Question-Answering System for Coffee Machines

Marius Benkert, Jochen Schmidt, Lennard Rose

Abstract

Our study presents a comprehensive Question Answering (QA) system for coffee machine related questions. The system covers a wide range of topics such as maintenance, usage, and troubleshooting of coffee machines, offering quick and accurate answers to the users through its intuitive interface and natural language processing capabilities. The system provides a seamless experience for coffee machine owners and users to access the information they need, ensuring smooth operation of their machines and allowing them to enjoy their favorite beverages with ease. To evaluate the performance of these models, we fine-tuned a range of BERT-based Transformers on a manually created dataset of 653 question-answer pairs. In conclusion, our findings demonstrate the feasibility of using NLP Question Answering models to deliver technical answers about coffee machines, and highlight the importance of fine-tuning these models on task-specific data.

1 Introduction

Coffee machines have significantly evolved in recent decades. This development has been driven by advances in technology and the demand for greater convenience and automation leading to today's modern, fully automatic machines. By using a variety of brewing methods, these machines provide a more efficient and consistent brewing process and a greater variety of coffee preparation options.

However, the increasing complexity of the machines and the introduction of new and advanced features made it difficult for users to fully understand and utilize all of the machines' capabilities. Therefore, the integration of new software and technology into everyday devices raised the need for more information to help users navigate the complex appliances, which resulted in user manuals becoming increasingly complicated.

Higher complexity also makes searching through the technical documents of these machines to find an answer to a specific question a more tedious task. Different manufacturers have different layouts and styles in writing up the instructions for their products. Even in products from the same manufacturer, it is not uncommon to find different structures within their documentations. This further makes it a more challenging task to find the specific answer to a question.

As a result, researching and developing techniques that can quickly and accurately return results for the imposed question within a specific manual could be beneficial. Such techniques could save both tedious work and time.

Recent advancements in Natural Language Processing (NLP) and Natural Language Understanding (NLU) driven by the use of transformer-based Large Language Models (LLMs) have enabled the development of various natural language based applications. A use case for such a LLM would be to extract answers to questions from a defined context, also known as extractive question-answering [10, 13].

Our approach addresses the difficulty of finding answers to specific questions within technical documents with an application that provides answers to user inquiries regarding coffee machines. This application is an end-to-end question answering system for coffee machine manuals. To the best of our knowledge, this is the first of its kind in the particular domain.

To begin, we collected, stored and preprocessed the coffee machine manuals. In a second step, the created corpus was used to build a custom question-answering dataset. Different transformer-based models have been tested and compared in their ability to answer questions with respect to our domain.

Additionally, we provide a web service where users can ask questions regarding their coffee machine.

The following chapters contain a detailed description of the implementation, as well as the reasons for the choice of the methods used. Section 2 provides an overview of the design choices made to create a scalable, resilient application. Section 3 describes the methods we used to collect and preprocess the data needed in training the model and to serve as a basis for the answers. Section 4.1 outlines the training and evaluation of the model. The main outcomes of the research and also suggestions for future work are in section 5.

2 Q&A System Architecture

2.1 Design

We present a data-centric approach to building a question answering system for coffee machines [2]. The logical procedure for the app can be split in two sub-processes.
In the first sub-process, the necessary data is collected and preprocessed to build a corpus. Further, the data is transformed to a form that can be utilized quickly and easily. This involves downloading manual documents and storing them. The second step involves preprocessing the data, which includes extracting the text from the PDFs, segmenting and separating the text by language to form a multilingual corpus. This corpus is further split up to paragraphs which are embedded for better processing. This procedure is explained in detail in section 3.
The second sub-process represents the actual functionality of the app. A webservice allows users to ask questions about coffee machines. For this, suitable context for the answer has to be extracted from the database. Based on these context, the used BERT-model generates answers to be presented to the user. The components and procedure of the service is further described in section 4

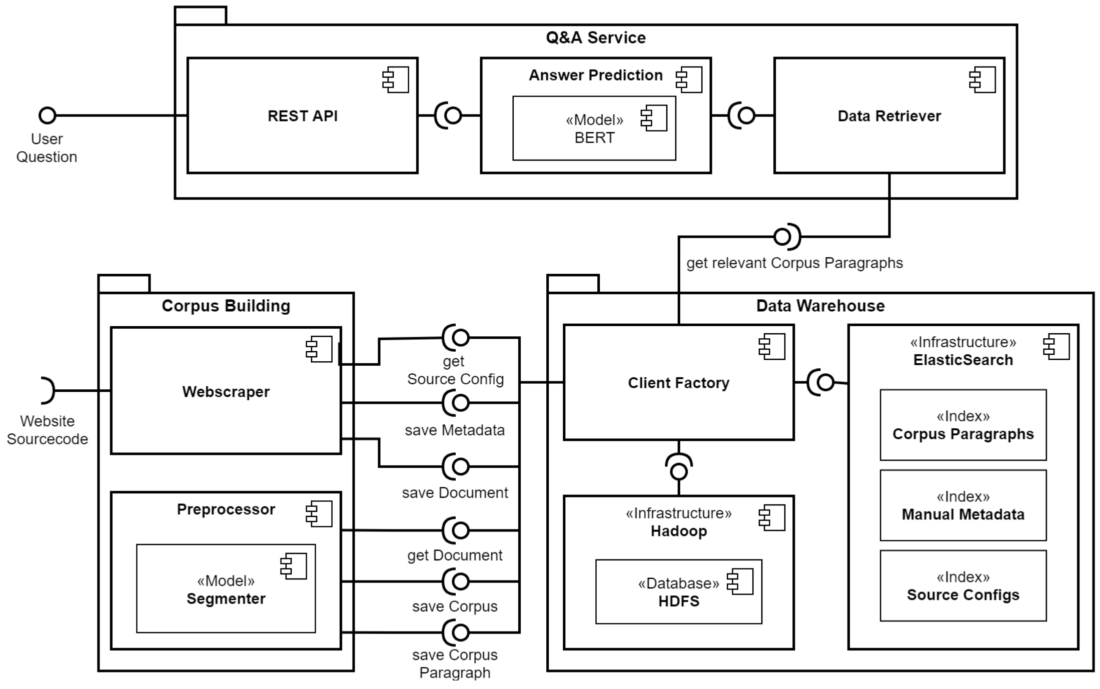


Figure 1: Applications Component Diagram

As seen in figure 1, the sub-processes are modeled in the applications architecture.

The process of data collection and processing is represented by the logical package *Corpus Building.* This contains the web scraper for collecting the documents and the preprocessor.
The question answering pipeline is modeled in the *Q&A Service* package. The structure follows the logical order from receiving the question request - processing the question - retrieving the data - processing the data - sending the answer response.
The data warehouse acts as the center point between both sub-processes. On the one hand, it stores all outputs from the *Corpus Building* package. On the other hand, it makes this stored data accessible easily and rapidly for the *Q&A service.* More about the data warehouse in the next section 2.2.

2.2 Data Warehouse

We chose a data warehouse as a centralized repository for storing and managing the large amounts of different data and to provide a single source of truth for the whole application. The data warehouse is optimized for consistent data storage, high availability as well as fast query and access.

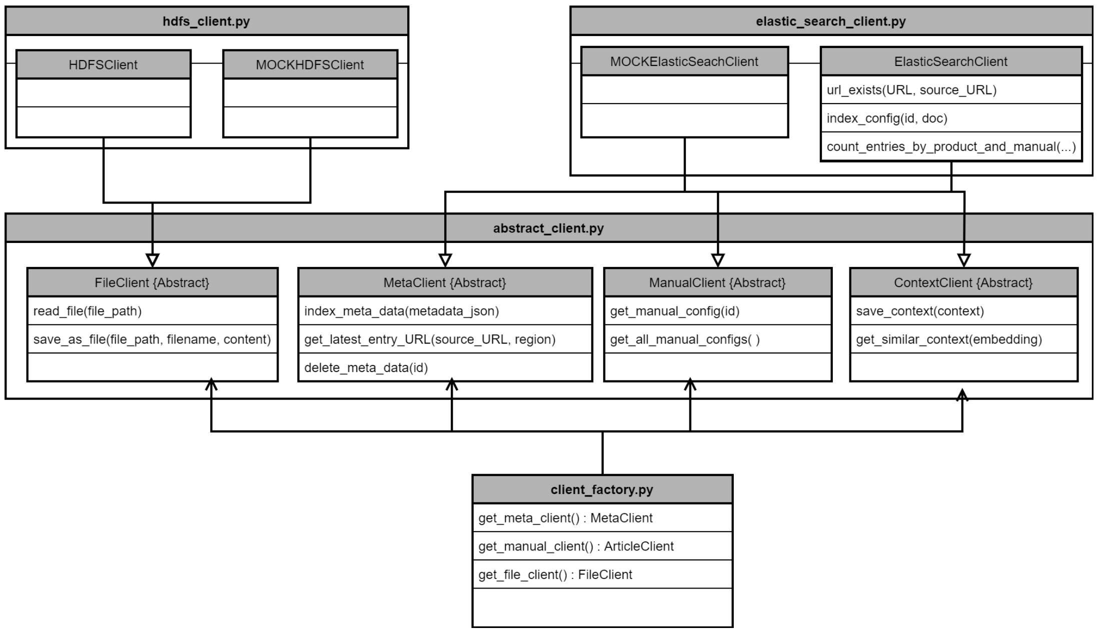


Figure 2: Data Warehouse Class Diagram

The abstract factory was the central module for building the data warehouse. The pattern allowed the encapsulation of client creation logic and to decouple the implementation of the database access to switch between different database and search engine implementations without affecting the overall system. Further, the data warehouse has to handle multiple types of data sources and structures, the abstract factory pattern simplifies this by encapsulating the logic for each kind of data. With changing requirements, this makes it easier to maintain and modify the system.
We defined four different abstract clients to handle our data types:

- FileClient: stores the PDF documents as well as the corpus documents
- MetaClient: stores meta data of other documents and data for easy query
- ManualClient: stores configurations needed to scrape documents from manufacturers websites
- ContextClient: stores the paragraphs that are used as a context for the question answering model

As the main data storage we chose Apache Hadoop. Hadoop provides a scalable and flexible solution for cost effective storage of large amounts of data storage [8]. Hadoop's distributed file system, HDFS, allows data to be stored across multiple nodes, providing high availability and fault tolerance. Additionally, this distributed file system makes Hadoop exceptionally scalable.

To rapidly search and access data we used the popular search engine Elasticsearch [3]. It has, like Hadoop, a distributed architecture which enables Elasticsearch a near real-time search capability across large amounts of data [6]. We utilized this to be able to search for documents in the HDFS via the meta data stored in Elasticsearch. Further, we used the inbuilt similarity search (see section 4.2 for more information) to facilitate the retrieval of context paragraphs for the Question Answering Service.

3 Data Collection & Processing

As mentioned in Architecture - Design (subsection 2.1), the application can be separated in two parts - collection and service. This subsection explains the steps necessary to provide the data required for the question answering service to operate.

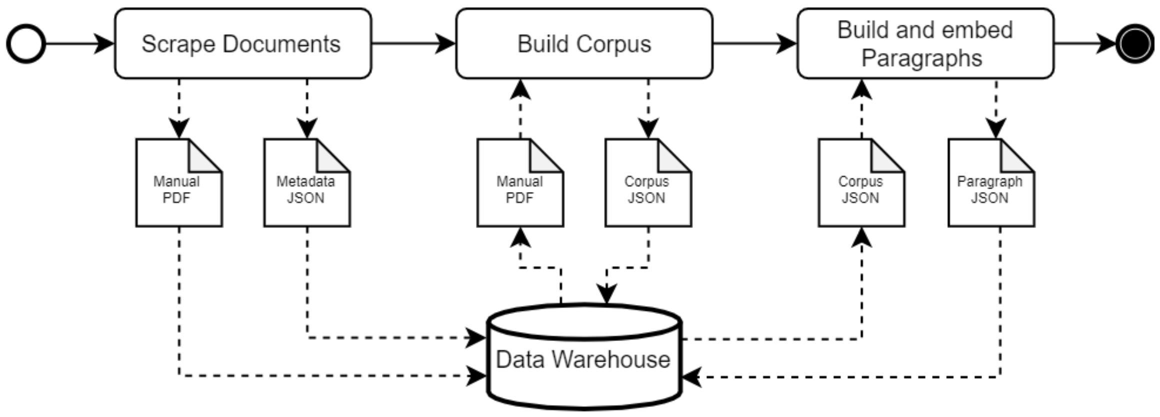


Figure 3: Data Processing Procedure

Figure 3 outlines the process of providing data. Starting with the collection process described in section Webscraping. Section Preprocessing explains the subsequent steps to make the data ready for use.

3.1 Webscraping

In order to apply natural language processing (NLP) techniques, a data basis is required. For that, we utilize web scraping. Web scraping is a method of automatically extracting data from websites and is widely used in the field of NLP for data collection. This automated and easily configurable method allowed us to gather large amounts of data from coffee machine manufacturer websites in a fast and efficient manner. Our web scraper can be easily configured to scrape data from additional sources, making it a flexible tool for gathering data for our question answering system.

The web scraper was developed from the ground up using Python programming language due to its comprehensive collection of libraries suitable for web interactions and scraping processes. The key packages employed in this implementation are *BeautifulSoup* [17] and *Selenium* [14]. *BeautifulSoup* is utilized to parse the web page and produce an accessible object from which relevant information can be extracted. However, some websites are not easily scrapable due to their use of JavaScript to dynamically load and render elements. To address this issue, we used *Selenium*'s WebDriver, which serves as an interface to a local web browser, enabling the execution of JavaScript scripts on the pages. This, in turn, enables *BeautifulSoup* to extract even dynamically loaded content.

Our web scraper was created to be flexible and easily configurable. It is configured by a JSON file for each source, enabling the simple addition of new sources. The scraper accesses the manufacturer's website and performs a breadth-first search for relevant products (see Figure 4), after which it accesses the detail pages to extract the meta data and manuals. We differentiate between the PDF files and the

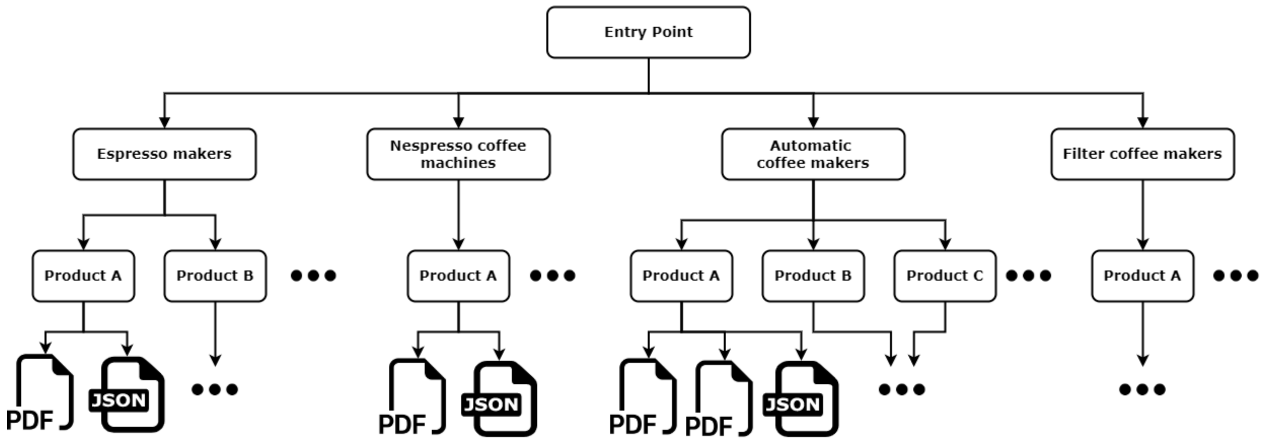


Figure 4: Example Hierarchy View of a Website

meta data, such as the product name, manual name, and product type. The meta data is saved in our data warehouse using Elasticsearch, while the manuals are stored in the Hadoop distributed file system (HDFS).

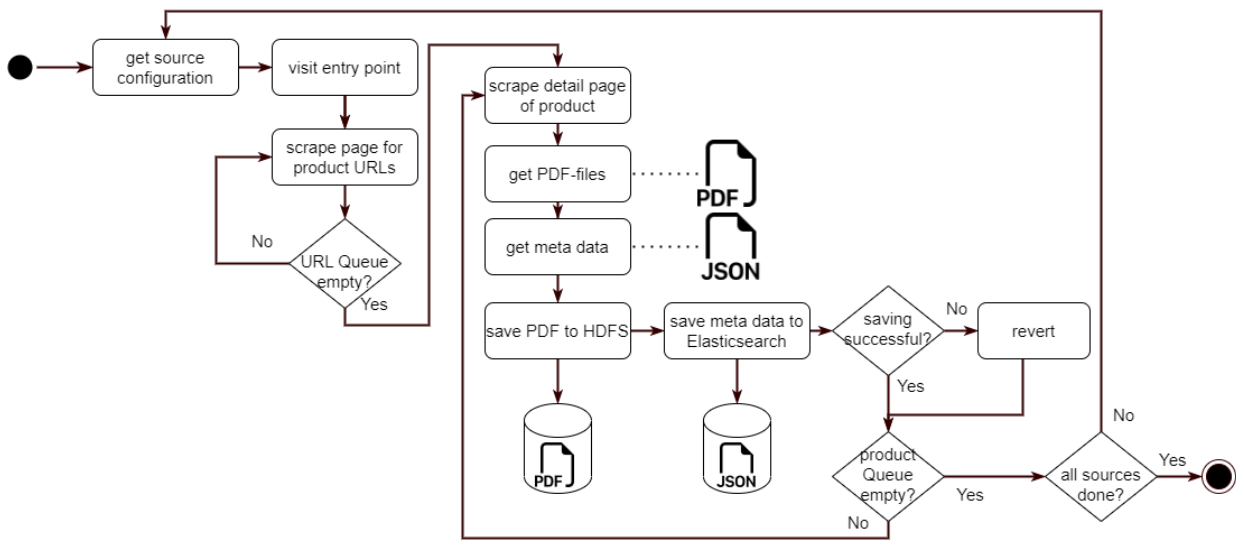


Figure 5: Flow Chart of the Web Scraper

By automatically extracting data from manufacturer websites, we were able to gather large amounts of information that would otherwise have been difficult and time-consuming to obtain. Nevertheless, in the case of a collection of available offline PDFs, it is possible to carry out the subsequent preprocessing steps with prior entry of the metadata. The data collected are the basis for the fine-tuning of our NLP model and later for providing the relevant context to the question of the user. In our data pipeline, web scraping is the initial and crucial step.

3.2 Preprocessing

With the crawled manuals being in PDF format, they first had to be preprocessed in order to be useful for downstream NLP tasks. There already exist a number of libraries that can easily extract the plain text from the meta data of PDF's [1]. Though, for this project the goal was to create a parallel, multilingual corpus, which relies on more information than just the plain text.

	AEG	**Braun**	**Delonghi**	**Jura**	**Krups**	**Melitta**	**Miele**	**Philipps**	**Russelhobbs**	**Sage**
Header AP	68.7	64.3	65.1	72.3	70.4	54.2	65.1	64.8	70.4	78.0
Subheader AP	70.4	63.6	58.6	67.0	57.4	65.3	65.8	68.0	61.1	71.0
mAP	70.0	64.0	62.3	70.0	65.7	59.6	65.5	66.4	65.8	74.5

Table 1: Average precision (AP) and mean average precision (mAP) in % obtained by the individual models on the the two classes Header and Subheader.

Definitie 1 *A parallel, multilingual text corpus consists of text segments in various languages that are aligned at a specific level such that each segment has translations or equivalents in other languages.*

In our case, alignment within the resulting corpus means having the same structure of headings, subheadings and paragraphs for every language available in the manuals. In order to obtain such a structure, additional segmentation of the documents is required.

3.2.1 Segmentation

Segmenting documents is not a new problem and a lot of different solutions have already been proposed. These solutions range from strictly rule-based segmentation to solutions where machine learning is involved. After initial investigation of the different kind of document layouts we are dealing with, we decided on using object detection methods to detect the headlines and sub-headlines within the manuals. In particular, the LayoutParser toolkit [20] was used for this purpose. They are building on top of the Detectron2 library [23] and provide various tools and resources [19] that help to reduce the amount of effort that is needed to train, evaluate and use custom object detection models.

In order to train such a custom model, several preparation steps needed to be completed (Figure 6). The first step for training a custom model was to convert the individual pages of the manuals into images. For the reason that the different manufacturers sometimes had very different layouts, a separate training set and model is created for each manufacturer. In a second step, the headings and subheadings had to be manually annotated within the selected images. Because of the fact that the labeling process was very time consuming, only around 100 images per manufacturer were annotated and used for the training of the individual models. For the purpose of annotating we used the open source data labeling platform Label Studio [21], which in the end provided a file export for the labeled data in COCO format.

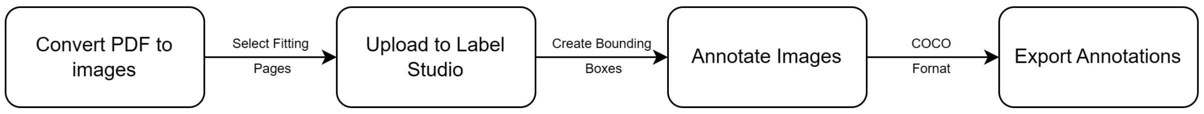
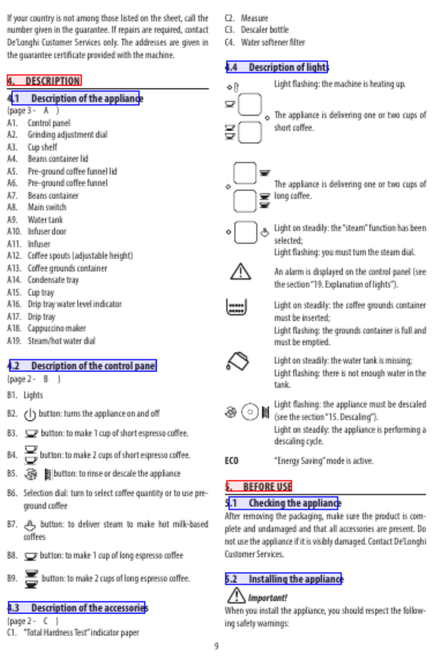


Figure 6: Pipeline for the preparation of training a custom object detection model.

The obtained COCO files for each manufacturer had to be split up into training and testing sets. These sets, together with an accompanying config file for the model, could then be fed into the training script that is provided in [19]. In the config file it is specified what model architecture should be used and how it should be trained. In our case, we used a Fast R-CNN with a ResNet-50 as the backbone [7], where large parts of the ResNet weights used were already pretrained on the ImageNet database [4] and only finetuned on our created training data.

For evaluating how good the individual models perform, we used the Average Precision (AP) and the mean Average Precision (mAP) of the predicted bounding boxes as described in [12]. Over all manufacturers combined we average a mAP of 66.4 (see Table 1). Considering that we only used around 100 images per manufacturer as training data, the results are pretty decent. As most deep learning architectures need a lot of data to really excel at their given task, having more training samples and more variations within our training data would probably increase the performance of the individual classifiers greatly.

Figure 7 showcases how the trained models predict the coordinates as well as the labels for the bounding-boxes for one page at a time. Comparing only the two manufacturers shown, it becomes



If your country is not among those listed on the sheet, call the number given in the guarantee. If repairs are required, contact De'Longhi Customer Services only. The addresses are given in the guarantee certificate provided with the machine.

4. DESCRIPTION

4.1 Description of the appliance

(page 3 - A)

A1. Control panel
A2. Grinding adjustment dial
A3. Cup shelf
A4. Beans container lid
A5. Pre-ground coffee funnel lid
A6. Pre-ground coffee funnel
A7. Beans container
A8. Main switch
A9. Water tank
A10. Infuser door
A11. Infuser
A12. Coffee spouts (adjustable height)
A13. Coffee grounds container
A14. Condensate tray
A15. Cup tray
A16. Drip tray water level indicator
A17. Drip tray
A18. Cappuccino maker
A19. Steam/hot water dial

4.2 Description of the control panel

(page 2 - B)

B1. Lights
B2. button: turns the appliance on and off
B3. button: to make 1 cup of short espresso coffee.
B4. button: to make 2 cups of short espresso coffee.
B5. button: to rinse or descale the appliance
B6. Selection dial: turn to select coffee quantity or to use pre-ground coffee
B7. button: to deliver steam to make hot milk-based coffees
B8. button: to make 1 cup of long espresso coffee
B9. button: to make 2 cups of long espresso coffee.

4.3 Description of the accessories

(page 2 - C)

C1. "Total Hardness Test" indicator paper
C2. Measure
C3. Descaler bottle
C4. Water softener filter

4.4 Description of lights

Light flashing: the machine is heating up.

The appliance is delivering one or two cups of short coffee.

The appliance is delivering one or two cups of long coffee.

Light on steadily: the "steam" function has been selected;
Light flashing: you must turn the steam dial.

An alarm is displayed on the control panel (see the section "19. Explanation of lights").

Light on steadily: the coffee grounds container must be inserted;
Light flashing: the grounds container is full and must be emptied.

Light on steadily: the water tank is missing;
Light flashing: there is not enough water in the tank.

Light flashing: the appliance must be descaled (see the section "15. Descaling").
Light on steadily: the appliance is performing a descaling cycle.

ECO "Energy Saving" mode is active.

5. BEFORE USE

5.1 Checking the appliance

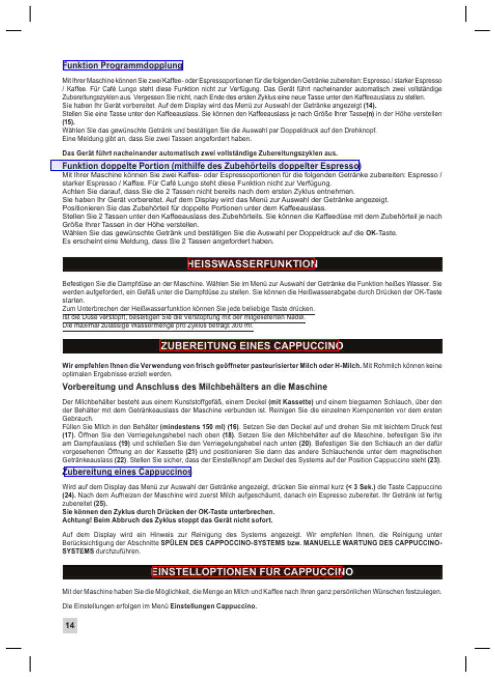
After removing the packaging, make sure the product is complete and undamaged and that all accessories are present. Do not use the appliance if it is visibly damaged. Contact De'Longhi Customer Services.

5.2 Installing the appliance

Important!
When you install the appliance, you should respect the following safety warnings:

9

(a) Bboxes obtained for one of the Delonghi manuals.



Funktion Programmdopplung

Mit Ihrer Maschine können Sie zwei Kaffee- oder Espressoportionen für die folgenden Getränke zubereiten: Espresso / starker Espresso / Kaffee. Für Café Lungo steht diese Funktion nicht zur Verfügung. Das Gerät führt nacheinander automatisch zwei vollständige Zubereitungszyklen aus. Vergessen Sie nicht, nach Ende des ersten Zyklus eine neue Tasse unter den Kaffeeauslass zu stellen.
Sie haben Ihr Gerät vorbereitet. Auf dem Display wird das Menü zur Auswahl der Getränke angezeigt **(14)**.
Stellen Sie eine Tasse unter den Kaffeeauslass. Sie können den Kaffeeauslass je nach Größe Ihrer Tasse(n) in der Höhe verstellen **(15)**.
Wählen Sie das gewünschte Getränk und bestätigen Sie die Auswahl per Doppeldruck auf den Drehknopf.
Eine Meldung gibt an, dass Sie zwei Tassen angefordert haben.

Das Gerät führt nacheinander automatisch zwei vollständige Zubereitungszyklen aus.

Funktion doppelte Portion (mithilfe des Zubehörteils doppelter Espresso)

Mit Ihrer Maschine können Sie zwei Kaffee- oder Espressoportionen für die folgenden Getränke zubereiten: Espresso / starker Espresso / Kaffee. Für Café Lungo steht diese Funktion nicht zur Verfügung.
Achten Sie darauf, dass Sie die 2 Tassen nicht bereits nach dem ersten Zyklus entnehmen.
Sie haben Ihr Gerät vorbereitet. Auf dem Display wird das Menü zur Auswahl der Getränke angezeigt.
Positionieren Sie das Zubehörteil für doppelte Portionen unter dem Kaffeeauslass.
Stellen Sie 2 Tassen unter den Kaffeeauslass des Zubehörteils. Sie können die Kaffeedüse mit dem Zubehörteil je nach Größe Ihrer Tassen in der Höhe verstellen.
Wählen Sie das gewünschte Getränk und bestätigen Sie die Auswahl per Doppeldruck auf die **OK**-Taste.
Es erscheint eine Meldung, dass Sie 2 Tassen angefordert haben.

HEISSWASSERFUNKTION

Befestigen Sie die Dampfdüse an der Maschine. Wählen Sie im Menü zur Auswahl der Getränke die Funktion heißes Wasser. Sie werden aufgefordert, ein Gefäß unter die Dampfdüse zu stellen. Sie können die Heißwasserabgabe durch Drücken der OK-Taste starten.
Zum Unterbrechen der Heißwasserfunktion können Sie jede beliebige Taste drücken.
~~Ist die Düse verstopft, beseitigen Sie die Verstopfung mit der mitgelieferten Nadel.~~
~~Die maximal zulässige Wassermenge pro Zyklus beträgt 300 ml.~~

ZUBEREITUNG EINES CAPPUCCINO

Wir empfehlen Ihnen die Verwendung von frisch geöffneter pasteurisierter Milch oder H-Milch. Mit Rohmilch können keine optimalen Ergebnisse erzielt werden.

Vorbereitung und Anschluss des Milchbehälters an die Maschine

Der Milchbehälter besteht aus einem Kunststoffgefäß, einem Deckel **(mit Kassette)** und einem biegsamen Schlauch, über den der Behälter mit dem Getränkeauslass der Maschine verbunden ist. Reinigen Sie die einzelnen Komponenten vor dem ersten Gebrauch.
Füllen Sie Milch in den Behälter **(mindestens 150 ml) (16)**. Setzen Sie den Deckel auf und drehen Sie mit leichtem Druck fest **(17)**. Öffnen Sie den Verriegelungshebel nach oben **(18)**. Setzen Sie den Milchbehälter auf die Maschine, befestigen Sie ihn am Dampfauslass **(19)** und schließen Sie den Verriegelungshebel nach unten **(20)**. Befestigen Sie den Schlauch an der dafür vorgesehenen Öffnung an der Kassette **(21)** und positionieren Sie dann das andere Schlauchende unter dem magnetischen Getränkeauslass **(22)**. Stellen Sie sicher, dass der Einstellknopf am Deckel des Systems auf der Position Cappuccino steht **(23)**.

Zubereitung eines Cappuccinos

Wird auf dem Display das Menü zur Auswahl der Getränke angezeigt, drücken Sie einmal kurz **(< 3 Sek.)** die Taste Cappuccino **(24)**. Nach dem Aufheizen der Maschine wird zuerst Milch aufgeschäumt, danach ein Espresso zubereitet. Ihr Getränk ist fertig zubereitet **(25)**.
Sie können den Zyklus durch Drücken der OK-Taste unterbrechen.
Achtung! Beim Abbruch des Zyklus stoppt das Gerät nicht sofort.

Auf dem Display wird ein Hinweis zur Reinigung des Systems angezeigt. Wir empfehlen Ihnen, die Reinigung unter Berücksichtigung der Abschnitte **SPÜLEN DES CAPPOCCINO-SYSTEMS bzw. MANUELLE WARTUNG DES CAPPUCCINO-SYSTEMS** durchzuführen.

EINSTELLOPTIONEN FÜR CAPPUCCINO

Mit der Maschine haben Sie die Möglichkeit, die Menge an Milch und Kaffee nach Ihren ganz persönlichen Wünschen festzulegen.

Die Einstellungen erfolgen im Menü **Einstellungen Cappuccino.**

14

(b) Bboxes obtained for one of the Krups manuals.

Figure 7: Obtained bounding boxes of the models on the example of two manufacturers. Red bounding boxes are the Header class. Blue bounding boxes are the Subheader class.

apparent how much the header and subheader representations can differ, which raised the need to train separate models. With the information about the location of the headers and subheaders on the pages, we can segment all the documents and store the information in a structured representation.

3.2.2 Corpus

Combining the plain text information obtained through the PyMuPDF library [22] as well as the bounding-box information obtained through the object detection, the parallel, multilingual corpus was built.

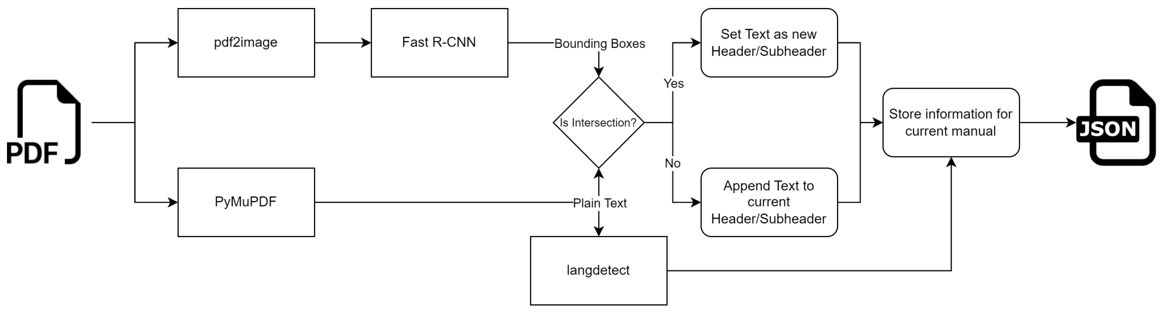

PDF
pdf2image
Fast R-CNN
Bounding Boxes
Is Intersection?
Yes
No
Set Text as new Header/Subheader
Append Text to current Header/Subheader
Store information for current manual
JSON
PyMuPDF
Plain Text
langdetect

Figure 8: General preprocessing procedure for one manual.

The process, as seen in Figure 8, was performed for every manual that was scraped from the manufacturers website. Through PyMuPDF we read in every page iteratively and got the textual information as well as their respective coordinates on the page. Each text block was compared with the detected bounding-boxes on the same page to check whether they intersected. If that was the case, a new current header/subheader was set. If they didn't intersect, the text block was appended to the currently set header/subheader. This way we could save the text blocks belonging to specific segments of the document correctly. Through another library called *langdetect* we also detected the language of the read-in manuals, which provided additional information for the structure of the resulting corpus.

3.2.3 Embedding

Additionally, to select only the most relevant paragraphs in the corpus for a given question, a Similarity Search was implemented. For that, all processed paragraphs within our corpus had to be embedded into some kind of vector representation, which is described in more detail in section 4.2.

4 Q&A Service

This section describes how the Q&A-Service utilizes the processed data from the previous section 3 to answer user questions.

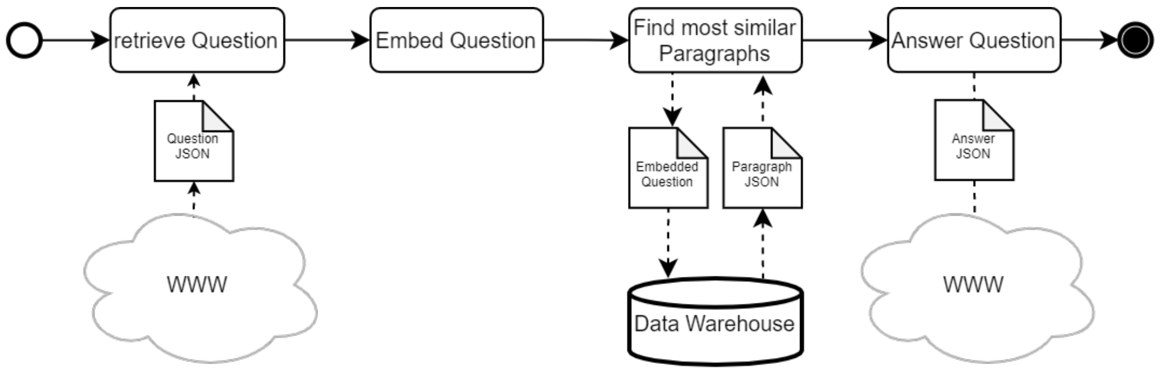


Figure 9: Question Answering Procedure

Figure 9 visualizes the general function of the Q&A-Service. First, the request containing the question as well as additional information is received by the API (see section 4.3. As described in section 4.2, the question is embedded in order to retrieve an appropriate context from the data warehouse. This context is then passed to the question-answer model (see next section 4.1) to extract an answer. The answer is then sent back as a response.

4.1 Model Training

In order to deliver precise and efficient answers to technical questions about coffee machines in our Question Answering System, we utilized Natural Language Processing (NLP). Within the NLP community, there are three distinct variants of Question Answering (QA) models:

- **Extractive QA**: The model extracts the answer from a provided context.
- **Open Generative QA**: The model generates free text based on the context.
- **Closed Generative QA**: In this scenario, no context is given, and the answer is generated entirely by the model.

Our approach adopts the Extractive Question Answering variant, where the context is delivered to the model via our data warehouse. Our data pipeline enables us to gather comprehensive and relevant documents from various manufacturers and products. By providing the most appropriate context, we anticipate to achieve a high level of accuracy in our results.

An important framework in the field of NLP is *HuggingFace* which serves as a comprehensive resource, providing not only datasets and (pre-trained) models, but also a community and educational resources [9].

For our experiment, we chose to test BERT-based Transformers, specifically BERT, DistilBERT [18], and RoBERTa [11]. BERT, which stands for Bidirectional Encoder Representations, was first introduced in [5]. These Transformers can be fine-tuned to meet specific task requirements with the addition of a single output layer, which *HuggingFace* makes accessible and convenient. We also utilized models that were pre-trained on the SQuAD 2.0 dataset [15] and fine-tuned them for our task. SQuAD, or Stanford Question Answering Dataset, comprises over $100,000$ questions based on Wikipedia articles, with answers drawn from corresponding passages [16]. SQuAD 2.0 expanded the dataset by adding an additional $50,000$ unanswerable questions.

To perform the fine-tuning, we created a dataset of 653 question-answer pairs, which were manually constructed based on the information in our data warehouse. The dataset was divided into 80% for training and 20% for testing. We evaluated the various models without fine-tuning and with fine-tuning using the F1 and Exact-Match metrics, which are the most commonly used evaluation metrics in QA.

Exact-Match The Exact-Match metric is calculated by comparing the predicted answer and the actual answer character by character. If the predicted answer exactly matches the actual answer, the Exact-Match score is 1. If there is any difference, no matter how small, the Exact-Match score is 0. This metric is strict and considers an answer either correct or incorrect with no room for partial accuracy.

F1 F1 score is a common metric in classification problems and widely used in QA. It combines the precision and recall metrics to provide a measure of the overlap between the predicted answer and the true answer:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = 2\frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

Precision is the ratio of the number of correctly predicted words to the total number of words in the prediction, while *recall* is the ratio of the number of correctly predicted words to the total number of words in the true answer. The F1 score represents the percentage of shared words between the prediction and the true answer.

Model	with Fine-Tuning		without Fine-Tuning	
	F1	Exact-Match	F1	Exact-Match
bert-base-uncased	61.7	54.6	10.4	0.3
distilbert-base-uncased-squad	66.4	57.0	49.9	21.1
bert-base-cased-squad2	67.6	55.8	52.3	22.0
roberta-base	66.7	57.2	14.6	0.15
roberta-base-squad2	**68.3**	**59.8**	**58.3**	**27.5**

Table 2: Evaluation Results of different Models on our Dataset

Based on the results in Table 2 where we tested base and already fine-tuned variants without our fine-tuning and with fine-tuning the RoBERTa-base-SQuAD 2 performed the best and is our deployed model for later use.

4.2 Similarity Search

Similarity search is a technique used in a wide range of applications, such as information retrieval, recommendation systems, image and audio search, and natural language processing. In our project, we applied similarity search to enable efficient and accurate search through a large corpus of text paragraphs.

The first step in our approach was to generate vector representations of the paragraphs using an embedding technique. Embedding techniques are widely used in natural language processing to represent words or sentences as dense vectors in a high-dimensional space. These vector representations capture the semantic meaning of the text in a way that can be compared using mathematical operations. We used a sentence-transformer model to generate vector representations of our text paragraphs as an embedding. We then stored these embeddings in Elastic Search, a highly scalable search engine that allows for fast and efficient search operations.

When a user enters a query, we first encode it as a vector using the same embedding technique used for the paragraphs. We then search through all the stored paragraph vectors using the cosine similarity metric. Cosine similarity is a measure of the similarity between two vectors that ranges from -1 to 1. A score of 1 indicates that the vectors are identical, while a score of -1 indicates that they are completely dissimilar. A score of 0 indicates that the vectors are orthogonal, i.e., they have no relation to each other.

We retrieve the paragraphs with the highest similarity scores and use them as input for the question-answering model (see section 4.1). By using similarity search, we can efficiently search through a large corpus of paragraphs and retrieve the most relevant ones in a matter of seconds.

Overall, the use of similarity search allowed us to build a highly accurate and efficient search engine that can retrieve the most relevant paragraphs to a user's query in real-time. By using vector representations of the paragraphs and cosine similarity as the search metric, we were able to efficiently search through a large corpus of text and retrieve the most relevant paragraphs for the user's query.

4.3 User Interface and API

For the User Interface and the REST API we decided to use Django, a Python web framework for building web applications. Django alleviates the burden of writing repetitive and standardized code from developers by promptly generating a functional web application programming interface (API) that comprises popular features, such as user management, thereby streamlining the development process. Since we only use the API to receive and answer questions from the user, we did not include any functionality to manage and operate the app. This left us with the two REST functions GET and POST.

Method: GET Retrieves a list of all products, grouped by their respective manufacturers, using a pre-configured factory meta-client. Returns the data in JSON format.

Method: POST Accepts a request with a set of parameters—manufacturer, product, language, and question—to instantiate a `QuestionAnswerer` object. If valid, the object processes the question and returns the answers in JSON format.
The user interface has been kept minimal in order not to distract from the actual function and to make it self-explanatory. In addition to the input field for the question, the user can enter all the information he has (manufacturer, product name) via various dropdowns to further narrow down his results. The results are displayed after a short period of time below the input fields.

5 Conclusion

The aim of this project was to build a question-answering system for coffee machines. In a first step, the manuals were scraped from their respective manufacturers website. In a second step, the content of the manuals was preprocessed into structured data, which made it easier to work with. The structured data then provided the basis for the question-answering system. A user can ask specific questions regarding his coffee machine and our system will extract suitable passages from said data basis. To reduce computational time and accuracy of the question-answering system, we find only the most relevant passages through similarity search. The results of the system demonstrated its ability to provide accurate and relevant answers to user queries.

The system's success highlights the value of NLP in the development of knowledge-based systems, particularly in the field of machine maintenance. The development of such systems could greatly improve the overall user experience, reducing the time and effort required to find information, and improving the efficiency of servicing and maintenance.

In future iterations, the system's capabilities could extend beyond merely extracting specific phrases from paragraphs to answer queries. Leveraging recent advancements in generative AI, we could facilitate the creation of comprehensive and grammatically accurate responses. Furthermore, to enhance efficiency and reduce processing time, we aim to implement a caching procedure that will store pairs of user questions and the model's responses. It's also worth noting that, due to time limitations, the custom built datasets we utilized were relatively small and leave room for improvement.

Overall, this project provides a proof-of-concept for the development of a practical and effective question-answering system for coffee machines, and demonstrates the potential for NLP-based solutions in the field of appliance maintenance.

Acknowledgement: Special thanks to our project supervisor Dr. *Sebastian Furth*.

References

[1] Hannah Bast and Claudius Korzen. "A Benchmark and Evaluation for Text Extraction from PDF". In: *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL). Toronto, ON, Canada: IEEE, June 2017, pp. 1–10.

[2] Marius Benkert, Jochen Schmidt, and Lennard Rosse. *Question Answering System for Coffee Machines*. https://github.com/LennardRose/Coffee_Corpus_QnA. Accessed on: Sep. 26, 2023. 2022-2023.

[3] *DB-Engines ranking of search engines*. https://db-engines.com/en/ranking/search+engine. Accessed: 2023-02-08.

[4] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[5] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[6] *ElasticSearch documentation*. https://www.elastic.co/guide/index.html. Accessed: 2023-02-08.

[7] Ross Girshick. *Fast R-CNN*. 2015.

[8] *Hadoop documentation*. https://cwiki.apache.org/confluence/display/HADOOP2. Accessed: 2023-02-08.

[9] Shashank Mohan Jain. "Hugging Face". In: *Introduction to Transformers for NLP: With the Hugging Face Library and Models to Solve Problems*. Springer, 2022, pp. 51–67.

[10] Kenton Lee et al. "Learning recurrent span representations for extractive question answering". In: *arXiv preprint arXiv:1611.01436* (2016).

[11] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[12] Rafael Padilla et al. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit". In: *Electronics* 10.3 (3 Jan. 2021), p. 279.

[13] Kate Pearce et al. "A comparative study of transformer-based language models on extractive question answering". In: *arXiv preprint arXiv:2110.03142* (2021).

[14] Sujay Raghavendra and Sujay Raghavendra. "Introduction to selenium". In: *Python Testing with Selenium: Learn to Implement Different Testing Techniques Using the Selenium WebDriver* (2021), pp. 1–14.

[15] Pranav Rajpurkar, Robin Jia, and Percy Liang. "Know what you don't know: Unanswerable questions for SQuAD". In: *arXiv preprint arXiv:1806.03822* (2018).

[16] Pranav Rajpurkar et al. “Squad: 100,000+ questions for machine comprehension of text”. In: *arXiv preprint arXiv:1606.05250* (2016).

[17] Leonard Richardson. *Beautiful soup documentation.* 2007.

[18] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv preprint arXiv:1910.01108* (2019).

[19] *Scripts for Training Layout Detection Models Using Detectron2.* `https://github.com/Layout-Parser/layout-model-training`. Accessed on: Feb. 7, 2023. layout-parser, Feb. 2, 2023.

[20] Zejiang Shen et al. “LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis”. In: *arXiv preprint arXiv:2103.15348* (2021).

[21] Maxim Tkachenko et al. *Label Studio: Data labeling software.* `https://github.com/heartexlabs/label-studio`. Accessed on: Feb. 7, 2023. 2020-2022.

[22] *Tutorial — PyMuPDF 1.21.1 Documentation.* `https://pymupdf.readthedocs.io/en/latest/tutorial.html`. Accessed on: Feb. 7, 2023.

[23] Yuxin Wu et al. *Detectron2.* `https://github.com/facebookresearch/detectron2`. Accessed on: Feb. 7, 2023. 2019.

Marius Benkert
Technical University Würzburg-Schweinfurt
CAIRO
97082 Würzburg
Germany
E-mail:
marius.benkert@study.thws.de

Jochen Schmidt
Technical University Würzburg-Schweinfurt
CAIRO
97082 Würzburg
Germany
E-mail:
jochen.schmidt.1@study.thws.de

Lennard Rose
Technical University Würzburg-Schweinfurt
CAIRO
97082 Würzburg
Germany
E-mail:
lennard.rose@study.thws.de