing it”	0.01309	57.702309	19.04191
ENQ4	”COVID-19 can spread via airborne transmission.”	0.14192	134.216533	13.339786
ENQ5	”Wash your hands,”	0.37165	43.985272	20.417625
ENQ6	”Compulsory”	0.43212	56.817268	19.259638
ENQ7	”very different”	0.70116	65.293769	13.397824
ENQ8	”preventing”	0.34662	92.693266	20.021086
ENQ9	”kids generally don’t seem to be getting severely ill.”	0.28995	104.239747	22.822405
ENQ10	”prolonged periods of time.”	0.98337	493.596096	19.434814

Table 6.1: No Tuning - Application Results on English QA Set

From the results presented on Table 6.1, we could observe some responses were correct, despite some being ambiguous (e.g. ENQ1 or ENQ2, which are not incorrect but also does not provide the required response) or incomplete (e.g. on ENQ5 the comma indicates the phrase could have more important information). Regarding ENQ6, the response was correct but for a period of time and on a given country (in this case, the answer was fetched from a news page about Singapore in April 2020). For ENQ3 and ENQ9, the answers obtained did not actually provide an answer to the questions presented. On ENQ7, the response was correctly extracted from the context, which stated that ”it is important to recognize that symptoms in children can be very different than in adults”, despite contradicting the response provided by the UN.

Question ID	**Answer**	**Answer Score**	**Execution Time (s)**	**ES Score**
PTQ1	”através de alimentos,”	0.99668	238.27001	6.22378
PTQ2	”Corona Virus Disease”	0.99431	75.942465	5.870456
PTQ3	”telefone.”	0.40931	237.712855	11.642396
PTQ4	”através de alimentos,”	0.99185	236.858443	14.410699
PTQ5	”lavar as mãos”	0.82095	275.750014	15.750288
PTQ6	”colaboradores”	0.00182	236.649484	12.931345
PTQ7	”Coronavirus Disease”	0.06816	166.522164	15.853691
PTQ8	”antibióticos não são efetivos”	0.02884	236.007342	11.729201
PTQ9	”coronavirus disease”	0.10511	49.596555	11.184871
PTQ10	”3 a 20 dias,”	0.99560	117.555118	17.520554

Table 6.2: No Tuning - Application Results on Portuguese QA Set

As for the Portuguese QA Set results, seen on Table 6.2, the outcomes were far from the expected. Most answers were wrong, not even answering the provided question. However, this is due to two situations: some pages returned by ElasticSearch were FAQ pages, which contained both the question and the answer, thus confusing our model which extracted the answer from the question and not from the answer (e.g. PTQ1 and PTQ4), and also the pages returned not containing the answers required (e.g. PTQ3 and PTQ6).

A relevant fact that was noticed was that some answers were being extracted from some dubious websites for the purpose. For example, for one question the answer was being extracted from an article called ”Overhyped Coronavirus Weaponized Against Trump”in the web page of a political show, which was a politically corrupted commentary, instead of scientifically correct data. Also, other answers were extracted from news sources (and also they were identified as belonging to private entities). This can lead to outdated and contextual information, as well as to manipulated data. Thus, the first approach to try to improve the chatbot's results were to restrict the crawler's application to only download pages from the trustworthy set of initial sources. The results for that approach are presented in the next section.

6.2.2 With Crawling Restrictions

As specified in the previous section, the first approach to improve the results relied on restricting the crawler's amplitude of pages to download: now it should only crawl web pages from the sources indicated as seeds, which we know are reliable sources of information and not politically corrupted.

This was done by setting the crawler configuration `link_storage.link_strategy.use_scope: true`. The crawler ran for circa 5 hours, having downloaded 3063 web pages. Of those, 2301 were in English and 553 in Portuguese. However, some pages were also found in other languages in the seed web pages: 9 in Spanish, 9 in French and 10 in Indonesian for example. Also, 171 pages had their language set as ”undefined”due to a language detection failure. These pages correspond to nearly 500MB of data, meaning that with an hour less of execution, the first iteration of the crawler gathered more 7GB of data than the curated iteration we just presented. Having the pages already indexed (we used a new naming for these pages, in order to maintain the previously indexed pages: *covid_<language>_v2*), we could run our QA Sets again, but now fetching the documents from the newly created indexes.

Question ID	**Answer**	**Answer Score**	**Execution Time (s)**	**ES Score**
ENQ1	”through faeces?”	0.79060	9.932024	4.313481
ENQ2	”coronavirus”	0.02161	4.308737	3.2946343
ENQ3	”Coronavirus disease”	0.25852	80.132636	17.5334
ENQ4	”Efforts to control COVID-19 transmission have reduced economic activity”	0.10635	29.505251	12.520907
ENQ5	”Read How to Protect Yourself to learn more.”	0.39509	163.305069	12.355144
ENQ6	”you should still keep physical distance from others as much as possible.”	0.11743	49.666623	13.833112
ENQ7	”they any different from grownups,”	0.13189	40.774625	10.547417
ENQ8	”CANNOT prevent”	0.07910	67.925064	11.104211
ENQ9	”Transmission could occur through direct contact with an infected individual,”	0.07502	20.81589	16.366974
ENQ10	”how long”	0.18234	35.32768	14.195935

Table 6.3: With Crawling Restrictions - Application Results on English QA Set

As seen on Table 6.3, with the crawling restrictions, the results on the English QA Set were still very ambiguous and incorrect on multiple questions, despite the fact we are now guaranteeing the trustworthiness of the information. We could still observe the answer being fetched from the question, instead of the answer, when the data source is a FAQ page (e.g. ENQ1). The execution times were better, but that only depends on the web page’s size. The scoring was very low, meaning the answer could not be successfully extracted for most cases. The results were again unsatisfactory.

Question ID	**Answer**	**Answer Score**	**Execution Time (s)**	**ES Score**
PTQ1	”para lhe fornecermos a melhor experiência de navegação.”	0.11047	11.724692	5.920704
PTQ2	”recuperação económica e social.”	0.12353	13.480936	4.981772
PTQ3	”desencadear efeitos indesejáveis.”	0.06333	100.835392	16.16233
PTQ4	”através de alimentos,”	0.96423	163.395218	10.164611
PTQ5	”ações preventivas habituais”	0.81296	165.699992	16.783964
PTQ6	”clientes,”	0.04648	162.602775	10.801659
PTQ7	”os dois vírus são muito diferentes”	0.03020	163.270843	12.067888
PTQ8	”antibióticos não são efetivos”	0.06301	163.275156	12.119408
PTQ9	”gotículas respiratórias.”	0.96316	163.912693	9.70346
PTQ10	”horas ou até dias,”	0.87411	163.424213	14.458093

Table 6.4: With Crawling Restrictions - Application Results on Portuguese QA Set

Regarding the Portuguese QA Set, the results obtained with the crawling restrictions are presented on Table 6.4. The execution times of the BERT answer extraction were mainly rounding 3 minutes, which is a very significant response time given the need for a quick answer. Also, we again observe the answer being extracted from the question of a FAQ (PTQ4), which resulted in a wrong and misleading answer. This issue cannot be overcome using the model we are providing and the web pages containing the questions themselves. Despite PTQ8 and PTQ10 having a correct answer and PTQ5 providing a generalist and imprecise answer, the others were not nearly correct.

Without changing the Text Processor logic, one more tweak we could provide to our system is the improvement of the ElasticSearch indexes. Since we are fetching the top 3 web pages from ElasticSearch, and then returning the answer with the highest score, the most relevant page can have its answer discarded (which could be the correct one) if the score returned by BERT was lower than a less relevant page. Thus, in order to guarantee that we retrieve the most relevant page effectively, and this way discarding the need of fetching the 3 pages (reducing the execution time by a third, given the fact that BERT will only execute over one page), we could enhance our ElasticSearch indexes by removing stop words (for English and Portuguese) and also by changing the tokenizer's approach to ”*whitespace*”instead of the standard one. This last enhancement prevents splitting terms with dashes, like ”COVID-19”, into two tokens, only splitting tokens that are separated by whitespaces. The results gotten with this approach are presented in the next section.

6.2.3 With Crawling Restrictions and ElasticSearch Indexes Enhancement

Trying to always fetch the most relevant web page for a given question while at the same moment reducing the execution times, we enhanced the ElasticSearch indexes, by changing the default analyzer and removing the stop words. This was done with the requests to our ElasticSearch instance seen on Listing 6.1, which creates new indexes with the naming ”covid_<lang>_v3”, keeping the previously created indexes untouched.

Listing 6.1: ElasticSearch Indexes Enhancement

```
curl -X PUT 'localhost:9200/covid_en_v3?pretty' -H 'Content-Type: application/json' -d '
{
 ”settings”: {
   ”analysis”: {
     ”analyzer”: {
       ”covid_en_analyzer”: {
         ”type”: ”whitespace”,
         ”stopwords”: ”_english_”,
         ”ignore_case”: true
       }
     }
   }
 }
}'

curl -X PUT 'localhost:9200/covid_pt_v3?pretty' -H 'Content-Type: application/json' -d '
{
 ”settings”: {
   ”analysis”: {
     ”analyzer”: {
       ”covid_pt_analyzer”: {
         ”type”: ”whitespace”,
         ”stopwords”: ”_portuguese_”,
          ”ignore_case”: true
        }
      }
    }
  }
}'
```

The stop word lists (English and Portuguese) are the default ones defined in the *Lucene* repository as stated in the official ElasticSearch documentation (*Stop token filter: Elasticsearch Guide [7.15]*). In order to guarantee all the stop words are ignored, we used the ”ignore case”option (this ignores stop words regardless of their letters' casing). After being created, the documents were reindexed to the newly created indexes, as seen on Listing 6.2.

Listing 6.2: ElasticSearch Portuguese Index Enhancement

```
curl -X POST ”localhost:9200/_reindex?pretty” -H 'Content-Type: application/json' -d'
{
  ”source”: {
    ”index”: ”covid_en_v2”
  },
  ”dest”: {
    ”index”: ”covid_en_v3”
  }
}
'

curl -X POST ”localhost:9200/_reindex?pretty” -H 'Content-Type: application/json' -d'
{
  ”source”: {
    ”index”: ”covid_pt_v2”
  },
  ”dest”: {
    ”index”: ”covid_pt_v3”
  }
}
'
```

Having the new indexes created and the documents reindexed to these, we could extract and analyze the results obtained with this refactoring.

As seen on Table 6.5, the execution times were greatly reduced, and even more considering this was the sole execution of the BERT model. However, the answers themselves remain poorly extracted. Only two of them were actually correct (with one being too vague), while the other were extracted from questions or provided a wrong answer for the question (we got three questions with the answer ”coronavirus”or ”coronavirus disease”). The scores obtained were also significantly low (with more than 60% being below the 0.1 mark) The fact that this is faster does not compensate for the wrongly generated answers.

Question ID	Answer	Answer Score	Execution Time (s)	ES Score
ENQ1	”through faeces?”	0.72600	15.667076	4.4349594
ENQ2	”coronavirus”	0.02161	4.609956	3.2946343
ENQ3	”Coronavirus disease”	0.25852	81.557437	17.5334
ENQ4	”Coronavirus disease”	0.00565	29.834129	12.521194
ENQ5	”Read How to Protect Yourself to learn more.”	0.39509	162.154876	12.355144
ENQ6	”fabric mask,”	0.00407	22.504819	14.622446
ENQ7	”even when the employee or a family member are exhibiting symptoms compatible”	0.06033	49.545466	11.737822
ENQ8	”CANNOT prevent”	0.07910	68.100438	11.104211
ENQ9	”Transmission could occur through direct contact with an infected individual,”	0.07502	20.84661	16.366974
ENQ10	”how long”	0.18234	35.314053	14.195935

Table 6.5: With Crawling Restrictions and ElasticSearch Indexes Enhancement - Application Results on English QA Set

Question ID	Answer	Answer Score	Execution Time (s)	ES Score
PTQ1	”saude.pt/?p=3579435”	0.06071	8.271065	8.08448
PTQ2	”Apoios Eletricidade”	0.00008	6.323288	5.1191373
PTQ3	”pandemia da doença COVID -19.”	0.01162	34.272169	19.957014
PTQ4	”através de alimentos,”	0.94156	169.374324	10.164611
PTQ5	”ações preventivas habituais”	0.62834	162.131818	16.783964
PTQ6	”colaboradores”	0.00692	163.114517	10.801659
PTQ7	”os dois vírus são muito diferentes”	0.03012	163.356272	12.067888
PTQ8	”antibióticos não são efetivos”	0.06301	613.835261	12.119408
PTQ9	”gotículas respiratórias.”	0.96316	833.484162	9.70346
PTQ10	”horas ou até dias,”	0.85417	166.543576	14.458093

Table 6.6: With Crawling Restrictions and ElasticSearch Indexes Enhancement - Application Results on Portuguese QA Set

When it comes to the Portuguese results, observed on Table 6.6, they were even worse than the previously obtained answers: we got a part of an URL now (PTQ1) and despite the high score PTQ4 and PTQ9 are wrong. Again, PTQ4 was again extracted from the question on the FAQ page. PTQ2 is nowhere near the requested answer, with its meaning being far away from the question. On the other hand, an improvement on the execution times was noted, despite an outlier with more than 800 seconds of execution (PTQ9), that translated to a wrong answer at the same time.

6.3 Outcome Analysis

Comparing the results obtained with the three described iterations (No Tuning, With Crawling Restrictions and With Crawling Restrictions and ElasticSearch Indexes Enhancement), we can conclude that:

- for the English QA Set, the answers obtained were very distinct between the first and the other two iterations. This is due to the fact that the web pages that served as input were reduced to only the reliable sources. However, the answers were better extracted in the first iteration, providing more meaningful responses to the questions presented. On the other, those could be misleading since they were retrieved from dubious websites, not considered trustworthy or having political interest conflicts. Regarding the second iteration, only four answers could be used to respond to the query in a correct or incomplete way. As for the third iteration, only two could be used.

When it comes to the BERT scores outputted, they were significantly higher than the ones obtained in the second and third iterations. Nonetheless, this is merely a score on how BERT could retrieve the answer from the context. A correct answer could have a lower score if BERT did not consider it ”easy”or direct to retrieve according to its training and fine-tuning. The execution times were also greatly reduced, moving from the minimum of 43 seconds in the first iteration to circa 4 seconds in the second and third iterations).

- for the Portuguese QA Set, the outcomes are similar to the ones obtained for the English QA Set, with the answers gotten in the first iteration diverging largely from the two latter ones. The first iteration produced the best results, but with the drawback of having not so trustworthy sources. Also, only 3 answers were correct or incomplete, the others having a context far away from the required answer. The fact that some answers were extracted from FAQ questions led to a number of incorrect and misleading responses. The results worsen on the second and third iterations, having in the last one part of an URL as the answer. The number of correct answers declined from 3 to two in the second and third iterations.

  Regarding the BERT scores, they went downhill between iterations, mirroring the difficulty of the model in extracting the correct answer (in the third iteration, we got a BERT score of circa 0.000075 for the most relevant retrieved web page, which did not contain the answer in it). However, not everything is bad. The execution times reduced significantly (from the minimum 49 seconds on the first iteration to nearly 12 seconds in the second and circa 6 in the third), but it does not compensate for the fact that the answers were not correct.

When it comes to the ElasticSearch scores, no relevant difference could be observed between the last two iterations (the first one is not comparable given the fact that the documents indexed were different collections with significant differences in volume).

In conclusion, without a better treatment of the web pages, the BERT model cannot successfully extract or generate the correct answer. Also, the fact that running the model requires a windowing approach rather than processing the whole context brings limitations like the breakage of context and the upper execution times needed. To improve the results, we could also use another, more evolved, transformer model, or even BERT with a better fine-tuning, with more resources and more epochs for instance.

Chapter 7

Conclusions

In this final chapter, the work is concluded, making an overview of what was done and accomplished and describing the future work that could be done to improve it and obtain better results.

7.1 Conclusion

Chatbots can make a huge difference in the modern world. They can facilitate communication between people and services, helping them answer all of their doubts or questions, and even allowing them to take several actions for them, e.g. booking a flight. However, these chatbots need to be well trained and fine tuned to each expected task. This dissertation focused on chatbots that could lighten the weight put on Health Services in the situation of a new disease or health concern that could create a lot of doubts, even more in a time where social media is used a mean to spread fake news and false information, leading people to commit action that can put theirs lives at risk. The context used in this work was the COVID-19 pandemic, given its contemporaneity.

This dissertation described how an intelligent chatbot could be built using solely the information available on trustworthy sources of information. In the context of health services, these sources were the governmental and organizational websites, like the Health departments of a country or of the United Nations or World Health Organization, for example.

After analyzing the application that was aimed to be developed, it was established that such needed to be composed by three main components: the web crawler (responsible for downloading the relevant pages from the reliable sources), the text processor (which would parse the downloaded pages and store only the textual information, discarding any metadata) and the chatbot itself (consisting in a Machine Model that could extract the answer from the stores pages and a UI that allowed users to interact with the application).

To understand what technologies and approaches should be followed, a systematic literature review was conducted. With this procedure, an extensive state of the art collection was produced. However, none of the studies analyzed provided a solution to the problem that this work wanted to solve: the creation of a chatbot that could answer questions based on raw data extracted from web pages. On the other hand, from some studies, a solution could be orchestrated, by picking up the developments made and putting them together in a single solution.

However, the aforementioned investigation had some limitations. Without having a clear and precise search query, when there is not a deep understanding of the thematics being researched, a significant number of studies was found. Many of these studies were deemed irrelevant after a deep analysis, making it difficult to retain the focus. Also, with the Quality Assessment policies, some interesting studies could be ignored and discarded.

The solution that was developed in this work consisted in, firstly, running a web crawler (the application used was the ACHE crawler) that downloaded only the relevant pages concerning COVID-19 (this was done using a URL-based filter with regular expressions). Then, a text processor application manages the downloaded pages, by guaranteeing they were not processed multiple times, extracting the metadata existent in the HTML pages and storing only the textual information. This information is stored in ElasticSearch, a search engine application that would allow the chatbot to retrieve the most relevant pages (already processed and cleansed) to a given question. At last, the chatbot mainly consisted of a fine-tuned BERT model with the SQuAD dataset (with more than 100000 question-answer pairs), which was one of the most cited models used to extract answers from context paragraphs. The BERT model was trained two times: one with the original version of SQuAD dataset and another with a translated version of it to Portuguese. Then, a UI built with the React framework was developed to allow the interaction between the users and the backend REST API built with FastAPI that made the connection to the trained BERT models.

The chosen crawler application, ACHE, proved to be exactly the tool we were looking for. It is extremely configurable, allowing us to define the wanted methodology for web page filtering and the re-crawling time intervals. Also, there is the option of downloading only pages from the domains defined in the URL start set. This configuration guarantees that only reliable information is downloaded.

To process the downloaded web pages, a Python application was developed using the 'html2text' library. Before extracting the HTML metadata, the pages had to be decoded from Base64, which was the encoding defined by the web crawler. Having the textual information extracted from the HTML documents, the text processor module identifies its language swiftly using the 'langdetect' library and then sets the correct index for that page.

The bridge between the processed data and the chatbot developed using ElasticSearch was extremely performant, taking less than a second to find the most relevant pages for a given query. ElasticSearch allows the creation of several indexes, with a customizable tokenizer and indexing method. In this work we used mostly the default configurations in the first development phase, which had to be later reviewed and reconfigured. For each index, the indexing date and the URL of the web page were stored, as well as the processed text.

The process of fine-tuning BERT was the first major block that was faced in this work. The main fact that led to this issue was the lack of hardware required to swiftly train the model. Since this work required the chatbot to work with both English and Portuguese, the pre-trained multilingual BERT model was chosen, as it had already been pre-trained in multiple languages with Wikipedia data, allowing the addition of other language support if required. However, this model has a significant dimension compared to the other versions of BERT, which led to huge amounts of fine-tuning. As there was no physical hardware available to train the models, this had to be done in a virtual machine in Google Cloud, which did not possess GPU. So, the processing was entirely done with the CPU. This translated into 7 days of fine-tuning for the English model and 4 days for the Portuguese model, each with 2 epochs only.

The second issue that delayed the development of the application was the lack of a correctly translated version to Portuguese of the SQuAD 2.0 dataset. As a solution, a translation of the dataset's version 1.1 was used, but it was not as complete as the latest version, leading to a loss of performance and accuracy of the trained chatbot.

Lately, another issue has surged. BERT was not able to process the downloaded pages as most of them exceeded the token limit defined by the model. Thus, a windowing approach had to be developed to allow the indexed documents being processed by BERT. However, this created a very significant overhead in its processing, increasing the response time and harming the application's usability.

Although the application built served the purpose of having a running chatbot that was fed by reliable information downloaded from web pages, the answers that were obtained from the chatbot were far from satisfactory. Some improvements were developed in order to obtain better results, like restricting the downloaded pages to only the domains declared as trustworthy and improving the indexing and searching algorithm of ElasticSearch by removing stop words and changing the tokenizer type, but still the results were far from the expected.

In conclusion, an application that provides answers to questions using information extracted from reliable sources was successfully built. On the other hand, these answers were either incomplete or wrong most of the time, misleading the users, which was the opposite purpose of this work. To overcome this situation, more developments had to be made, but these are not implemented in this work. Having made those improvements, the chatbot tool described and developed in this dissertation could be seamlessly used to reduce the question-answering congestion on Health Services.

7.2 Future Work

In order to improve the outcomes obtained with the developed chatbot, there is a set of changes that could be developed, each aimed to fix a given issue. These are discussed in this section.

When testing the application, one of the major drawbacks was the significant amount of time it took to answer a question. This is due to the windowing approach developed to allow BERT processing the web pages as its context paragraph. To avoid this approach, another, more performant, transformer model could be used. There are already transformer models that allow bigger text inputs as context paragraphs, such as Longformer (Beltagy et al., 2020) or CogLTX (Ding et al., 2020). However, these would probably require pre-training on multi-lingual textual sources, such as the used BERT model which was pre-trained with Wikipedia data from multiple languages.

If the processing time was not an issue and the focus was rather on getting correct answers, two implementations could be conducted to increase the model's performance. First, the BERT model could be better fine-tuned, with more epochs for both languages and with a translated version of SQuAD 2.0 for the Portuguese model. Also, a combination of other BERT-derivative models with better accuracy could be used, such as described in SQuAD's homepage leaderboard (*SQuAD 2.0*).

Disregarding a model change, the used BERT models could still be an option. However, since it is not being able to extract the answer correctly due to it being confused by FAQ pages for example, where the model is extracting the answer from the question itself, some refinements could be done on the text processor component of the application. The questions from FAQ pages could be removed, thus eliminating the cases where the chatbot is extracting the answer from the question. Another modification that could be introduced was text summarization. This procedure would both reduce the processing times of the BERT model (if the summarized texts did not exceed BERT's token limits) but also remove extra information that may be "confusing"the chatbot.

With the proposed implementation refinements, we believe the chatbot presented in this work could improve significantly not only the extracted answers but also the response times of the application, eliminating thus the drawbacks aforementioned.

Bibliography

Adamopoulou, E. and L. Moussiades (2020). "Chatbots: History, technology, and applications." *Machine Learning with Applications* 2, p. 100006. issn: 2666-8270.

Aggarwal, K. (2019). "An efficient focused web crawling approach." *Software Engineering*. Springer, pp. 131–138.

Ahmadi-Abkenari, F. and A. Selamat (2012). "An architecture for a focused trend parallel Web crawler with the application of clickstream analysis." *Information Sciences* 184(1), pp. 266–281.

Alderratia, N. and M. Elsheh (2019). "Using Web Pages Dynamicity to Prioritise Web Crawling." *Proceedings of the 2019 2nd International Conference on Machine Learning and Machine Intelligence*, pp. 40–44.

Almuhareb, A. (2016). "Arabic poetry focused crawling using SVM and keywords." *2016 4th Saudi International Conference on Information Technology (Big Data Analysis)(KACSTIT)*. IEEE, pp. 1–4.

Amalia, A., D. Gunawan, A. Najwan, and F. Meirina (2016). "Focused crawler for the acquisition of health articles." *2016 International Conference on Data and Software Engineering (ICoDSE)*. IEEE, pp. 1–6.

Amara, S., J. Macedo, F. Bendella, and A. Santos (2016). "Group formation in mobile computer supported collaborative learning contexts: A systematic literature review." *Journal of Educational Technology & Society* 19(2), pp. 258–273.

Amrin, A., C. Xia, and S. Dai (Jan. 2015). "Focused Web Crawling Algorithms." *Journal of Computers* 10, pp. 245–251.

Andrenucci, A. and E. Sneiders (2005). "Automated question answering: Review of the main approaches." *Third International Conference on Information Technology and Applications (ICITA'05)*. Vol. 1. IEEE, pp. 514–519.

Arasu, A., J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan (2001). "Searching the web." *ACM Transactions on Internet Technology (TOIT)* 1(1), pp. 2–43.

Badampudi, D., C. Wohlin, and K. Petersen (2015). "Experiences from using snowballing and database searches in systematic literature studies." *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, pp. 1–10.

Badawi, M., A. Mohamed, A. Hussein, and M. Gheith (2013). "Maintaining the search engine freshness using mobile agent." *Egyptian Informatics Journal* 14(1), pp. 27–36.

Balaji, S and S Sarumathi (2012). "TOPCRAWL: Community mining in web search engines with emphasize on topical crawling." *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*. IEEE, pp. 20–24.

Barbosa, A. P. M., J. Macedo, and O. Craveiro (2020). "Chatbot Development to Assist Patients In Health Care Services." *Relatório Técnico no Departamento de Informática da Universidade do Minho*.

Bavaresco, R., D. Silveira, E. Reis, J. Barbosa, R. Righi, C. Costa, R. Antunes, M. Gomes, C. Gatti, M. Vanzin, S. C. Junior, E. Silva, and C. Moreira (2020). "Conversational agents in business: A systematic literature review and future research directions." *Computer Science Review* 36, p. 100239. issn: 1574-0137.

Baykan, E., M. Henzinger, L. Marian, and I. Weber (2011). "A comprehensive study of features and algorithms for URL-based topic classification." *ACM Transactions on the Web (TWEB)* 5(3), pp. 1–29.

Bedi, P., A. Thukral, and H. Banati (2013). "Focused crawling of tagged web resources using ontology." *Computers & Electrical Engineering* 39(2), pp. 613–628.

Bedi, P., A. Thukral, H. Banati, A. Behl, and V. Mendiratta (2012). "A multi-threaded semantic focused crawler." *Journal of Computer Science and Technology* 27(6), pp. 1233–1242.

Beltagy, I., M. E. Peters, and A. Cohan (2020). "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150*.

Boldi, P., A. Marino, M. Santini, and S. Vigna (2018). "BUbiNG: Massive crawling for the masses." *ACM Transactions on the Web (TWEB)* 12(2), pp. 1–26.

Bonaccorsi, A. and C. Rossi (2003). "Why open source software can succeed." *Research policy* 32(7), pp. 1243–1258.

Brants, T., F. Chen, and I. Tsochantaridis (2002). "Topic-based document segmentation with probabilistic latent semantic analysis." *Proceedings of the eleventh international conference on Information and knowledge management*, pp. 211–218.

Brennen, J. S., F. Simon, P. N. Howard, and R. K. Nielsen (2020). "Types, sources, and claims of COVID-19 misinformation." *Reuters Institute* 7, pp. 3–1.

Building a QA system with Bert on Wikipedia. url: https://qa.fastforwardlabs.com/pytorch/hugging%20face/wikipedia/bert/transformers/2020/05/19/Getting_Started_with_QA.html#Using-a-pre-fine-tuned-model-from-the-Hugging-Face-repository (visited on 09/15/2021).

Carver, P. E. and J. Phillips (2020). "Novel Coronavirus (COVID-19): What You Need to Know." *Workplace Health & Safety* 68(5), pp. 250–250.

Chakrabarti, C. and G. F. Luger (2015). "Artificial conversations for customer service chatter bots: Architecture, algorithms, and evaluation metrics." *Expert Systems with Applications* 42(20), pp. 6878–6897. issn: 0957-4174.

Chandra, Y. W. and S. Suyanto (2019). "Indonesian Chatbot of University Admission Using a Question Answering System Based on Sequence-to-Sequence Model." *The 4th International Conference on Computer Science and Computational Intelligence (ICCSCI 2019) : Enabling Collaboration to Escalate Impact of Research Results for Society* 157, pp. 367–374. issn: 1877-0509.

Chen, D., F. Liying, Y. Jianzhuo, and B. Shi (2010). “Semantic focused crawler based on Q-learning and Bayes classifier.” *2010 3rd International Conference on Computer Science and Information Technology*. Vol. 8. IEEE, pp. 420–423.

Chen, H., X. Liu, D. Yin, and J. Tang (2017). “A Survey on Dialogue Systems: Recent Advances and New Frontiers.” *SIGKDD Explor. Newsl.* 19(2), pp. 25–35. issn: 1931-0145.

Chou, T.-L. and Y.-L. Hsueh (2019). “A Task-oriented Chatbot Based on LSTM and Reinforcement Learning - NLPIR 2019: 2019 3RD INTERNATIONAL CONFERENCE ON NATURAL LANGUAGE PROCESSING AND INFORMATION RETRIEVAL.” 3rd International Conference on Natural Language Processing and Information Retrieval (NLPIR), Tokushima Univ, Tokushima, JAPAN, JUN 28-30, 2019, 87–91.

Chowanda, A. and A. D. Chowanda (2018). “Generative Indonesian Conversation Model using Recurrent Neural Network with Attention Mechanism.” *The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life* 135, pp. 433–440. issn: 1877-0509.

Cinelli, M., W. Quattrociocchi, A. Galeazzi, C. M. Valensise, E. Brugnoli, A. L. Schmidt, P. Zola, F. Zollo, and A. Scala (2020). “The covid-19 social media infodemic.” *arXiv preprint arXiv:2003.05004*.

Coronavirus disease (COVID-19) FAQs. url: `https://www.un.org/en/coronavirus/covid-19-faqs` (visited on 10/10/2021).

Dahiwale, P., M. Raghuwanshi, and L. Malik (2014). “Design of improved focused web crawler by analyzing semantic nature of URL and anchor text.” *2014 9th International Conference on Industrial and Information Systems (ICIIS)*. IEEE, pp. 1–6.

Dale, R. (2016). “The return of the chatbots.” *Natural Language Engineering* 22(5), pp. 811–817.

Day, M.-Y., J.-T. Lin, and Y.-C. Chen (2018). “Artificial Intelligence for Conversational Robo-Advisor.” ASONAM ’18, pp. 1057–1064. issn: 978-1-5386-6051-5.

Deng, S. (2020). “Research on the Focused Crawler of Mineral Intelligence Service Based on Semantic Similarity.” *Journal of Physics: Conference Series*. Vol. 1575. 1. IOP Publishing, p. 012142.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018). “Bert: Pre-training of deep bidirectional transformers for language understanding.” *arXiv preprint arXiv:1810.04805*.

Ding, M., C. Zhou, H. Yang, and J. Tang (2020). “Cogltx: Applying bert to long texts.” *Advances in Neural Information Processing Systems* 33, pp. 12792–12804.

Dong, H., F. K. Hussain, and E. Chang (2008). “A survey in traditional information retrieval models.” *2008 2nd IEEE International Conference on Digital Ecosystems and Technologies*. IEEE, pp. 397–402.

Du, Y., W. Liu, X. Lv, and G. Peng (2015). “An improved focused crawler based on semantic similarity vector space model.” *Applied Soft Computing* 36, pp. 392–407.

Du, Y., Q. Pen, and Z. Gao (2013). “A topic-specific crawling strategy based on semantics similarity.” *Data & Knowledge Engineering* 88, pp. 75–93.

Feng, S., L. Zhang, Y. Xiong, and C. Yao (2010). “Focused crawling using navigational rank.” *Proceedings of the 19th ACM international conference on Information and knowledge management*, pp. 1513–1516.

Flejter, D., K. Wieloch, and W. Abramowicz (2007). “Unsupervised methods of topical text segmentation for polish.” *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing*, pp. 51–58.

Ganguly, B. and D. Raich (2014). “Performance optimization of focused web crawling using content block segmentation.” *2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies*. IEEE, pp. 365–370.

Gao, C. and J. Ren (2019). “A topic-driven language model for learning to generate diverse sentences.” *Neurocomputing* 333, pp. 374–380. issn: 0925-2312.

Gaur, R. and D. K. Sharma (2014). “Review of ontology based focused crawling approaches.” *2014 International Conference of Soft Computing Techniques for Engineering and Technology (ICSCTET)*. IEEE, pp. 1–4.

Goyal, N., R. Bhatia, and M. Kumar (2016). “A genetic algorithm based focused Web crawler for automatic webpage classification.” *3rd International Conference on Electrical, Electronics, Engineering Trends, Communication, Optimization and Sciences (EEECOS 2016)*. IET, pp. 1–6.

Greengrass, E. (2000). “Information retrieval: A survey.”

Gupta, A. and P. Anand (2015). “Focused web crawlers and its approaches.” *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. IEEE, pp. 619–622.

Gupta, S. (2016). “Design of focused crawler based on feature extraction, classification and term extraction.” *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, pp. 3430–3434.

Han, M., P.-H. Wuillemin, and P. Senellart (2018). “Focused crawling through reinforcement learning.” *International Conference on Web Engineering*. Springer, pp. 261–278.

Hao, H.-W., C.-X. Mu, X.-C. Yin, S. Li, and Z.-B. Wang (2011). “An improved topic relevance algorithm for focused crawling.” *2011 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, pp. 850–855.

Hassan, T., C. Cruz, and A. Bertaux (2017). “Ontology-based approach for unsupervised and adaptive focused crawling.” *Proceedings of The International Workshop on Semantic Big Data*, pp. 1–6.

Hernández, I., C. R. Rivero, D. Ruiz, and R. Corchuelo (2016). “CALA: ClAssifying Links Automatically based on their URL.” *Journal of Systems and Software* 115, pp. 130–143.

Horev, R. (2018). *BERT Explained: State of the art language model for NLP*. url: `https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270` (visited on 10/10/2021).

Huang, X., F. Peng, D. Schuurmans, N. Cercone, and S. E. Robertson (2003). “Applying machine learning to text segmentation for information retrieval.” *Information Retrieval* 6(3-4), pp. 333–362.

Hussain, S., O. Ameri Sianaki, and N. Ababneh (2019). “A Survey on Conversational Agents/Chatbots Classification and Design Techniques.” *Advances in Intelligent Systems and Computing* 927. Cited By :16 Export Date: 21 March 2021, pp. 946–956.

Jaganathan, P and T Karthikeyan (2015). “Highly Efficient Architecture for Scalable Focused Crawling Using Incremental Parallel Web Crawler.” *Journal of Computer Science* 11(1), p. 120.

Jalali, S. and C. Wohlin (2012). “Systematic literature studies: database searches vs. backward snowballing.” *Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement*. IEEE, pp. 29–38.

Jian, F., C. Jing-zhou, and C. Lei (2014). “Design and Implementation of A Focused Crawler—TargetCrawler.” *International Journal of Grid and Distributed Computing* 7(4), pp. 149–156.

Joe Dhanith, P. and B Surendiran (2019). “An ontology learning based approach for focused web crawling using combined normalized pointwise mutual information and Resnik algorithm.” *International Journal of Computers and Applications*, pp. 1–7.

Keele, S. et al. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Tech. rep. Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

Khalil, S. and M. Fakir (2017). “RCrawler: An R package for parallel web crawling and scraping.” *SoftwareX* 6, pp. 98–106.

Koster, M. (1994). *A standard for robot exclusion*. NEXOR.

Kulkarni, T., M. Kabra, and R. Shankarmani (2019). “User Profiling Based Recommendation System for E-Learning.” *2019 IEEE 16th India Council International Conference (INDICON)*. IEEE, pp. 1–4.

Kumar, M., R. Bhatia, and D. Rattan (2017). “A survey of Web crawlers for information retrieval.” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7(6), e1218.

Kumar, M., A. Bindal, R. Gautam, and R. Bhatia (2018). “Keyword query based focused Web crawler.” *Procedia Computer Science* 125, pp. 584–590.

Kumar, M. and R. Vig (2013). “Online Library Content Generation Using Focused Crawling Based Upon Meta Tags and Tf-Idf.” *2013 International Symposium on Computational and Business Intelligence*. IEEE, pp. 158–161.

LI, C.-x., Y.-r. SU, R.-j. WANG, Y.-y. WEI, and H. Huang (2012). “Structured AJAX data extraction based on agricultural ontology.” *Journal of Integrative Agriculture* 11(5), pp. 784–791.

Li, L., C. Li, and D. Ji (2021). “Deep context modeling for multi-turn response selection in dialogue systems.” *Information Processing Management* 58(1), p. 102415. issn: 0306-4573.

Li, X., M. Xing, and J. Zhang (2011). “A comprehensive prediction method of visit priority for focused crawler.” *2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing*. IEEE, pp. 27–30.

Liu, H. and E. Milios (2012). “Probabilistic models for focused web crawling.” *Computational Intelligence* 28(3), pp. 289–328.

Liu, L. and T. Peng (2014). “Clustering-based topical Web crawling using CFu-tree guided by link-context.” *Frontiers of Computer Science* 8(4), pp. 581–595.

Liu, W. and Y. Du (2014). “A novel focused crawler based on cell-like membrane computing optimization algorithm.” *Neurocomputing* 123, pp. 266–280.

Lotfi, M., M. R. Hamblin, and N. Rezaei (2020). “COVID-19: Transmission, prevention, and potential therapeutic opportunities.” *Clinica chimica acta*.

Lu, H., D. Zhan, L. Zhou, and D. He (2016). “An improved focused crawler: using web page classification and link priority evaluation.” *Mathematical Problems in Engineering* 2016.

Lucene’s Practical Scoring Function. url: https://www.elastic.co/guide/en/elasticsearch/guide/master/practical-scoring-function.html (visited on 11/18/2021).

Luo, J., Y. L. Lu, and C. X. Lin (2014). “Research on Content Analysis algorithm of Focused Crawler based on LBTF-IDF.” *Advanced Materials Research*. Vol. 971. Trans Tech Publ, pp. 1722–1725.

Lutkevich, B. (2020). *What is Bert (language model) and how does it work?* url: https://searchenterpriseai.techtarget.com/definition/BERT-language-model (visited on 10/10/2021).

Mali, S, S Ninoriya, and B. Meshram (2011). “Freshness tuning in focused crawler.” *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pp. 548–552.

Mali, S. and B. Meshram (2011). “Focused web crawler with revisit policy.” *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pp. 474–479.

Mallawaarachchi, V., L. Meegahapola, R. Madhushanka, E. Heshan, D. Meedeniya, and S. Jayarathna (2020). “Change Detection and Notification of Web Pages: A Survey.” *ACM Computing Surveys (CSUR)* 53(1), pp. 1–35.

Miner, G., J. Elder IV, A. Fast, T. Hill, R. Nisbet, and D. Delen (2012). *Practical text mining and statistical analysis for non-structured text data applications*. Chap. Text Classification and Categorization, pp. 881–892.

Minhas, G. and M. Kumar (2013). “LSI based relevance computation for topical web crawler.” *Journal of Emerging Technologies in Web Intelligence* 5(4), pp. 401–406.

Mironczuk, M. M. and J. Protasiewicz (2018). “A recent overview of the state-of-the-art elements of text classification.” *Expert Systems with Applications* 106, pp. 36–54.

Misra, H., F. Yvon, O. Cappé, and J. Jose (2011). “Text segmentation: A topic modeling perspective.” *Information Processing & Management* 47(4), pp. 528–544.

Mor, J., N. Kumar, and D. Rai (2019). “An Improved Crawler Based on Efficient Ranking Algorithm.” *International Journal of Advanced Trends in Computer Science and Engineering* 8(2), pp. 119–125.

Naeem, S. B. and R. Bhatti (2020). “The Covid-19 ‘infodemic’: a new front for information professionals.” *Health Information & Libraries Journal*.

Naghibi, M. and A. T. Rahmani (2012). “Focused crawling using vision-based page segmentation.” *International Conference on Information Systems, Technology and Management*. Springer, pp. 1–12.

Nguyen, T. and M. Shcherbakov (2018). “A Neural Network based Vietnamese Chatbot.” *2018 International Conference on System Modeling Advancement in Research Trends (SMART)*, pp. 147–149. issn: VO -.

Nimavat, K. and T. Champaneria (2017). “Chatbots: an overview types, architecture, tools and future possibilities.” *Int. J. Sci. Res. Dev* 5(7), pp. 1019–1024.

Ning, H., H. Wu, Z. He, and Y. Tan (2011). “Focused crawler URL analysis model based on improved genetic algorithm.” *2011 IEEE International Conference on Mechatronics and Automation*. IEEE, pp. 2159–2164.

Nithuna, S and C. Laseena (2020). “Review on Implementation Techniques of Chatbot.” *2020 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, pp. 0157–0161.

Nuruzzaman, M. and O. K. Hussain (2018). “A survey on chatbot implementation in customer service industry through deep neural networks.” *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*. IEEE, pp. 54–61.

Nuruzzaman, M. and O. K. Hussain (2019). “Identifying Facts for Chatbot’s Question Answering via Sequence Labelling Using Recurrent Neural Networks.” ACM TURC ’19. issn: 978-1-4503-7158-2.

Nuruzzaman, M. and O. K. Hussain (2020). “IntelliBot: A Dialogue-based chatbot for the insurance industry.” *Knowledge-Based Systems* 196, p. 105810. issn: 0950-7051.

Odhiambo, J. and J. Okungu (Oct. 2020). “Top 15 Covid-19 Research Topics and Areas.” preprint on webpage at `https://www.researchgate.net/publication/343380794_Top_15_Covid-19_Research_Topics_and_Areas`.

Onan, A. (2016). “Classifier and feature set ensembles for web page classification.” *Journal of Information Science* 42(2), pp. 150–165.

Osman, D. J. and J. L. Yearwood (2007). “Opinion search in web logs.” *Proceedings of the eighteenth conference on Australasian database-Volume 63*, pp. 133–139.

Özel, S. A. (2011). “A web page classification system based on a genetic algorithm using tagged-terms as features.” *Expert Systems with Applications* 38(4), pp. 3407–3415.

Pak, I. and P. L. Teh (2018). “Text segmentation techniques: a critical review.” *Innovative Computing, Optimization and Its Applications*. Springer, pp. 167–181.

Parvez, M. S., K. S. A. Tasneem, S. S. Rajendra, and K. R. Bodke (2018). “Analysis Of Different Web Data Extraction Techniques.” *2018 International Conference on Smart City and Emerging Technology (ICSCET)*, pp. 1–7.

Pasari, R., V. Chaudhari, A. Borkar, and A. Joshi (2016). “Parallelization of vertical search engine using Hadoop and MapReduce.” *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, pp. 1–5.

Patel, A. and N. Schmidt (2011). “Application of structured document parsing to focused web crawling.” *Computer Standards & Interfaces* 33(3), pp. 325–331.

Pawar, N., K Rajeswari, and A. Joshi (2016). “Implementation of an efficient web crawler to search medicinal plants and relevant diseases.” *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*. IEEE, pp. 1–4.

Peng, T. and L. Liu (2013). “Focused crawling enhanced by CBP–SLC.” *Knowledge-Based Systems* 51, pp. 15–26.

Pham, K., A. Santos, and J. Freire (2018). “Learning to Discover Domain-Specific Web Content.” *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 432–440.

Pirkola, A. and T. Talvensaari (2010). “A topic-specific Web search system focusing on quality pages.” *International Conference on Theory and Practice of Digital Libraries*. Springer, pp. 490–493.

Prassanna, J., K. Khadar Nawas, J. Christy Jackson, R. Prabakaran, and S. Ramanathan (2020). “Towards building a neural conversation chatbot through seq2seq model.” *International Journal of Scientific and Technology Research* 9(3). Export Date: 21 March 2021, pp. 1219–1222.

Qiu, J., Q. Du, W. Wang, K. Yin, C. Lin, and C. Qian (2017). “Topic Crawler for OpenStack QA Knowledge Base.” *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, pp. 309–317.

Rajalakshmi, R and C. Aravindan (2013). “Web page classification using n-gram based URL features.” *2013 fifth international conference on advanced computing (ICoAC)*. IEEE, pp. 15–21.

Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (Nov. 2016). “SQuAD: 100,000+ Questions for Machine Comprehension of Text.” *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics: Austin, Texas, pp. 2383–2392.

Rheinländer, A., M. Lehmann, A. Kunkel, J. Meier, and U. Leser (2016). “Potential and pitfalls of domain-specific information extraction at web scale.” *Proceedings of the 2016 international conference on management of data*, pp. 759–771.

Richardson, L. (2017). “Beautiful Soup Documentation.[online].” url: `https://www.crummy.com/software/BeautifulSoup/bs4/doc` (visited on 07/05/2021).

Rochwerg, B., R. Parke, S. Murthy, S. M. Fernando, J. P. Leigh, J. Marshall, N. K. Adhikari, K. Fiest, R. Fowler, F. Lamontagne, et al. (2020). “Misinformation during the coronavirus disease 2019 outbreak: How knowledge emerges from noise.” *Critical Care Explorations* 2(4).

Saini, C. and V. Arora (2016). “Information retrieval in web crawling: A survey.” *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, pp. 2635–2643.

Saleh, A. I., A. E. Abulwafa, and M. F. Al Rahmawy (2017). “A web page distillation strategy for efficient focused crawling based on optimized Naïve bayes (ONB) classifier.” *Applied Soft Computing* 53, pp. 181–204.

Santos, A., B. Pasini, and J. Freire (2016). “A first study on temporal dynamics of topics on the web.” *Proceedings of the 25th International Conference Companion on World Wide Web*, pp. 849–854.

Scacchi, W. (2007). “Free/open source software development.” *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 459–468.

Schwartz, B. (2020). *Google: BERT now used on almost every English query*. url: `https://searchengineland.com/google-bert-used-on-almost-every-english-query-342193` (visited on 10/10/2021).

Selamat, A. and F. Ahmadi-Abkenari (2011). “Architecture for a parallel focused crawler for clickstream analysis.” *Asian Conference on Intelligent Information and Database Systems*. Springer, pp. 27–35.

Seyfi, A. and A. Patel (2016). “A focused crawler combinatory link and content model based on T-Graph principles.” *Computer Standards & Interfaces* 43, pp. 1–11.

Seyfi, A., A. Patel, and J. C. Júnior (2016). "Empirical evaluation of the link and content-based focused Treasure-Crawler." *Computer Standards & Interfaces* 44, pp. 54–62.

Shahi, G. K. and D. Nandini (2020). "FakeCovid–A Multilingual Cross-domain Fact Check News Dataset for COVID-19." *arXiv preprint arXiv:2006.11343*.

Shang, S., H. Wu, and J. Ma (2019). "An Improved Focused Web Crawler based on Hybrid Similarity." *International Journal of Performability Engineering*(10).

Shrivastava, G. K., P. Kaushik, and R. K. Pateriya (2019). "Comprehensive Analysis of Web Page Classifier for Fsocused Crawler." *International Journal of Innovative Technology and Exploring Engineering Regular Issue* 8(9), pp. 57–65.

Singh, N., H. Sandhawalia, N. Monet, H. Poirier, and J.-M. Coursimault (2012). "Large scale url-based classification using online incremental learning." *2012 11th International Conference on Machine Learning and Applications*. Vol. 2. IEEE, pp. 402–409.

Sleiman, H. A. and R. Corchuelo (2012). "A survey on region extractors from web documents." *IEEE Transactions on Knowledge and Data Engineering* 25(9), pp. 1960–1981.

SQuAD 2.0. url: `https://rajpurkar.github.io/SQuAD-explorer/` (visited on 12/03/2021).

Stop token filter: Elasticsearch Guide [7.15]. url: `https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-stop-tokenfilter.html#analysis-stop-tokenfilter-stop-words-by-lang` (visited on 11/18/2021).

Suebchua, T., B. Manaskasemsak, A. Rungsawang, and H. Yamana (2018). "Efficient topical focused crawling through neighborhood feature." *New Generation Computing* 36(2), pp. 95–118.

Szpektor, I., D. Cohen, G. Elidan, M. Fink, A. Hassidim, O. Keller, S. Kulkarni, E. Ofek, S. Pudinsky, A. Revach, S. Salant, and Y. Matias (2020). "Dynamic Composition for Conversational Domain Exploration." WWW '20, pp. 872–883. issn: 978-1-4503-7023-3.

Tan, Q. and P. Mitra (2010). "Clustering-based incremental web crawling." *ACM Transactions on Information Systems (TOIS)* 28(4), pp. 1–27.

Taylan, D., M. Poyraz, S. Akyokuş, and M. C. Ganiz (2011). "Intelligent focused crawler: Learning which links to crawl." *2011 International Symposium on Innovations in Intelligent Systems and Applications*. IEEE, pp. 504–508.

Torkestani, J. A. (2012). "An adaptive focused web crawling algorithm based on learning automata." *Applied Intelligence* 37(4), pp. 586–601.

Uemura, Y., T. Itokawa, T. Kitasuka, and M. Aritsugi (2012). "An effectively focused crawling system." *Innovations in Intelligent Machines–2*. Springer, pp. 61–76.

Vamsi, G. K., A. Rasool, and G. Hajela (2020). "Chatbot: A Deep Neural Network Based Human to Machine Conversation Model." *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–7. issn: VO -.

Velavan, T. P. and C. G. Meyer (2020). "The COVID-19 epidemic." *Tropical medicine & international health* 25(3), p. 278.

Venu, S. H., V. Mohan, K. Urkalan, and T. Geetha (2016). "Unsupervised domain ontology learning from text." *International Conference on Mining Intelligence and Knowledge Exploration*. Springer, pp. 132–143.

Wai, H. P. M., P. P. Tar, and P. Thwe (2018). "Ontology Based Web Page Classification System by Using Enhanced C4. 5 and Naïve Bayesian Classifiers." *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. Vol. 3. IEEE, pp. 286–291.

Wang, W., X. Chen, Y. Zou, H. Wang, and Z. Dai (2010). "A focused crawler based on naive bayes classifier." *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*. IEEE, pp. 517–521.

Wang, Y., Z. Hong, and M. Shi (2018). "Research on lda model algorithm of news-oriented web crawler." *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*. IEEE, pp. 748–753.

Waszak, P. M., W. Kasprzycka-Waszak, and A. Kubanek (2018). "The spread of medical fake news in social media–the pilot quantitative study." *Health policy and technology* 7(2), pp. 115–118.

WHO Coronavirus (COVID-19) Dashboard. url: `https://covid19.who.int/` (visited on 03/01/2020).

Wikipedia (2021). *F-score — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=F-score&oldid=1030531397`. (Visited on 10/24/2021).

Wohlin, C. (2014). "Guidelines for snowballing in systematic literature studies and a replication in software engineering." *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pp. 1–10.

Xie, D. X. and W. F. Xia (2014). "Design and implementation of the topic-focused crawler based on scrapy." *advanced materials research*. Vol. 850. Trans Tech Publ, pp. 487–490.

Xu, G., P. Jiang, C. Ma, M. Daneshmand, and S. Xie (2019). "VRPSOFC: a framework for focused crawler using mutation improving particle swarm optimization algorithm." *Proceedings of the ACM Turing Celebration Conference-China*, pp. 1–7.

Yan, W. and L. Pan (2018). "Designing focused crawler based on improved genetic algorithm." *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, pp. 319–323.

Yang, S.-Y. (2010). "OntoCrawler: A focused crawler with ontology-supported website models for information agents." *Expert Systems with Applications* 37(7), pp. 5381–5389.

Yang, S.-Y. and C.-L. Hsu (2010). "An ontology-supported web focused-crawler for Java programs." *2010 3rd IEEE International Conference on Ubi-Media Computing*. IEEE, pp. 266–271.

Yin, Z., K.-H. Chang, and R. Zhang (2017). "DeepProbe: Information directed sequence understanding and chatbot design via recurrent neural networks." *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Cited By :15 Export Date: 21 March 2021, pp. 2131–2139.

Yu, Y.-B., S.-L. Huang, N. Tashi, H. Zhang, F. Lei, and L.-Y. Wu (2018). "A survey about algorithms utilized by focused web crawler." *Journal of Electronic Science and Technology* 16(2), pp. 129–138.

Yuki, K., M. Fujiogi, and S. Koutsogiannaki (2020). “COVID-19 pathophysiology: A review.” *Clinical Immunology* 215, p. 108427. issn: 1521-6616.

Zhang, H. and J. Lu (2010). “SCTWC: An online semi-supervised clustering approach to topical web crawlers.” *Applied Soft Computing* 10(2), pp. 490–495.

Zhang, R., J. Guo, Y. Fan, Y. Lan, and X. Cheng (2020). “Dual-Factor Generation Model for Conversation.” *ACM Trans. Inf. Syst.* 38(3). issn: 1046-8188.

Zhang, Y., T. Xu, and Y. Dai (2019). “Research on Chatbots for Open Domain: Using BiLSTM and Sequence to Sequence.” AICS 2019, pp. 145–149. issn: 978-1-4503-7150-6.

Zheng, S. (2011). “Genetic and ant algorithms based focused crawler design.” *2011 Second International Conference on Innovations in Bio-inspired Computing and Applications*. IEEE, pp. 374–378.

Zheng, Z. and D. Qian (2016). “An improved focused crawler based on text keyword extraction.” *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*. IEEE, pp. 386–390.

Appendix A

Systematic Literature Review: Quality Assessment

The following tables (ordered by *Score* and then alphabetically by *Study*) represent the Quality Assessment results performed on the selected studies within the SLR conducted.

Table A.1: Quality Assessment: Web Crawling

Study	Q1	Q2	Q3	Q4	Q5	Score
Du et al. (2015)	2	1	1	1	1	6
Hernández et al. (2016)	2	1	1	1	1	6
Mallawaarachchi et al. (2020)	2	1	1	1	1	6
Peng and Liu (2013)	2	1	1	1	1	6
Pham et al. (2018)	2	1	1	1	1	6
Saleh et al. (2017)	2	1	1	1	1	6
Santos et al. (2016)	2	1	1	1	1	6
Tan and Mitra (2010)	2	1	1	1	1	6
Ahmadi-Abkenari and Selamat (2012)	2	1	0.5	1	1	5.5
Baykan et al. (2011)	1.5	1	1	1	1	5.5
Feng et al. (2010)	1.5	1	1	1	1	5.5
Kumar et al. (2017)	1.5	1	1	1	1	5.5
Liu and Du (2014)	1.5	1	1	1	1	5.5
Seyfi et al. (2016)	1.5	1	1	1	1	5.5
Seyfi and Patel (2016)	1.5	1	1	1	1	5.5
Sleiman and Corchuelo (2012)	1.5	1	1	1	1	5.5
Torkestani (2012)	1.5	1	1	1	1	5.5
Yang (2010)	1.5	1	1	1	1	5.5

Table A.1 continued from previous page

Study	Q1	Q2	Q3	Q4	Q5	Score
Bedi et al. (2013)	1	1	1	1	1	5
Boldi et al. (2018)	1	1	1	1	1	5
Du et al. (2013)	1.5	1	1	0.5	1	5
Hao et al. (2011)	1	1	1	1	1	5
Lu et al. (2016)	1	1	1	1	1	5
Ning et al. (2011)	1	1	1	1	1	5
Onan (2016)	1	1	1	1	1	5
Patel and Schmidt (2011)	1.5	1	0.5	1	1	5
Zhang and Lu (2010)	1.5	1	1	0.5	1	5
Bedi et al. (2012)	0.5	1	1	1	1	4.5
Dahiwale et al. (2014)	0.5	1	1	1	1	4.5
Han et al. (2018)	1	1	0.5	1	1	4.5
LI et al. (2012)	0.5	1	1	1	1	4.5
Liu and Milios (2012)	1	1	0.5	1	1	4.5
Liu and Peng (2014)	0.5	1	1	1	1	4.5
Singh et al. (2012)	0.5	1	1	1	1	4.5
Suebchua et al. (2018)	0.5	1	1	1	1	4.5
Taylan et al. (2011)	0.5	1	1	1	1	4.5
Wang et al. (2018)	0.5	1	1	1	1	4.5
Almuhareb (2016)	0	1	1	1	1	4
Amalia et al. (2016)	0	1	1	1	1	4
Amrin et al. (2015)	0	1	1	1	1	4
Balaji and Sarumathi (2012)	0	1	1	1	1	4
Ganguly and Raich (2014)	0	1	1	1	1	4
Gaur and Sharma (2014)	0	1	1	1	1	4
Gupta and Anand (2015)	0	1	1	1	1	4
Hassan et al. (2017)	0	1	1	1	1	4
Khalil and Fakir (2017)	0	1	1	1	1	4
Li et al. (2011)	0	1	1	1	1	4
Mali et al. (2011)	0	1	1	1	1	4
Mor et al. (2019)	0	1	1	1	1	4
Pawar et al. (2016)	0	1	1	1	1	4
Rajalakshmi and Aravindan (2013)	0	1	1	1	1	4
Rheinländer et al. (2016)	0	1	1	1	1	4
Venu et al. (2016)	0	1	1	1	1	4

Table A.1 continued from previous page

Study	Q1	Q2	Q3	Q4	Q5	Score
Wai et al. (2018)	0	1	1	1	1	4
Wang et al. (2010)	0	1	1	1	1	4
Xu et al. (2019)	0	1	1	1	1	4
Yan and Pan (2018)	0	1	1	1	1	4
Yang and Hsu (2010)	0	1	1	1	1	4
Yu et al. (2018)	0	1	1	1	1	4
Alderratia and Elsheh (2019)	1	1	1	0.5	0	3.5
Mali and Meshram (2011)	0	1	1	0.5	1	3.5
Naghibi and Rahmani (2012)	0	1	0.5	1	1	3.5
Özel (2011)	0	1	0.5	1	1	3.5
Pasari et al. (2016)	0	1	1	1	0.5	3.5
Selamat and Ahmadi-Abkenari (2011)	0.5	1	0.5	0.5	1	3.5
Zheng (2011)	0	1	0.5	1	1	3.5
Zheng and Qian (2016)	0	1	1	0.5	1	3.5
Chen et al. (2010)	0	1	0.5	0.5	1	3
Deng (2020)	0	1	1	1	0	3
Goyal et al. (2016)	0	1	1	1	0	3
Gupta (2016)	0	0.5	1	0.5	1	3
Jaganathan and Karthikeyan (2015)	0	1	0.5	0.5	1	3
Joe Dhanith and Surendiran (2019)	0	1	1	1	0	3
Kumar et al. (2018)	0	1	0.5	0.5	1	3
Shrivastava et al. (2019)	0	1	1	1	0	3
Xie and Xia (2014)	0	0.5	1	0.5	1	3
Jian et al. (2014)	0	1	1	0.5	0	2.5
Minhas and Kumar (2013)	0	1	0	0.5	1	2.5
Pirkola and Talvensaari (2010)	1	0.5	0	0	1	2.5
Qiu et al. (2017)	0	1	1	0.5	0	2.5
Shang et al. (2019)	0	1	1	0.5	0	2.5
Uemura et al. (2012)	0	0.5	1	0	1	2.5
Aggarwal (2019)	0	0.5	0.5	0	1	2
Kulkarni et al. (2019)	0	0.5	1	0.5	0	2
Kumar and Vig (2013)	0	1	0	0.5	0	1.5
Luo et al. (2014)	0	0.5	0.5	0.5	0	1.5

Table A.2: Quality Assessment: Chatbot Development

Study	Q1	Q2	Q3	Q4	Q5	Score
Bavaresco et al. (2020)	2	1	1	1	1	6
Chakrabarti and Luger (2015)	2	1	1	1	1	6
Gao and Ren (2019)	2	1	1	1	1	6
Li et al. (2021)	2	1	1	1	1	6
Nuruzzaman and Hussain (2020)	2	1	1	1	1	6
Yin et al. (2017)	2	1	1	1	1	6
Szpektor et al. (2020)	2	1	1	0.5	1	5.5
Nuruzzaman and Hussain (2018)	1	1	1	1	1	5
Zhang et al. (2020)	1.5	1	1	1	0	4.5
Adamopoulou and Moussiades (2020)	0	1	1	1	1	4
Chen et al. (2017)	0	1	1	1	1	4
Hussain et al. (2019)	0	1	1	1	1	4
Nguyen and Shcherbakov (2018)	0	1	1	1	1	4
Chou and Hsueh (2019)	0	1	0.5	1	1	3.5
Chowanda and Chowanda (2018)	0	1	1	0.5	1	3.5
Day et al. (2018)	0	1	0.5	1	1	3.5
Nuruzzaman and Hussain (2019)	0	1	1	0.5	1	3.5
Chandra and Suyanto (2019)	0	1	0.5	0.5	1	3
Vamsi et al. (2020)	0	1	1	1	0	3
Prassanna et al. (2020)	0	1	1	0	0	2
Zhang et al. (2019)	0	0	0.5	0	1	1.5

Appendix B

Web Crawler: ACHE Configuration File

The listings in this Appendix are the representation of the configurations used to setup the ACHE Web Crawler, defined in the *ache.yml* file.

B.1 Target Storage

Listing B.1: ACHE Target Storage Configurations

```
# Change to false if you don't want to store pages classified as irrelevant
target_storage.store_negative_pages: false

# Configuration for data format used to store crawled data
target_storage.data_format.type: FILESYSTEM_JSON
target_storage.data_format.filesystem.hash_file_name: true
target_storage.data_format.filesystem.compress_data: false

# Performs hard focus or soft focus. When hard focus is enabled,
# the crawler only follows links from pages classified as relevant
target_storage.hard_focus: false

# Run bipartite crawler
target_storage.bipartite: false

# Maximum number of pages to visit
target_storage.visited_page_limit: 1000000000

# Store only pages that contain english text using language detector
target_storage.english_language_detection_enabled: false
```

B.2 Link Storage

Listing B.2: ACHE Link Storage Configurations

```
# Max number of pages to be crawled from each web domain
link_storage.max_pages_per_domain: 1000000000
# Restricts the crawler to crawl the websites provided as seeds
link_storage.link_strategy.use_scope: false
# Allows the crawler to follow forward links
link_storage.link_strategy.outlinks: true
# Gets backlinks of the pages from a search engine used by the bipartite crawling
link_storage.link_strategy.backlinks: false

# Type of link classifier used by link storage
# - LinkClassifierBaseline: random link strategy when no page classifier is provided,
# or Soumen's baseline strategy when a page classifier is provided
# - LinkClassifierImpl: link strategy using a link classifier
# - LinkClassifierAuthority: link strategy for the bipartite crawling
link_storage.link_classifier.type: LinkClassifierBaseline

# Retrain link classifiers on-the-fly
link_storage.online_learning.enabled: false

link_storage.link_selector: TopkLinkSelector

# Enable recrawling of sitemaps using at fixed time intervals (in minutes)
link_storage.recrawl_selector: SitemapsRecrawlSelector
link_storage.recrawl_selector.sitemaps.interval: 360 # 6 hours

# Discovery of new links using sitemap.xml protocol
link_storage.download_sitemap_xml: true

link_storage.max_size_cache_urls: 10000

# Directory to store link storage's frontier database
link_storage.directory: ”data_url/dir”

link_storage.scheduler.host_min_access_interval: 5000
link_storage.scheduler.max_links: 10000
```

B.3 Crawler Manager

Listing B.3: ACHE Crawler Manager Configurations

```
crawler_manager.downloader.user_agent.name: ACHE
crawler_manager.downloader.user_agent.url: https://github.com/ViDA-NYU/ache
crawler_manager.downloader.user_agent.email: a80261@alunos.uminho.pt

crawler_manager.downloader.download_thread_pool_size: 100
crawler_manager.downloader.max_retry_count: 3
crawler_manager.downloader.valid_mime_types:
  - text/xml
  - text/html
  - text/plain
  - application/x-asp
  - application/xhtml+xml
  - application/vnd.wap.xhtml+xml

# Use OkHttpFetcher instead of SimpleHttpFetcher
crawler_manager.downloader.use_okhttp3_fetcher: true

# okhttp3 proxy Configuration
crawler_manager.downloader.okhttp3.proxy_host: null
crawler_manager.downloader.okhttp3.proxy_username: null
crawler_manager.downloader.okhttp3.proxy_password: null
crawler_manager.downloader.okhttp3.proxy_port: 8080
```

Appendix C
Code Listings

This Appendix contains listings of the code used to build the application.

C.1 Chunkify, Tokenize and Get Answer with BERT model

Listing C.1: Chunkify, Tokenize and Get Answer with BERT model

```python
import torch
from collections import OrderedDict

def tokenize(tokenizer, model, question, text):
   inputs = tokenizer.encode_plus(question, text, add_special_tokens=True, return_tensors="pt"
       ↪ )
   input_ids = inputs["input_ids"].tolist()[0]

   chunked = False

   if len(input_ids) > model.config.max_position_embeddings:
      inputs = chunkify(inputs)
      chunked = True

   return chunked, inputs

def chunkify(inputs):
   # create question mask based on token_type_ids
   # value is 0 for question tokens, 1 for context tokens
   qmask = inputs['token_type_ids'].lt(1)
   qt = torch.masked_select(inputs['input_ids'], qmask)
   chunk_size = model.config.max_position_embeddings - qt.size()[0] - 1
```

```
    # the ”-1” accounts for having to add an ending [SEP] token to the end

    # create a dict of dicts; each sub-dict mimics the structure of pre-chunked model input
    chunked_input = OrderedDict()
    for k,v in inputs.items():
        q = torch.masked_select(v, qmask)
        c = torch.masked_select(v, ~qmask)
        chunks = torch.split(c, chunk_size)

        for i, chunk in enumerate(chunks):
            if i not in chunked_input:
                chunked_input[i] = {}

            thing = torch.cat((q, chunk))
            if i != len(chunks)-1:
                if k == 'input_ids':
                    thing = torch.cat((thing, torch.tensor([102])))
                else:
                    thing = torch.cat((thing, torch.tensor([1])))

            chunked_input[i][k] = torch.unsqueeze(thing, dim=0)
    return chunked_input


def convert_ids_to_string(tokenizer, input_ids):
    return tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(input_ids))


def get_answer(tokenizer, model, chunked, inputs):
    if chunked:
        answer = ''
        for k, chunk in inputs.items():
            answer_start_scores, answer_end_scores = model(**chunk)

            answer_start = torch.argmax(answer_start_scores)
            answer_end = torch.argmax(answer_end_scores) + 1
            ans = convert_ids_to_string(tokenizer, chunk['input_ids'][0][answer_start:answer_end
                ↪ ])
            if ans != '[CLS]':
                answer += ans + ” / ”
        return answer
    else:
        answer_start_scores, answer_end_scores = model(**inputs)

    answer_start = torch.argmax(answer_start_scores) # get the most likely beginning of
        ↪ answer with the argmax of the score
    answer_end = torch.argmax(answer_end_scores) + 1 # get the most likely end of answer
        ↪ with the argmax of the score

    return convert_ids_to_string(tokenizer, inputs['input_ids'][0][answer_start:answer_end],
        ↪  tokenizer)
```

C.2 Get Answer with BERT model using a Pipeline

Listing C.2: Get Answer with BERT model using a Pipeline

```
import torch
from transformers import pipeline, BertTokenizer, BertForQuestionAnswering

# model_path is the location of the folder containing the trained BERT model files

# initialize BERT tokenizer
tokenizer = BertTokenizer.from_pretrained(model_path, use_fast=False)

# initialize BERT model
model = BertForQuestionAnswering.from_pretrained(model_path, return_dict=False)

# initialize the Pipeline
pipe = pipeline('question-answering', model=model, tokenizer=tokenizer)

def get_answer_with_pipeline(pipe, context, question):
   answer = pipe(
      {
         'question': question,
         'context': context
      }
   )
   return answer
```

Appendix D

COVID-19 Testing QA Set (10 QA pairs)

This appendix contains the QA set used for testing the application. It is composed of ten question-answer pairs in English and their translation to Portuguese. This set was adapted from *Coronavirus disease (COVID-19) FAQs*.

D.1 English QA Set

ENQ1 **How does COVID-19 spread?**

SARS-CoV-2, the virus that causes COVID-19, can spread from person to person through droplets produced during coughing or breathing during close contact with an infected individual. Infection can also occur indirect contact when these droplets land on objects and surfaces around the infected individual and the other person touches these objects or surfaces, then touches their eyes, nose or mouth. This is why it is important to stay at least 1-2 meters (3-6 feet) away from a person who is sick. Given that some individuals have no symptoms while still infected with the virus, physical distancing of 1-2 meters should be observed regardless of whether the other person seems sick.

ENQ2 **What are the symptoms of COVID-19?**

The most common symptoms of COVID-19 are fever, cough and fatigue. Some patients may have loss of taste or smell, conjunctivitis, headache, muscle aches and pains, nasal congestion, runny nose, sore throat, diarrhea, nausea or vomiting, and different types of skin rashes. These symptoms are usually mild and begin gradually. Some people become infected but don't develop any symptoms and don't feel unwell. Most people (about 80%) recover from the disease without needing special management. Approximately 1 out of every 6 people who get COVID-19 becomes seriously ill and develops symptoms of severe COVID-19, which include difficulty breathing/shortness of breath, confusion, loss of appetite, persistent pain or pressure in the chest, and needs hospitalization.

Older people and those with underlying medical problems like high blood pressure, heart problems or diabetes are more likely to develop serious illness. People with fever, cough and difficulty breathing should seek medical attention.

ENQ3 **How do I know if it is COVID-19 or just the common cold?**

A COVID-19 infection has similar signs and symptoms as the common cold or influenza, and you can only differentiate them through laboratory testing to determine the virus type.

ENQ4 **Can the virus that causes COVID-19 be transmitted through the air?**

Studies to date suggest that the virus that causes COVID-19 is mainly transmitted through respiratory droplets rather than through the air. See previous answer on "How does COVID-19 spread?" Aerosols may be generated during certain medical procedures and other activities, such as singing, but are not considered the predominant route of spread for this infection.

ENQ5 **What can I do to protect myself and prevent the spread of disease?**

Review the latest information on the COVID-19 pandemic available in the WHO website and through your national and local public health authority. The situation is dynamic so check regularly for the latest news.

You can reduce your chances of being infected or spreading COVID-19 by taking the following precautions:

- Clean your hands regularly and thoroughly with an alcohol-based hand rub or wash them with soap and water. Washing your hands with soap and water or using alcohol-based hand rub kills viruses that may be on your hands.

- Maintain at least 1-2-meter (3-6 feet) distance between yourself and anyone who is coughing or sneezing. When someone coughs or sneezes, they spray small liquid droplets from their nose or mouth which may contain virus. If you are too close, you can breathe in the droplets, including the COVID-19 virus if the person coughing or sneezing has the disease.

- Follow physical distancing rules of at least 1-2-meter (3-6 feet) distance between yourself and others regardless of whether they are showing symptoms.

- Avoid touching your eyes, nose and mouth. Hands touch many surfaces and can pick up viruses. Once contaminated, your hands can transfer the virus to your eyes, nose or mouth. From there, the virus can enter your body and can make you sick.

- Make sure you and the people around you follow good respiratory hygiene. This means covering your mouth and nose with your bent elbow or tissue when coughing or sneezing, then disposing of the used tissue immediately. Droplets spread virus. By following good respiratory hygiene, you protect the people around you from viruses such as the common cold, flu and COVID-19.

- Stay home if you feel unwell. If you have a fever, cough and difficulty breathing, seek medical attention and call in advance. Follow the directions of your local health authority. National and local authorities will have the most up to date information on the situation in your area. Calling in advance will allow your health care provider to quickly direct you to the right health facility. This will also protect you and help prevent spread of viruses and other infections.

- Wear a mask for the duration of your illness and while you have symptoms.as source control to prevent onward spread of COVID-19 if you are infected.

- Wear a mask as part of the comprehensive public health measures targeted to prevent the spread of COVID-19 even if you do not have symptoms and/or are not infected.

- Keep up to date on the latest COVID-19 hotspots (cities or local areas where COVID-19 is spreading widely). If possible, avoid traveling to those places – especially if you are an older person or have diabetes, heart or lung disease, because you have a higher chance of catching COVID-19 in one of these areas.

ENQ6 **Should I wear a mask while exercising?**

The WHO recommends that masks not be worn during vigorous physical activity. Please ensure a 1-2-meter distance from others when exercising and that there is adequate ventilation.

ENQ7 **Are the symptoms of COVID-19 different in children than in adults?**

The symptoms of COVID-19 are similar in children and adults. However, children with confirmed COVID-19 have generally presented with mild symptoms. Although children tend to have a milder disease, critical illness have been reported. A COVID-19 Multisystem Inflammatory Syndrome in Children (MIS-C) has also been described in children and adolescents.

ENQ8 **Are antibiotics effective in preventing or treating COVID-19?**

Antibiotics are used to treat bacterial infections. Since COVID-19 is a virus, antibiotics are not indicated for the direct treatment. However, it may be required in some instances, such as for treating secondary bacterial infections.

ENQ9 **Can humans become infected with COVID-19 from an animal source?**

Possible animal sources of COVID-19 have not yet been confirmed though are postulated. To protect yourself, such as when visiting live animal markets, avoid direct contact with animals and surfaces in contact with animals. Ensure good food safety practices at all times. Handle raw meat, milk or animal organs with care to avoid contamination of uncooked foods and avoid consuming raw or undercooked animal products.

ENQ10 **How long does the virus survive on surfaces?**

It is not certain how long the virus that causes COVID-19 survives on surfaces, but it seems to behave like other coronaviruses. Studies suggest that coronaviruses (including preliminary information on the COVID-19 virus) may persist on surfaces for a few hours or up to several days. This may vary under different conditions (e.g. type of surface, temperature or humidity of the environment).

If you think a surface may be infected, clean it with simple disinfectant to kill the virus and protect yourself and others. Cleaning your hands with an alcohol-based hand rub or washing them with soap and water is very important. Avoid touching your eyes, mouth, or nose.

D.2 Portuguese QA Set

PTQ1 **Como se transmite a COVID-19?**

O SARS-CoV-2, o vírus que causa a COVID-19, pode se espalhar de pessoa para pessoa por meio de gotículas produzidas durante a tosse ou respiração durante o contato próximo com um indivíduo infetado. A infeção também pode ocorrer por contato indireto, quando essas gotículas pousam em objetos e superfícies ao redor do indivíduo infetado e a outra pessoa toca esses objetos ou superfícies e, a seguir, toca seus olhos, nariz ou boca. É por isso que é importante ficar a pelo menos 1 a 2 metros (3 a 6 pés) de distância de uma pessoa doente. Dado que alguns indivíduos não apresentam sintomas quando estão infetados com o vírus, deve-se manter um distanciamento físico de 1-2 metros, independentemente de a outra pessoa parecer doente ou não.

PTQ2 **Quais são os sintomas da COVID-19?**

Os sintomas mais comuns de COVID-19 são febre, tosse e fadiga. Alguns pacientes podem apresentar perda de paladar ou olfato, conjuntivite, dor de cabeça, dores musculares, congestão nasal, corrimento nasal, dor de garganta, diarreia, náusea ou vómito e diferentes tipos de erupções cutâneas. Esses sintomas são geralmente leves e começam gradualmente. Algumas pessoas ficam infetadas, mas não desenvolvem quaisquer sintomas e não se sentem mal. A maioria das pessoas (cerca de 80 %) recupera da doença sem precisar de tratamento especial. Aproximadamente 1 em cada 6 pessoas que apanham COVID-19 fica gravemente doente e desenvolve sintomas de COVID-19 grave, que incluem dificuldade em respirar / falta de ar, confusão, perda de apetite, dor persistente ou pressão no peito e precisa de hospitalização . Idosos e pessoas com problemas médicos subjacentes, como hipertensão, problemas cardíacos ou diabetes, têm maior probabilidade de desenvolver sintomas graves. Pessoas com febre, tosse e dificuldade respiratória devem procurar atendimento médico.

PTQ3 **Como posso saber se tenho COVID-19 ou uma constipação?**

Uma infeção por COVID-19 tem sinais e sintomas semelhantes aos da constipação comum ou gripe, e só se pode diferenciá-los por meio de testes laboratoriais para determinar o tipo de vírus.

PTQ4 **O vírus que transmite a COVID-19 pode ser transmitido pelo ar?**

Os estudos até ao momento indicam que o vírus que causa COVID-19 é transmitido principalmente por gotículas respiratórias, e não pelo ar. Consulte a resposta anterior em "Como se transmite a COVID-19?". Aerossóis podem ser gerados durante certos procedimentos médicos e outras atividades, como cantar, mas não são considerados o método predominante de propagação desta infeção.

PTQ5 **O que posso fazer para me proteger e evitar a disseminação da doença?**

Reveja as informações mais recentes sobre a pandemia de COVID-19 disponíveis no site da OMS ou através da sua autoridade nacional e local de saúde pública. A situação é dinâmica, portanto, verifique regularmente as últimas notícias.

Pode reduzir as hipóteses de estar infectado ou espalhar COVID-19 tendo em consideração as seguintes precauções:

- Limpe as mãos regularmente e cuidadosamente com um produto à base de álcool ou lave-as com água e sabão. Lavar as mãos com água e sabão ou usar um produto à base de álcool elimina os vírus que podem estar nas suas mãos.

- Mantenha uma distância de pelo menos 1-2 metros (3-6 pés) entre si e qualquer pessoa que esteja a tossir ou a espirrar. Quando alguém tosse ou espirra, pequenas gotas de líquido do nariz ou da boca que podem conter vírus são espalhadas. Se estiver muito perto, pode respirar as gotículas, incluindo o vírus COVID-19, se a pessoa que tosse ou espirra tem a doença.

- Siga as regras de distanciamento físico de pelo menos 1 a 2 metros (3 a 6 pés) de distância entre si e os outros, independentemente de apresentarem ou não sintomas.

- Evite tocar nos olhos, nariz e boca. As mãos tocam em muitas superfícies e podem apanhar vírus. Uma vez contaminadas, as suas mãos podem transferir o vírus para os seus olhos, nariz ou boca. A partir daí, o vírus pode entrar no seu corpo e deixá-lo doente.

- Certifique-se de que o próprio e as pessoas ao seu redor seguem uma boa higiene e etiqueta respiratória. Isso significa cobrir a boca e o nariz com o cotovelo dobrado ou com um lenço de papel ao tossir ou espirrar e, em seguida, descartar o lenço usado imediatamente. As gotas espalham vírus. Ao seguir uma boa higiene respiratória, protege as pessoas ao seu redor de vírus como a constipação comum, gripe e COVID-19.

- Fique em casa se não se sentir bem. Se tiver febre, tosse e dificuldade em respirar, procure atendimento médico e ligue com antecedência. Siga as instruções da autoridade de saúde local. As autoridades nacionais e locais terão as informações mais atualizadas sobre a situação na sua área. Ligar com antecedência permitirá que o seu médico o encaminhe rapidamente para a unidade de saúde certa. Isso também o protegerá e ajudará a prevenir a propagação de vírus e outras infeções.

- Use uma máscara durante a doença e enquanto tiver sintomas, de maneira a evitar a propagação de COVID-19 se estiver infectado.

- Use uma máscara como parte das medidas abrangentes de saúde pública destinadas a prevenir a propagação de COVID-19, mesmo não tendo sintomas e/ou não estiver infectado.

- Mantenha-se atualizado sobre os pontos de acesso COVID-19 mais recentes (cidades ou áreas locais onde a COVID-19 está a espalhar-se amplamente). Se possível, evite viajar para esses lugares, especialmente se for uma pessoa idosa ou tiver diabetes, doença cardíaca ou pulmonar, porque tem uma maior probabilidade de apanhar COVID-19 numa dessas áreas.

PTQ6 **Devo usar máscara enquanto faço exercício?**

A OMS recomenda que as máscaras não sejam utilizadas durante atividades físicas vigorosas. Certifique-se de que mantenha uma distância de 1-2 metros de outras pessoas ao exercitar-se e de que haja ventilação adequada.

PTQ7 **Os sintomas da COVID-19 são diferentes nos adultos e nas crianças?**

Os sintomas de COVID-19 são semelhantes em crianças e adultos. No entanto, as crianças com COVID-19 confirmado geralmente apresentam sintomas leves. Embora as crianças tendam a ter uma doença leve, sintomas críticos foram também relatados. Um síndrome inflamatório multissistémico COVID-19 em crianças (MIS-C) também foi observado em crianças e adolescentes.

PTQ8 **Os antibióticos são eficazes na prevenção ou tratamento da COVID-19?**

Antibióticos são usados para tratar infeções bacterianas. Por ser o COVID-19 um vírus, os antibióticos não são indicados para o tratamento direto. No entanto, podem ser necessários em alguns casos, como para o tratamento de infeções bacterianas secundárias.

PTQ9 **Os humanos podem ser infetados com a COVID-19 através de animais?**

Possíveis fontes animais de COVID-19 ainda não foram confirmadas, embora sejam postuladas. Para se proteger, como ao visitar mercados de animais vivos, evite o contato direto com animais e superfícies em contato com os animais. Garanta boas práticas de segurança alimentar em todos os momentos. Manuseie carne crua, leite ou órgãos de animais com cuidado para evitar a contaminação de alimentos crus e evite consumir produtos de origem animal crus ou mal cozinhados.

PTQ10 **Quanto tempo sobrevive o vírus nas superfícies?**

Não se sabe ao certo quanto tempo o vírus que causa a COVID-19 sobrevive em superfícies, mas parece comportar-se como outros coronavírus. Estudos sugerem que os coronavírus (incluindo informações preliminares sobre o vírus da COVID-19) podem persistir nas superfícies por algumas horas ou até vários dias. Isso pode variar em diferentes condições (por exemplo, tipo de superfície, temperatura ou humidade do ambiente).

Se acha que uma superfície pode estar infetada, limpe-a com um desinfetante simples para matar o vírus e proteger-se a si e a outras pessoas. É muito importante limpar as mãos com um produto à base de álcool ou lavá-las com água e sabão. Evite tocar nos seus olhos, boca ou nariz.