UPPSALA
UNIVERSITET

Självständigt arbete i informationsteknologi
8 juni 2020

Surmize: An Online NLP System for Close-Domain Question-Answering and Summarization

Alexander Bergkvist,
Nils Hedberg,
Sebastian Rollino,
Markus Sagen

Civilingenjörsprogrammet i informationsteknologi

Master Programme in Computer and Information Engineering

UPPSALA
UNIVERSITET

Institutionen för informationsteknologi

Besöksadress:
ITC, Polacksbacken
Lägerhyddsvägen 2

Postadress:
Box 337
751 05 Uppsala

Hemsida:
https://www.it.uu.se

Abstract

Surmize: An Online NLP System for Close-Domain Question-Answering and Summarization

Alexander Bergkvist,
Nils Hedberg,
Sebastian Rollino,
Markus Sagen

The amount of data available and consumed by people globally is growing. To reduce mental fatigue and increase the general ability to gain insight into complex texts or documents, we have developed an application to aid in this task. The application allows users to upload documents and ask domain-specific questions about them using our web application. A summarized version of each document is presented to the user, which could further facilitate their understanding of the document and guide them towards what types of questions could be relevant to ask. Our application allows users flexibility with the types of documents that can be processed, it is publicly available, stores no user data, and uses state-of-the-art models for its summaries and answers. The result is an application that yields near human-level intuition for answering questions in certain isolated cases, such as Wikipedia and news articles, as well as some scientific texts. The application shows a decrease in reliability and its prediction as to the complexity of the subject, the number of words in the document, and grammatical inconsistency in the questions increases. These are all aspects that can be improved further if used in production.

Extern handledare: Roland Hostettler, Uppsala Universitet
Handledare: Mats Daniels, Dilushi Piumwardane, Björn Victor och Tina Vrieler
Examinator: Björn Victor

Sammanfattning

Mängden data som är tillgänglig och konsumeras av människor växer globalt. För att minska den mentala trötthet och öka den allmänna förmågan att få insikt i komplexa, massiva texter eller dokument, har vi utvecklat en applikation för att bistå i de uppgifterna. Applikationen tillåter användare att ladda upp dokument och fråga kontextspecifika frågor via vår webbapplikation. En sammanfattad version av varje dokument presenteras till användaren, vilket kan ytterligare förenkla förståelsen av ett dokument och vägleda dem mot vad som kan vara relevanta frågor att ställa.
Vår applikation ger användare möjligheten att behandla olika typer av dokument, är tillgänglig för alla, sparar ingen personlig data, och använder de senaste modellerna inom språkbehandling för dess sammanfattningar och svar. Resultatet är en applikation som når en nära mänsklig intuition för vissa domäner och frågor, som exempelvis Wikipedia- och nyhetsartiklar, samt viss vetensaplig text. Noterade undantag för tillämpningen härrör från ämnets komplexitet, grammatiska korrekthet för frågorna och dokumentets längd. Dessa är områden som kan förbättras ytterligare om den används i produktionen.

Contents

List of Figures

1 Introduction

The amount of data available and consumed by people all over the world is growing, but our ability to process it is not. This coupled with that documents may be complex, as well as large in size, can make it hard to find information online. To help people who may or may not have a background in the field they are researching find information in documents, we have developed a web application to aid in this process. Our application aids users in interpreting electronic documents or texts based on the questions they ask about the document. While there are a few examples of tools to help with text analysis today, our application differentiates itself by bringing new research development in the field of text analysis into the general public's hands. This is done with an emphasis on it being free and user friendly.

Our application Surmize is a web application, designed to be used with either a computer or a smartphone. Surmize assists its user by first summarizing the user's uploaded documents. The user can then use these summaries as an aid to formulate questions to ask about the document's content. The questions are presented as a chat conversation, allowing the user to ask a question, get a reply, and then ask a new question. The questions are parsed through an algorithm on a remote server, that analyses the document for all possible relevant answers. The most relevant answer is sent back to the user, together with approximate confidence from the model. To run these algorithms, pre-trained NLP models from the company Huggingface were utilized.

To adhere to our emphasis on usability, a set of tests were devised to guide development. Features and design were driven by focus groups to help create an application that could be used easily and efficiently. Models and algorithms were chosen based on a scoring system, where reliability in both answers and summaries where tested. This was done to create an understanding of the limitations of the system. Speeds for the models were also measured to validate that the system performed its tasks, summation and question-answering, 5 to 25 times faster than the average human reader. This was important, as the goal of the project was to help make text comprehension faster for the user.

Based on these tests we conclude that all parts of our system were significantly faster than a human reader. The question answering mechanism performed very well (based on our scoring evaluation) and predictable for texts shorter than 3000 words. However, with longer texts it started to lose accuracy, losing track of details, and making significant mistakes. The summaries that were meant to help the user formulate questions also worked as intended, however, with the caveat that the summarization model exhibited unaccountable behavior when supplied with longer texts. The reasons for this behavior is as of now not known but hypothesized to be related to the model's earlier training

made by Huggingface. Generally, our focus groups indicated that the application was well-received as a tool. The actual usefulness provided to people tended to vary between documents. The reliability is still the most important concern for people as they would not use the application unless they believed the results to be reliable

To extend the application, a set of improvements were suggested one of which included running on better hardware. This meant upgrading our hosting service to some other service that had GPU clusters available. Having sufficient hardware for this type of application proved to be difficult without any form of funding for the project. Other improvements included retraining the models, possibly with new data, or using different model architectures. This was not done in the project due to the time frame not permitting that type of evaluation.

Acknowledgment
We the authors would like to thank our external stakeholder, assistant professor Roland Hostettler, for his valuable feedback, support, and belief in this project. Without him, this project would not have been possible. We would also like to thank professor Anders Hast for insightful conversations and general interest regarding the project.

Contact Details
Anyone interested in our work, results or source code[1] is welcome to contact us on one of the following e-mails addresses:

Markus Sagen:	Markus.John.Sagen@gmail.com
Sebastian Rollino:	Sebbeezk@gmail.com
Nils Hedberg:	Nils.Hedberg.1240@student.uu.se
Alexander Bergkvist:	Alexander.Bergkvist@hotmail.com

[1] https://github.com/MarkusSagen/Surmize

2 Background

In this section, we will detail the context and background surrounding the project. We will explain why computerized solutions to text processing are needed, how it was achieved historically and some of the present techniques. We will also elaborate on our external stakeholders and their respective roles.

2.1 The complexity of information in modern society

Many tools are used to navigate the Internet like search engines, links, or social media, which can provide the user with access to information about a wide range of subjects. There are complicated subjects however, like science, medicine, or political analyses, where relevant articles on the Internet might prove challenging for a user to read. If a text contains rare, difficult words and long complicated sentences it might be detrimental to its readability [51]. Readability is commonly defined as the ease with which a reader can understand written text and can be measured by many factors, most relevant to this project is the speed of reading, speed of perception and fatigue in reading [90].

There are quantitative measures of readability which give a comparative grade of a text's complexity, such as the Flesch-Kincaid grade level. The Flesch-Kincaid grade level has been used to indicate that the readability of scientific text was steadily decreasing between 1881 and 2015, and this trend seems to continue [74]. This makes scientific text less accessible for non-scientists over time and might increase the challenge for scientists trying to reproduce and understand each other's work [74].

Readability affects consumer information as well. For example, there are legal documents like the terms of service of various private companies that consumers are required to accept. These documents are typically long, so most consumers do not see it as worth their time to read them and understand what they agree to [60]. This opens the question of whether technology could help present this information in a way where fewer consumers will ignore the details of these agreements.

Google has used machine learning for natural language processing in language translation, with tools that have now become ubiquitous such as Google Translate [102]. This project investigates the possibility to utilize similar methods, but for reducing the complexity of the text to make information more easily accessible.

2.2 Deep Learning

Deep learning is a structured learning approach for automatically learning patterns from data. The learning approach is based on layered and interconnected *artificial neurons*,

inspired by the structure of the human brain [81]. Deep learning architectures such as Deep Neural Networks (DNNs) has since the 2010s proven to be "unreasonably" effective in solving a large range of tasks in AI, as stated by Terrence J. Sejnowski in the paper *The Unreasonable Effectiveness of Deep Learning in Artificial Intelligence* [82]. Deep learning is one of many approaches encapsulated in the study on machine learning.

2.3 Natural Language Processing

Natural Language Processing (NLP) investigates the use of computers to understand, interpret, and perform useful tasks on human (natural) languages [15, p. 1]. Since its inception in the 1950s, NLP has been an interdisciplinary study combining linguistics, computer science and artificial intelligence [50, 61, 12] to solve language tasks with computers. NLP has historically been considered a difficult area to solve [50, 15]. The reason why can best be summed up by a quote from philosopher Ludwig Wittgenstein "One cannot guess how a word functions. One has to look at its use and learn from that. The meaning of a word is its use in the language" [31, p. 173–179]. Before the late 2000s, domain knowledge experts in linguistics encoded semantic relationship, meaning, and sentiment for each language and task [50, p. 12–15]. However, since the advent of deep learning models outperforming previous NLP models, the recent progress in the field of NLP is strongly linked with that of deep learning research [50, 15, 104, 1].

Text data are considered in the context of machine learning as sequences or sequential data. What characterizes sequential data is that the order of the data matters. Changing the order of the words in a sentence changes its meaning. Several deep learning architectures exist that incorporates this property, namely Recurrent Neural Networks (RNNs) and, the most recent architecture, Transformers, which is described in more detail in Section 4.2.1).

2.3.1 Question-Answering

Question-Answering (QA) is a task in NLP, where the goal is to provide an answer to a question asked by a human, much like how a search engine operates [50, p. 118–120]. A question-answering model has a database of known information, called a knowledge base. Based on the question posed, the model retrieves all results matching the question and then returns the answer which best matched the question. In this regard, a question-answering model is related to that of a search engine or a chatbot. A QA model is defined as trying to solve one of two fields [50, p. 118–120]:

- Closed-domain QA (cdQA), which aims to answer questions posed within a certain domain (medicine, automotive, science) or based on context, such as from a set of documents.

- Open-domain QA (odQA), where the goal is to provide answers to nearly all questions posed. These models typically rely on a vast document corpus for answering questions.

2.3.2 Summarization

Automatic text summarization (SUS) is the process of taking a sequence of words and reducing the number of words, whilst retaining the most essential information from the original context[56]. The approach of summarization are divided into either *extraction-based* or *abstraction-based* [33]:

- In *extractive summarization* (ESUS) the aim is to summarize the content using only the words provided in the text.
- In *abstractive summarization* (ASUS), the model instead aims to learn the inherent language representation to make summarization more like how a human makes summaries, i.e. using their own choice of words.

Extractive summarization has historically been the more extensively researched of the two since it is considered a simpler problem to solve [50, p. 119–120][56].

2.4 External Stakeholder

Roland Hostettler is an assistant professor at the Department of Electrical Engineering at Uppsala University and specializes in modeling and inference for nonlinear dynamic systems. He serves as the main stakeholder for this project

Anders Hast is a professor at Uppsala University's division of visual information and interaction. Professor Hast sparked the initial idea of how simplifying data retrieval and inquiry on a large document corpus could be beneficial. He serves as the industry expert on this project.

3 Purpose, Aims, and Motivation

This project aims to minimize the reading workload of people interacting with texts. As defined by our stakeholder Roland Hostettler in Section 7.1, the application should allow users to quickly ascertain context-specific details from various types of documents and its essential information quickly and accurately.

3.1 Motivation

When attempting to read a complex text, an application such as the one being developed here could be an intermediary between the text and the user. In this application, users read a summarized version of a text from which they might discern the context, specific important details, and get a general idea of the content. From this information, the user might decide whether it is of sufficient interest to read further. Question-Answering can complement this by answering specific queries about the contents of a text or the domain of a larger corpus.

3.1.1 Sustainability

This project might work towards sustainability by creating an assisting tool in education and reducing the mental strain of knowledge work. Such a tool could be used in general education or specific domains like technical education and social science. This corresponds to the United Nations global initiatives for sustainability, specifically to Section 4.4 and 10.2 of the United Nations goals for sustainable development[93]. It is declared that Section 4.4 relates to the promotion of technical skills and Section 10.2 relates to the inclusivity of economic and political life.

3.2 Ethics

An ethical concern when working with machine learning is the problem of algorithmic bias. Algorithmic bias is where the algorithm wrongfully selects certain results over others in a discriminatory way [41]. This could arise in the NLP models used in this project because of biases present in the data on which the NLP machine learning models were trained [41]. The imperfections of the NLP models may cause some information about the content of a text to not be presented because the algorithm was biased against including it. This might become problematic if the content of a text is misrepresented in the summary.

Another ethical problem with the application is that it might encourage people to not read primary sources and gain full technical understanding. Instead, they might rather rely on a reduced, summarized version of the text, since the summarization algorithm cannot produce a perfect summary of its content.

3.2.1 Data Storage

In the current version, no account system is implemented. User information is never collected nor is any cookies of session tokens stored that could identify a user. Hence, the application has no liability under the General Data Protection Regulation (GDPR) [23].

When a user exits their session or refreshes the window, the files they have uploaded and the progress made in the application is removed.

3.3 Delimitations

The algorithms that produce text summary or answer a user's questions are very resource-intensive in terms of memory usage and processing power [37]. Achieving an application with quick response-time will not be the aim of this project, given the limited available computational resources. This project has no funding, hence no services will be bought and computational resources will not be expanded. Because of the aforementioned limitations, as well as the limited time budget, this project will also not involve the training of a new machine learning model. Consequently, the project is limited to using available open-source libraries that have pre-trained models. Depending on the content of the data set used in training, the models used might perform better on certain types of texts, and worse on others. Adjusting this would fall outside the scope of the project since it would require the training of new models.

4 Related Work

In this section, we explore related applications that we have found in our research, how they are similar to Surmize and in what ways our application differentiates itself. In the section related methods, we discuss research models in NLP which Surmize is based upon and the models which our application extends. Hardware and memory requirements are crucial aspects concerning if and how quickly our application will yield results, and this affects what technologies we could bring into the project.

4.1 Related Applications

Watson [30] is a QA/Conversational system developed by IBM. Watson was originally developed to answer questions on the popular TV-show Jeopardy but has since evolved to become a general-purpose QA machine utilized in many fields like economics, customer support, law, and healthcare. Relative to Surmize, Watson is a large system being quoted as to use more than 100 different techniques [8] to analyze natural language, identify sources, find and generate hypotheses, and more. To accomplish this at a fast pace for its thousands of simultaneous users, Watson utilizes the resources of the IBM Cloud, which allows the system to process data at high enough speeds [85]. For Surmize to take on similar, but scaled-down tasks, the system similarly needed to be scaled-down. In summary, the size, speed, and complexity of Watson are not achievable due to resource limitations. In our application, we could not use Watson directly since it uses proprietary components [86], which would not allow us to make modifications to

it. Another key consideration for not using Watson is because the software itself costs money, and a soft requirement for the project was for it to be publicly available and free. This project could potentially serve as an open-source alternative to Watson in the future.

Other large actors in the field of NLP are the AI research team at Google Brain [39]. In their paper on their recent work on PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization) [105], they explain how they achieved state-of-the-art performance on a large variety of topics, ranging from economy and news articles to medical documents. What was important with this work to us was the approach taken towards text summarization as abstractive, instead of extractive. This means that the system needs to understand the text it is being presented with, instead of selecting what seems to be informative fragments. Surmize is an attempt at bringing technology with this level of sophistication into an application that is intuitive for the general public to use. However, as with applications like Watson, our capacity to gather and process data are far more limited in comparison and should be taken into consideration.

4.2 Related Methods

In this section, we detail related methods and architectures in deep learning for solving specifically Question-Answering and Summarization. We will first list the underlying architectures and models used and then present how these models are commonly used for solving NLP tasks.

4.2.1 The Transformer model

One of the biggest problems with utilizing deep learning models in production is the time and resources it takes to train and use the models [50, 77]. These problems are further magnified in the field of natural language processing, where the training of these advanced deep learning models could not be trained in parallel. This is because the data is sent sequentially through the model, where each part of the models depends on the previous part to have evaluated the data; thus parallel training is not feasible [50, 11]. This creates two types of problems in the network architectures before the transformer architecture.

The first problem is, when sending data sequentially, each network in an RNN takes in the weights from all the previous neural networks in the sequence to update its weights [77], therefore, if the weights in one or more networks become large or close to zero, all subsequent models will get larger weights or weight approaching zero [37]. This means that the network will not learn and update its weights based on new data and

the model will learn poorly. This is commonly referred to as *The vanishing/exploding gradient problem* [11]. In NLP, the vanishing gradient problem typically occurred even in well-tuned models for word counts above 500-2000 words [50, 15].

The second problem is that training a deep learning model is both computationally-expensive and depending on the model and problem, can take hours, days, or weeks, even with the right hardware [73]. Therefore, many practitioners aim to extract large deep learning models into smaller parts and train them in parallel to reduce training time [73]. This, however, is not possible when working with RNNs [37], the models used in NLP before the transformer model. This is because the weights from all the previous nodes in the networks are used to update the weights for the current one and therefore. Each node in the RNN therefore needs all the previous nodes to have been trained before the current node can be trained.

To combat these two problems, the transformer neural network architecture was introduced in the paper *Attention Is All You Need* by Ashish Vaswani et. al., 2017 [96]. The three most prevalent improvements stated in the paper, which has made transformer models widely utilized are:

1. It allows the models to be trained in parallel
2. It allows learning entire sentences instead of sequences of words
3. The model learns to distinguish the words in a sentence based on its context. This is explained more in the following Section 4.2.1).

The company Huggingface [48] has provided a Python module of several pre-trained transformer models from several of the latest research papers. These transformer models are deep learning models aimed for general language comprehension that can be retrained to solve specific problems using transfer learning [37]. We have used some of the pre-trained NLP models provided by Huggingface, mainly a model called BERT (see below), and tailored it to solve our specific problems.

Bidirectional Encoder Representation from Transformers

BERT or Bidirectional Encoder Representation from Transformers is a deep learning model presented by the Google research team in 2018 [16]. Since its introduction, it has revolutionized the field of NLP surpassing all previous models up to that point in a wide variety of tasks [16, 2, 83]. BERT is based on and expands a new model architecture called a *Transformer*, which allows the model to learn entire sentences at a time instead of sequences of words. It also allows the network to learn how sentences and languages are constructed, based on the context of the surrounding words [83]. The transformer model, and subsequently BERT is based on an encoder-decoder structure instead of

a recurrent structure and uses a concept called *attention* [96]. Attention is a metric assigned to each word in a sentence. It represents for each prediction how important each word is and which words should be emphasized more than others [77, p. 613–618][58]. This allows transformer models such as BERT to learn the context of words and sentences based on the context and surrounding words.

5 Method and Implementation

In these sections, we describe what components our application are comprised of, what tools, frameworks, and languages were used, and why. The structure of our application and its interactions are further elaborated upon in Section 6.

The application is implemented as a client-server model. The illustration below depicts the different frameworks or libraries used for the different sections. For brevity, not all libraries used are depicted, such as for handling files and paths in Python. We will discuss what frameworks and libraries are used in the application and why in each section.

5.1 Client-Side / Front-end

As mentioned above, we decided to implement a web application that communicates with a server to retrieve data, such as requesting summaries and or answers to questions related to specific documents. In this subsection we discuss the client-side of the application, the frameworks used, and why they were chosen.

5.1.1 React

React is an open-source JavaScript library for building user interfaces developed by Facebook [78] and is currently maintained by Facebook, Instagram, and community developers [20]. React uses a component-based system to display views, what is rendered to the screen. The components are specified as custom HTML tags, which provide ease of use since they can be reused in different views, but also inside other components. React is also very efficient in updating the HTML document with new data. As the state changes, the content is re-rendered. [20, 6]. This allows React to display dynamic content on a web-page without making server-side requests or changes.

React is one of the most popular front-end web frameworks today [36]. Other very popular web frameworks are Angular and Vue [36], which are also open-source JavaScript frameworks. Angular is maintained by Google [38], and Vue is maintained by its creator and a smaller team [97]. All three are component-based, meaning that you build up

your front-end by putting together different components to create a final product. The frameworks named above have similarities but also differences, with the most important for us being the learning curve. Angular having the steepest learning curve and Vue the flattest [14]. Based on this we chose to not use Angular due to the statement above and the amount of time we had to work on this project. Even if Angular is very powerful, the time needed to get a front-end in a working state was not worth it for this project. For larger applications with a larger time frame, Angular would be the choice [17]. The decision between Vue and React was based on the group's preferences. Both frameworks have a lot of documentation and for further implementation, Vue would be a better choice, due to it being easier to integrate into existing projects [71]. Furthermore, Vue had the flattest learning curve and some of us already had previous experience with it. React however is very popular and there are many more jobs to React compared to Vue [42]. We also wanted to challenge ourselves, hence the choice of React over Vue. An example code of how components are constructed in React can be found in Appendix Section B.1.

5.1.2 CSS

CSS or Cascading Style Sheets is a markup language for HTML. It is used to style elements in the HTML. To style an HTML element, it needs to be targeted, which can be done in one of several ways. There are different ways of targeting HTML elements, by different selectors [98]. Some of these selectors are element selector, id selector, and class selector. When an element is selected, attributes such as shape, color, size, and more can be specified/altered. For an example of how this can be done, see in Appendix Section B.2.

At the beginning of the project, we considered using a CSS framework for the styling of our app. Out of Bulma, Semantic UI, Materialize CSS, Material UI, and Bootstrap we decided to use Bootstrap [7]. Bootstrap is the most popular CSS framework today [55]. It was chosen due to its popularity, styling of elements, and previous experience with the framework. After some deliberation, we decided to redesign our application, and bootstrap was no longer useful for how we wanted our app to look. Instead, we wrote our CSS for the appearance of our application.

5.2 Server-Side/ Back-end

In this section, we present the back-end framework we used, the alternatives to it, and how we communicate with it.

5.2.1 FastAPI

FastAPI is described as a modern, fast, web-framework for building APIs with Python [29]. It is a simple framework for defining web-end points to set up and make a request to. FastAPI claims that it is very fast, on par with Node.js and Go [29]. Even if it is relatively new, teams are starting to use it for projects, especially in projects related to machine learning, such as Explosion AI and SpaCy [29].
Other libraries that were brought up for discussion were Flask [32] and Node.js [70]. Flask is a web framework written in Python, with a lot of documentation [65]. Node.js is an open-source server environment, that allows the usage of running JavaScript on the server [70, 100]. When we began this project, we considered using either Node.js or a Python-based framework. Since the NLP models were written in Python, we reasoned that integrating the NLP models in one language for the server and logic would be easier than using two. Due to the models being written in Python, we decided to go for a Python framework and began using Flask. After some initial testing, we noted a slow response time compared to a Node-server, therefor, we switched to FastAPI. The difference in response time was noticeably 1.8 to 2.5 times faster. Another reason for choosing FastAPI, was its similarity to Node.js in the way of how files are structured and HTTP end-points defined, which was welcomed since many of us had previous experience working with Node.js. A final selling-point was its extensive and clear documentation.

5.2.2 Communication

Since we were implementing a web application we used HTTP for communication. To make an HTTP request from our front-end we used the built-in JavaScript function fetch [62]. It allows passing different headers and types of content, method, credentials, and other things that we did not use in this project. We chose to use JSON [99] objects to send data to and from the server and client since both Python and JavaScript support JSON-formation for sending and receiving data.

5.3 Natural Language Processing Model

Our application is designed for users who wish to get answers to specific questions they have for one or several documents. However, we quickly realized that this posed some other important problems we needed to solve:

1. How can a user know what types of questions might arise without first reading the document they want to ask questions about?

2. How can we help users to quickly gain insight about the document to inform them about what could be worth asking more about?

3. How can one ensure that a user asks questions in a structured way to give expected and reliable answers?

From this problem statement, we realized that our application needed to consist of two parts. The first part of the application is a **Question-Answering** model (QA), which allows the user to ask a question of a document and is given an answer to their question. The second part of the application is an abstractive **summarizing model**, which gives a summary of the document provided. By including a summarization model, we could provide the user with a summarized version of the document and its most important aspects. This might then help the user to quickly gain insight into the document and from it learn what could be useful to ask. The following sections will detail how the QA and summarization model is used in our application, as well as the methods.

5.3.1 Closed-Domain Question-Answering Model

In our application, a closed domain QA model built on BERT (see Section 4.2.1) is used for allowing the user to make questions and receive answers on specific documents of text. Given a text and a question, a BERT-based QA model assigns a probability to each word in the text for how likely each word is of being the beginning or ending word to the answer. The answer returned is all words between the word with the highest probability of being the starting word and the word with the highest probability of being the ending word [16].

Software and Frameworks

The cdQA-suite is an open-source bundle of cdQA applications developed by master students André Farias, Matyas Amrouche, Théo Nazon and Olivier Sans at Télécom ParisTech, in partnership with the Data LAB of BNP Paribas [25]. The main component in this bundle is their cdQA model implemented in Python, which in turn is based on an open-domain QA model (odQA) called DrQA[2], developed by Facebook Research [10]. In this project, this component from the cdQA-suite[3] was used, which uses pre-trained BERT models for closed-domain QA. This package was chosen after an evaluation of around 10 different QA models. This model proved the easiest to install and incorporate into our application, while also seemingly returning more accurate answers to the questions posed.

[2] https://github.com/facebookresearch/DrQA/

[3] https://github.com/cdqa-suite/cdQA

5.3.2 Summarize model

Initially, the goal of the project was to utilize only abstractive summarization. This was due to it being a more prominent problem to solve [50, p. 119–120][56], with the possible reward of making the summaries seem more intelligent and human-like. However, due to concerns regarding this technique's consistency and for improving the speed of summarization, an alternative algorithm using extractive summarization was also implemented. The user can then select which method to use, either an abstractive or an extractive summarization.

Software and Frameworks

As mentioned in related work, Section 4, all of our NLP models are based on pre-trained transformer models, provided by Huggingface. In addition to this, Huggingface also provides some models trained to solve specific examples[4]. One such model is a pre-trained BERT model for abstractive summarization, based on Yang Liu and Mirella Lapata's paper *Text Summarization with Pretrained Encoders*, 2019 [56]. We have modified and extended this model in our application to make abstractive summaries

We chose to extend Huggingface's implementation of a BERT abstractive summarization model, based on the testing we did on several other summarization models, where accuracy, time, and reliability of the summations were weighted into consideration. We also needed good documentation in order to be able to more easily modify the system. This is something that made Huggingface stand out, due to their whole business [68] revolving around providing institutions and people interested in NLP with trained models. The alterations we had to make, as explained in their documentation [47], was to provide certain parameters to make it work on our system. These parameters included turning off GPU support, as well as the level of abstractive interpretation in the text and reformating how the text was read. We refer to Huggingface's documentation [47] for more information on these parameters.

The extractive model is not a deep learning model, it is instead a graph-based algorithm called TextRank [63]. TextRank was implemented using the NLTK library [69] in Python, and GloVe [49] word representation vectors, which is a vector representation of English words. The algorithm transforms each sentence into a vector, utilizing GloVe's word representation. Each sentence is then compared to every other sentence in the document. The sentences that are the most similar to all other sentences are deemed to be the most important or informative and are therefore ranked higher. In order to create the final summary, one chooses as many of the top-ranked sentences making the summary as large as desired.

[4] https://github.com/huggingface/transformers/tree/master/examples/summarization/bertabs

Adding TextRank as an alternative in the application was, as mentioned previously, a decision made late into the project. This was due to concerns regarding the consistency and speed of the abstractive summarizer. Hence, we needed an alternative that was consistent and predictable. Due to their black-box nature [45], we decided not to use any deep learning algorithms for the extractive summarization. Instead, we tested a number of classical NLP algorithms, with TextRank being the most consistent at selecting informative parts in the texts.

5.4 Website and Server Hosting

The difficulty in hosting our application lied in the lack of funding for the project. A hosting solution would ideally be free, alternatively, come at a very low cost. This was problematic because of the resource consumption inherent in machine learning models [34]. Initially, using the project group's own hardware was discussed as the group owned a high-end consumer system. This coupled with fast internet speeds, likely would allow the website to handle a large number of requests at high speeds. However, when weighing performance versus other trade-offs, it was decided that a hosting service would be the better choice. One of these trade-offs was a desire for 24/7 availability since the member with the system might not be able to have it turned on at all times. Another concern would be security, as our own system would have no built-in protection against hackers or similar. Also, the possibility of a future expansion of the project would be more difficult. Because of all these concerns, we decided that hosting would be handled by a free hosting service.

Hosting of the website and back-end server was initially done on Heroku [44]. This was due to their free entry-level services, which suited our no cost goal. It was also easy to set-up, which was of the essence for a 6-week project. However, running pre-trained deep learning models was a lot more demanding in terms of memory and hardware on the server, we therefore switch to DigitalOcean [18]. Setting up the resources required on DigitalOcean required a minimal fee to successfully run the system. However, this was deemed acceptable since it allowed us the memory and computational power to run the entire application.

6 System Structure

In our application, we use the client-server model [35], where a client sends a request to add, get, or change data and the server responds to each request. In the following subsections, we describe what each part of the application does and how the different

parts communicate. Below is illustrated an overview of how communication between different parts of the system is made.

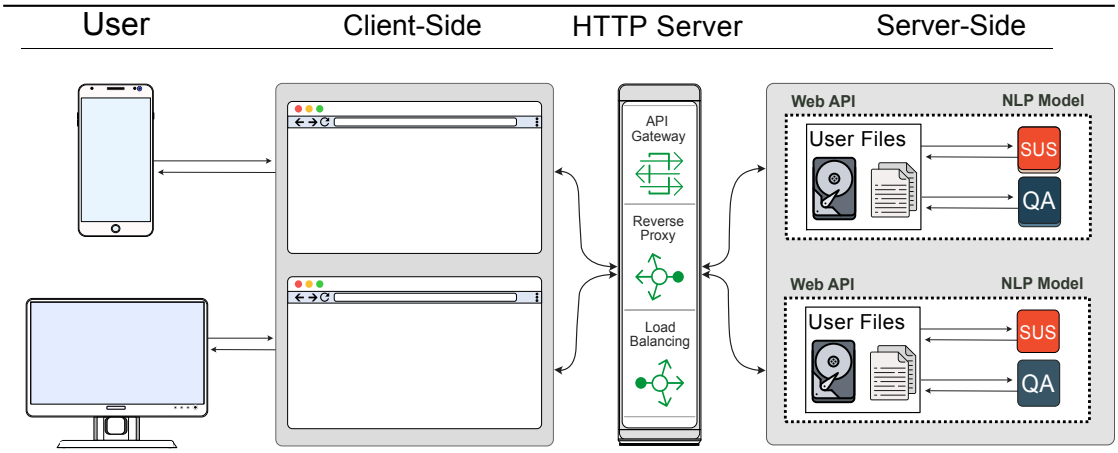


Figure 1 An overview of the system structure

6.1 User

The user can access the website with a web browser on a device of their choosing. The user can then upload files and interact with the system through the client-side view.

6.2 Client-Side

The client-side view is a single-page web application and is what the user interacts with. The client-side is implemented using the JavaScript framework React [78] for functionality and displaying dynamic content to the user. UI and UX are implemented using CSS. Communication with the server is achieved using HTTP requests.

6.3 HTTP Server

The HTTP server bridges the interaction and low-level logic between the client-side and the server-side. When users make requests from a web-browser, the HTTP server interprets each request and distributes them evenly to the different workers to handle each request. When a request is sent to and from the client-side and server-side, the HTTP server handles routing, ensuring that each user gets back the corresponding response to their request.

6.4 Server-Side

Our server serves as a web API [92], which was implemented with the Python web framework FastAPI [29]. The server receives different HTTP requests on different routes and processes the data and requests received. This could be a request to send the contents of a file, ask questions to the QA Model, or ask for a summary from the summarization model. When the data has been processed using the method specified by the user, the server sends back a JSON [99] object, which is rendered on the client-side in the user's browser. The HTTP requests used in this project are mainly GET and POST requests. Delete requests are only used when removing a file or exiting from a session in the application.

6.5 NLP Model and Data Processing

When the user uploads a document to the server, the document is converted and stored as a TXT file on the file system of where the web API is hosted. Each TXT file is then summarized sequentially using one of two summarization methods, which the user-specified when the files were first uploaded. The server communicates via sockets to inform the user when a summary is completed and available to view in their browser. A user can ask questions about a file even if the summary is not completed. The summaries and files are converted into TXT files and stored on the server. Then a user requests to view a file, that file and summary are sent as JSON objects to the client-side for each user, stored using a React state and displayed in the browser. This is done to ensure a perceived faster response time for each request and user.

7 Requirements and Evaluation Methods

In this section, we will discuss the requirements posed on the project. We will start introducing the initial requirements from our external stakeholder, in the following sections we then define metrics and methods for validating that these requirements are met.

7.1 Initial Requirements

Based on the interest of the external stakeholders Roland Hostettler and Anders Hast, see Section 2.4, the following requirements were stipulated based on their goal for the project.

1. The application should yield reliable and expected answers for each question. See Section 7.2

2. The application needs to be tested in regards to what type of texts it will and will

not be able to give reliable answers to. It should also be clear what type of content the application can handle. See Section 7.2

3. The application should inform users how to use the application to ensure predictable and reliable results. See Section 7.3

4. The application should if possible be optimized to give answers back quickly to the user. See Section 7.4

Both Roland and Anders have emphasized the importance of reliable and accurate answers for each question from the application, rather than speed.

7.2 Reliability

Evaluating the reliability or accuracy of NLP models can be a highly subjective task. What could be considered a good answer to a question, or a good summary might vary from person to person. One may think that grammar and fluency in the text is the most important factor, while others may settle for a poorer structure in the text as long as the content presented by the system is highly relevant. Having a reliable system is, however key if people are to use it, and therefore we need to be able to evaluate how reliable our system is.

7.2.1 Common Methods for Evaluating Machine Generated Texts

Because of the above-mentioned difficulties with assessing quality in a machine-generated text, a standardized metric for quality does not seem to exist for QA applications, nor summarization applications. We found that the preferred method of evaluation in this field is divided into two subcategories [43]. The first method is to compare the generated text to some human-generated reference text and then score the text on the similarity. How the similarities are handled can then differ based on implementation, but a benefit is that the comparison can be done by machines [54]. The second method is to have a human score the perceived quality of the machine-generated text.

Both these techniques suffer from the problem of subjectivity. The first method needs a reference text created by a human, and the second method has a human doing the comparison. However, we anticipated that the second method would give us a more nuanced evaluation. One example of such a benefit could be that we would avoid texts being negatively impacted if their summary or answers differed from ours but still had good content and structure. The trade-off of having a more nuanced evaluation is that it introduces even more subjectivity and a possibility of bias. This will be discussed in the following section.

7.2.2 Methodology Employed in this Project

Evaluating the NLP models in Surmize was done at two different stages. However, the methodology was kept consistent. The first set of tests was conducted in the research phase. At this phase, the goal was to select the most promising model for our QA and summary features. After removing models that were deemed too slow, only worked on drivers with dedicated GPU support, or failed to be installed, the models were tested using a small collection of ten texts. These texts were divided into two types: news articles and academic reports, each with varying degrees of difficulty in regards to subject and language. The models were then given a score between 0 to 5 based on its performance, with 5 being an almost perfect summary or answer and 0 meaning that the model returned an incomprehensible result.

The grading was done in groups to avoid bias and personal preference as much as possible. As mentioned in Section 5.3 these scores were then weighed into the decision of what models to use. The second time we evaluated our models was after their implementation into the application. This time we used twice as many texts, and including texts from books and blog posts. Each text received its score, from 0 to 5. This was recorded together with the length (in words) and perceived complexity (rated by us) of the text. We used this to benchmark the ability of our QA and summarization system, to see what kind of text lengths and topics the application could handle. These results can be found in Section 9.

7.3 Usability

To adhere to the requirements posed by our stakeholders regarding usability, features to aid the user with our application have been developed. How well these features work is evaluated using user tests with volunteers. The user tests will follow the layout presented by Mattias Arvola in his book ”Interaktionsdesign och UX” [3, p.134-139], as this methodology for user tests is the one we are most familiar with conducting. The participants will be asked to perform a set of actions related to our application, see the entire scripted user test in Appendix Section A. Their success will first be graded on a scale of 0 to 3, where 0 means the task was not complete and 3 that it was completed with high confidence by the user. These grades will then be investigated together with a set of parameters, these include gender, age, and self-rated technical-prowess. Having a diverse representation with these parameters should, according to Arvola [3, p.134-135], give a good indication of the guidance the design gives is sufficient. The minimum requirement on these tests is that all participants make it through the test without intervention from test conductors. For us to say the design is sufficient, most participants would have to make it through the test with fairly high confidence, grade 2 or 3. The users will also have the opportunity to reflect on their experiences after the

test has been concluded.

7.4 Speed

The speed of the system will be measured separately for the QA model and summary, as these components are anticipated to be responsible for most of the time consumption in the system. The time will be measured from when the request of action is sent until the result is returned to the user. Speed is not the most prioritized metric, due to the application running on limited hardware. To motivate the use of the system, it must at least be faster than a fairly proficient human reader. Therefore, we'll use the result presented in a 2019 study [9], to assume that an average reader can read at a speed of about 250 words per minute. For our system to be notably faster, we set the minimum requirement as interpreting 450 words per minute.

8 Detailed System Implementation

In this section, we describe in-depth the implementation of the various parts of our system. We will detail the datasets used, training methods, and similar for the deep learning NLP models. For each section, we will describe the implementation in detail. The figures below depict the different frameworks and libraries used to implement the different components of the system

Figure 2 Overview of the frameworks and libraries used in the application

The user interface is implemented entirely using CSS, ECMAScript 6 (JavaScript), and React. All parts of the application, with an exception for the client-side application and HTTP server, have been implemented in Python.

8.1 Client-Side

The client-side application was designed with usability, simplicity, and minimalism in mind. The following section describes a detailed implementation of the user interface and client-side application. For illustrations of the user interface and website as a whole, see Appendix Section C.

8.1.1 React

As mentioned earlier, our single-page application is built using React. Since React is a component-based framework, it allowed for quicker development. Project members could be working on different components without disturbing or slowing down the process. In total, we had 12 components for the web application. The reason for the small amount was that many of these components were reused. An example of this can be seen in the figure below, where the same buttons and boxes are used. Furthermore,

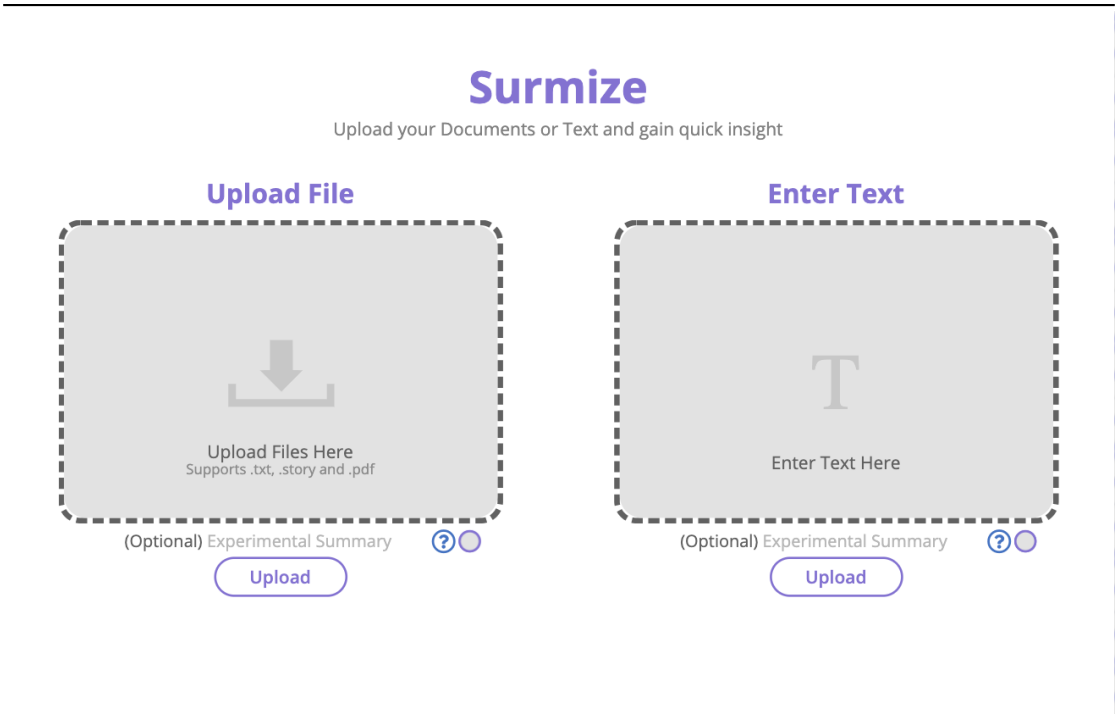


Figure 3 Landing page with reusable components such as buttons and boxes

we have two routes in our application. The root route "`/`" and the file managing route "`files/:id`", these are used to show our different views with different components. The routes are implemented using the library React-Router. The final feature of React is its state management, which enables dynamical and conditional rendering. This enables only portions of the website to be updated, without the need to request changes to the server.

8.1.2 UI

One of the requirements stated for the project was for the application to be intuitive to use. We, therefore, used designing principles to ease the user into the usage of the application. Inspiration for these design principles was taken from Google's Material Design

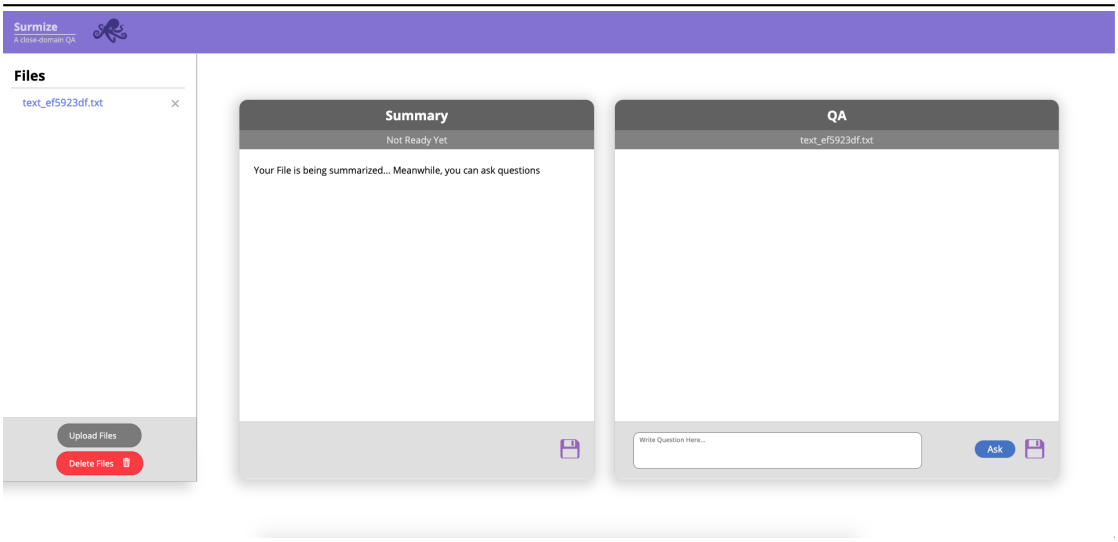


Figure 4 File Managing (workspace) view without Navbar links
Left most part depicts the files currently uploaded. The center of the figure depicts two windows. The first (left) depicts a summary of the text, the other (right) a dialogue system window, where users ask questions and receive answers to their questions

principles [46]. We wanted the user to quickly use the application, without extensive reading of how the application operates. We tried to minimize the number of written instructions on the page and instead guide users on how to use the application through the use of iconography and visual recall to related technologies. We used these principles, for example by changing the upload buttons to be grayed-out if there is nothing to upload, descriptive error messages are displayed to help users understand they are allowed or not to upload. We also changed the cursor when hovering over interactive elements in the browser to further help users. A limited number of primary colors have been used to reduce visual stimuli or fatigue, and emphasis has been put on utilizing surrounding white space. Whenever possible, we tried to rely upon and design certain parts of the application around technologies and systems the users were already familiar with. One such example is the QA system, which was designed to resemble a chat application. For a visual illustration of these design principles used in our application, see Appendix Section C.

8.2 Server-Side

The client-side and server-side is separated into two distinct parts. When a user makes a request in our system, it is made from the client-side to the server-side using HTTP or HTTPS, depending on if it is in development mode or not. Since traditional HTTP/Web servers do not understand or run Python code [59], an interface standard protocol such

as Web Server Gateway Interface (WSGI) or Asynchronous Server Gateway Interface (ASGI) [5, 4] is used when building web frameworks in Python. This allows developers to use dedicated HTTP Servers such as NGINX or Apache, while still writing server-side applications or web frameworks in Python [4]. This allows for the strength and speed of dedicated HTTP Servers, often built in a compiled language such as C or C++ [64], with the flexibility and simplicity of writing a web framework in Python.

When referring to the server-side as a whole, it is the combination of API Gateway HTTP server, ASGI/WSGI HTTP server, and Python web framework. The HTTP server is NGINX, the WSGI server is Gunicorn, the ASGI server is Uvicorn, and the web framework is FastAPI respectively. When developing on a local computer, NGINX and Gunicorn are not needed [27].

8.2.1 NGINX as a Web Server

NGINX is a dedicated web server, first released in 2004 by Igor Sysoev [64]. NGINX is free open-source software with many different capabilities and is one of, if not the most commonly used web server [101, 66]. NGINX is most commonly used as a web server, but and also be used as a load balancer, reverse proxy, mail proxy, and/or HTTP cache. NGINX also provides support for WSGI application servers. In our application, it is used as a web server, reverse proxy, and for serving static files.

8.2.2 Gunicorn as a WSGI Server

WSGI is a convention for web servers to forward requests to Python web frameworks. It was standardized 2010 as version 1.0.1 in PEP 3333 [19] and is composed of two components: a server/gateway component, often running against NGINX or Apache; and an application component, which is a Python web framework such as FastAPI, Flask, Django or similar [26]. Gunicorn is a WSGI HTTP Server, which provides automatic worker process management and configuration, where the recommended number of workers is set as $(2 \times \text{ Number of Cores}) + 1$ [40]. Gunicorn themselves rely on the webserver or operating system to provide all load balancing functionality when handling requests. When in production mode, this is assured by running behind a dedicated web server and using a reverse proxy, such as NGINX [67, 21, 88]

8.2.3 Uvicorn as an ASGI Server

ASGI is a succession of the WSGI standard but adds additional functionality from modern web development concepts, which was introduced in Python 3.6 [29]. These features include Async/Await and WebSocket support. In addition to this, ASGI is backward-compatible with WSGI. Since the ASGI protocol is relatively new, not many ASGI web servers exist yet, therefore it is common to translate the WSGI implementation into the

equivalent one using ASGI [4] and vice versa. This allows developers to write modern Python web syntax, but ensure that the code works on WSGI servers. Uvicorn is an ASGI server, built to utilize the asyncio frameworks from Python version 3.6+. According to its developers, it was designed to be comparable to Node.js and Go in terms of throughput in IO-bound contexts [95]. In addition to this, Uvicorn includes a Gunicorn worker class, which allows one to use Gunicorn for worker and process manager, but utilizing Uvicorn as an ASGI server, with Async/ Await support [94, 27]. Gunicorn is used in production mode to manage, restart, and dynamically increase or decrease the number of Uvicorn worker processes. In development mode, a single Uvicorn worker process is used without the use of either NGINX or Gunicorn.

8.2.4 FastAPI as a Web Framework

FastAPI is an ASGI web framework for building web APIs, that can leverage the features from Python 3.6+, such as asyncio. FastAPI claims to be one of the fastest Python-based web frameworks and on par with some Node.js and Go frameworks [29, 87]. FastAPI extends the data validation library Pydantic and the ASGI web framework Starelette. This inclusion means that FastAPI can rely on the performance, WebSocket, background process, and CORS support from Starelette, but also define and validate the data sent in and returned from the API end-point [28]. One key feature of FastAPI is its inclusion of OpenAPI and Swagger UI. OpenAPI specifies API creation, parameters, body requests, and more. Swagger UI is interactive API documentation that can be accessed directly in the browser and is based on the OpenAPI specification [28]. This allows for automatic documentation for each API end-point. It also allows for each end-point to be tested directly in the browser by visiting either the route "`/docs`" or "`/redoc`". The different routes use in our application and how they are rendered using Swagger UI can be seen in Appendix Section D.

8.2.5 How Requests are Handled in Surmize - In-Depth

When accessing the application from a dedication server (running in production mode), the communication is made using HTTPS instead of HTTP. HTTPS is provided to our application by the Certificate Authority (CA) Let's Encrypt [22, 91]. The interaction and communication between the different components are as follows, when in production mode:

1. The User accesses the Client-Side view via a browser and gets a random Session ID for that session or makes a request to our API.

2. The user makes a request. This request is sent to an API Gateway HTTP server along with the Session ID.

3. The *HTTP Server* handles each user request and distributes them correctly to the respective web end-point.
4. A *WSGI server* handles and interprets the requests into something Python interpretable.
5. The Gunicorn WSGI server launches several concurrent Uvicorn ASGI workers to handle the requests. This ensures that modern Async/Await is supported.
6. Each ASGI worker invokes the specific FastAPI web end-point for that request and verifies that the Session ID is of the correct format.
7. The web framework runs the code specified at that end-point, such as uploading a file or asking a question to a file.
8. Each file and its summary is temporarily stored for that Session ID on the server.
9. When a response is ready from the web framework, that worker returns it to the API Gateway / HTTP server as a JSON object, with a status code.
10. The response is finally redirected back to the correct user.
11. When a user closes or leaves a session, all uploaded files and data under that Session ID is removed from the server.

As described in the section above, we use a temporary Session ID to allow users access to the application. The reasoning for this was that we wanted to allow any user to use the application, but still separate users from each other. We also wanted to ensure that no one would be required to create an account before using the application. By using random temporary Session ID tokens, which could be validated, we found a good middle ground of not gathering data about the users, but still, allow them to easily use the application.

8.3 Question-Answering Model

The cdQA model extended in this project described in Section 5.3.1 requires the documents used to be in a Python Pandas Dataframe[89] format. Therefore, a Python class was implemented that handles file conversion from TXT, CSV and PDF files into a usable Data Frame. The Data Frame is loaded and initilized in the cdQA pipeline, which processes the document data and question asked [25]. It consists of two basic components, the retriever and the reader. Given a question, the retriever selects from a pool of available documents the paragraphs that are the most probable to contain the answer. These paragraphs are then passed on to the reader, which is a deep learning BERT model

as described in Section 4.2.1. The reader model is trained to pick out the sequence of words in the paragraph most likely to be a good answer. The reader model was trained using the Stanford Question Answering Dataset (SQuAD) version 1.1 [76]. Any biases in the model about what constitutes a good answer will stem from this dataset. Each answer is computed from each paragraph and given a score from the reader model. After each answer has been evaluated with a score, the answer returned is the one with the highest score. During the project, the model used in the reader was changed from BERT to *distillBERT*, a less memory intensive, faster version of BERT [80]. This substantially improved the speed of the application without noticeably compromising the answers the model yielded.

8.3.1 Confidence Score

A preliminary system for giving the user a confidence estimate of an answer was implemented using the score returned by the cdQA reader mentioned above. Depending on the size of the score an answer will be graded in 4 levels to approximate a measure of the answer's certainty. The score computed by the cdQA pipeline was not designed to be used in this way [24], but serves as a proof of concept.

8.4 Summarization Model

The abstractive summarization model (ASUS) is implemented in the application as a Python function, which can be imported. To use the model it must be supplied with the path to a folder containing the documents (target folder) that are to be summarized, as well as a destination folder (destination folder). The model only accepts documents with a *TXT* or *STORY* file extension, thus a check is always run before model initialization to verify this. Files with other file extensions will be converted if possible. The only files currently supported for conversion are *PDF* and *Story* files. Before initializing the model, a pre-processing step is executed to divide every sentence in the text with a newline character, this is done with the NLTK library [69]. This step is performed because the Huggingface model expects the data to be formatted with this convention.

When the model is initialized, the texts in the target folder are processed sequentially. Each sentence is transformed into a list of words, using the previously mentioned newline character. Each word in the sentence-list is transformed into a vector representation of that word, using an encoder. This process is referred to as word embedding [84], and it makes words and sentences simpler to understand for a machine learning model since vectors and numbers are more in line with how a machine represents information. The sequence of vectors is then passed through the network, where a mechanism referred to as attention is utilized. Attention and the transformer model is covered in Section 4.2.1.

Attention attempts to discern what parts of the sequence are most important. This is done by having the network produce a vector of the same length as our sentence vector with a number representing the importance of the word at the same location in the sentence vector. To create the actual summary, a new sequence is created by the second part of the model, using the parts deemed most important by the attention mechanism. Constructing the new sequence of word vectors is what gives the abstractive summarizer the ability to create summaries using its "own words". This is in contrast to the extractive method, where the model can select from the already existing sentences from the text it has read. The finished summaries are decoded from their vector representation into text, which is then saved in the destination directory.

9 Results and Discussion

In this section, we describe the results from the evaluation of our system concerning speed, reliability, and usability. These include metrics recorded, as well as the outcome of our user tests. For details regarding the methodology, see Section 7.

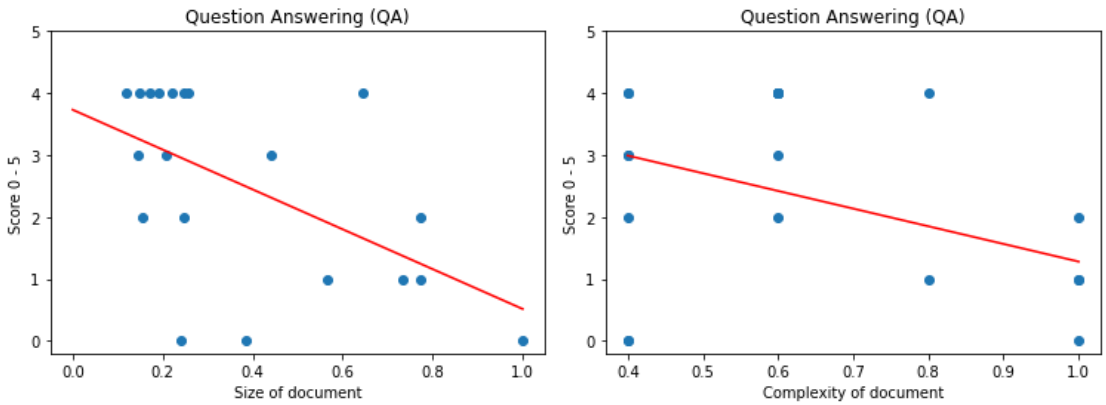
9.1 Model Performance and Reliability

Here we present the results from the evaluation and testing described in Section 7.2. Figure 5 shows the score of our QA model, between 0 and 5, concerning length of the document, as well as perceived complexity by the tester. Figure 6 shows the score of the abstractive summarizer, concerning length and complexity of text. A score of 0 means the result is incomprehensible or objectively wrong, a 5 is the result of a perfect summary or answer. The longest text was 5801 words, with all text lengths being given as a percentage of this. This means that a 0 represents 0 words, and the 1 represents 5801 words (the longest text). Note that multiple data points may have the same complexity and performance score in the Figures 5b, 6b, thus they will overlap in the graph. To resolve this and help visualize the pattern, a red line was fitted to the data using the Python function polyfit [13].

9.1.1 QA

The graphs in Figure 5 are representative of the capabilities that the QA model has exhibited during development. Quality of answer drastically drops when texts become larger. We found that at approximately 3000 words the model starts to produce inaccurate results. Complexity in the text and subject is also detrimental to answer quality, however not as severely. While these scores are subjective, they were carried out by a

minimum of two people, as discussed in Section 7.2. This was done to remove as much personal preference as possible, with the human resources available.



(a) Length of the document **(b)** Complexity of the document

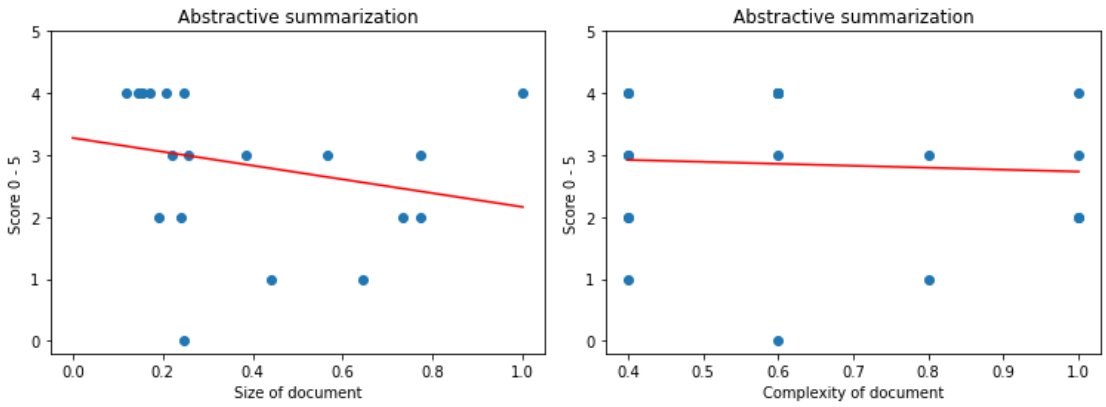
Figure 5 Evaluation of the QA model with respect to different parameters

9.1.2 Summary

The results in Figure 6 depicts the capabilities of our summarizer. According to Figure 6a, summaries perform fairly well with larger texts, and did not exhibit the performance drop that the QA model had. However, we believe this to be misleading. The reason for this to be false is that the summaries the model created when presented with longer texts, almost exclusively contained information from the earlier parts in the text. This seemed to occur somewhere around 1500-2000 words. The quality of the summary wasn't affected, but the content that would be included was almost exclusively taken from the first 1500-2000 words. Our hypothesis explaining this behavior is that when texts grow too large the model only retains the earlier parts, disregarding the latter. The reason for this might be that the model was trained with articles that rarely exceed 1500 words, and has therefore not been trained to use larger amounts of information. We know that it read the entire text, due to execution time increasing linearly as a function of the amount of words.

Figure 6b shows the score concerning complexity in the text. According to the graph, there is no significant relationship between the quality of the summary and the perceived complexity of the text. This is also suspected not to be accurate, since we encountered more problems when testing with academic reports than news articles during development. This does not mean that the scores are flawed or incorrect, instead, we believe these texts were a fortunate set of samples. Thus, a larger set of texts should have been used to evaluate the summarization model's performance. Please note that these scores are subjective, as mentioned in Section 9.1.1. However, we believe that working with

multiple group-members with testing made the scores more objective, and therefore more trustworthy.



(a) Length of the document **(b)** Complexity of the document

Figure 6 Evaluation of the abstractive summary with respect to different parameters

9.2 User Tests

Here we will present the results from the user tests described in Section 7.3. The users were asked to follow instructions from a script, which can be found in Appendix Section A. The progress they made with each tasks was graded with a score from 0 to 3, 0 meaning the tasks was not successful and 3 meaning it was successful with high confidence from the user. Firstly the scores will be discussed, as well as some other statistics. We will then highlight some important comments the participants had. Finally, we will discuss the implications these results had on our project.

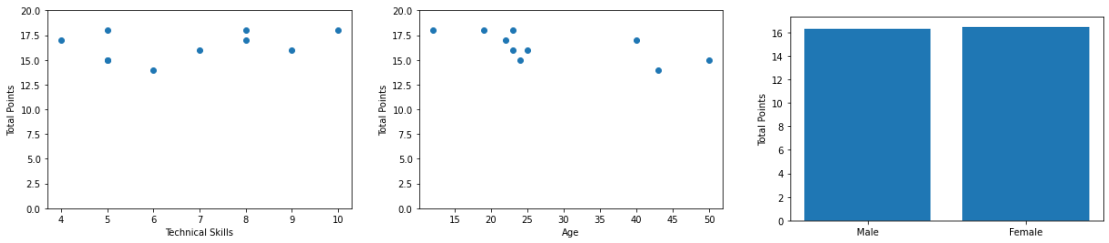
9.2.1 Metrics

In Figure 7 the results from the tests are visualized. The data collected in this test was: gender, age, technical skills and the 6 scores given for every task. Noteworthy is that most participants fared well during most parts of the test, while the first section that involved uploading documents proved challenging.

Gender	Age	Technical Skills	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Female	24	5	2	3	2	3	2	3
Male	50	5	1	3	2	3	3	3
Male	19	8	3	3	3	3	3	3
Male	25	7	2	3	3	3	2	3
Female	40	4	2	3	3	3	3	3
Female	12	5	3	3	3	3	3	3
Female	23	9	1	3	3	3	3	3
Male	23	10	3	3	3	3	3	3
Male	22	8	3	3	3	3	2	3
Male	43	6	1	2	3	3	3	2

Figure 7 User Test Results

In Figure 8, total score with respect to the three parameters: age, gender and technical skills are visualized. While these statistics are derived from a small sample group, we still believe they visualize that the emphasis on ease of use is working as intended. We also did not find any reason to believe the design to inhibit any particular groups of people from using the application.



(a) Technical Skills **(b)** Age **(c)** Gender

Figure 8 Score from Abstractive Summarization evaluation with respect to three different parameters

9.2.2 User Comments and Takeaways

Here we will present some frequent comments, that the users brought up and discuss the implications of these. The most frequent comments were:

- 6/10 people expressed concerns over the upload page, commenting that the design made it misleading where the correct buttons were located.
- 5/10 people expressed that they felt the summaries were poorly formulated, asking if the summaries could be better formulated grammatically.

- 3/10 people expressed that they thought the application misunderstood the text, when replying to questions.
- 6/10 people expressed that they did not think the application could be used in it's current form because of reliability issues.
- 10/10 people said that if the product would be approximately as trustworthy as a person, they could see one or multiple use cases for it.

As noted in the previous section, as well as in the users comments, the start page where one would upload the text and initialize the summary proved to be an obstacle. This was an opportunity to revise the design and add features such as an alternate cursor and intuitive highlights, to show the users where they were supposed to click.

The following comment describes the discomfort of trying to communicate with a machine that can not formulate good enough sentences. Out of all the participants, 50% expressed some form of concern that they had a hard time understanding the summaries from the application. This could potentially lead to misunderstandings, as well as a sense of distrust towards the application. While this problem cannot be addressed immediately, it would be interesting to see how this semantic transfer of information could be improved.

The following comments had less to do with design. What these comments demonstrated, was the underlying issue of trust. Some of the participants felt that the application did not do a good enough job of capturing the correct information. The group felt that reading the documents themselves would prove to be a better solution, combined with other tools such as built in document word search (Ctrl-F). The most notable takeaway from this result, is that the models must improve the quality and consistency of the information that they present to the user. This is already the goal in NLP research, thus we will not discuss it further.

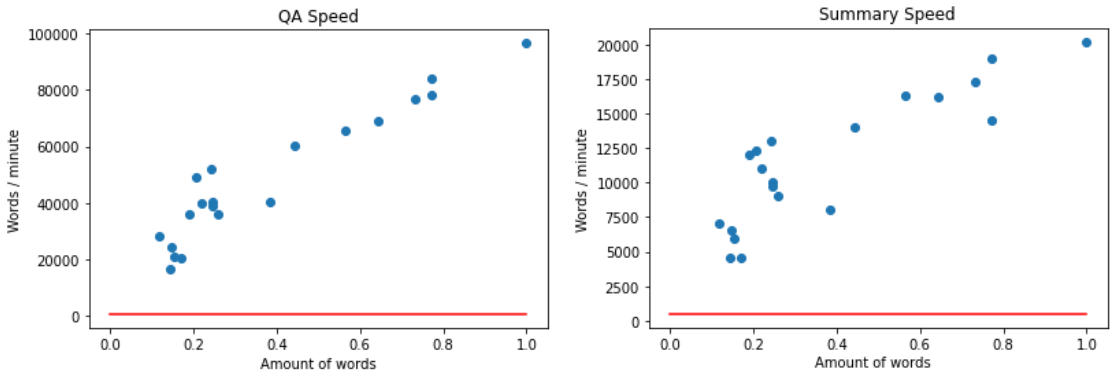
What we would be interested in, is to see how the methodology for presenting information to the user could be improved. Some of the distrust people feel towards the current application, may lie in the way it presents information. There might be a better way for the application to communicate, possibly giving insight into its thought process or similar. To achieve the level of trust required for people to consider using the application.

9.3 Application Speed

Here we present the recorded speeds of the system's models, for more details on this requirement, see Section 7.4. Figure 9a shows the speeds of the QA model. Figure 9b

shows the speed of the summarization model. Speeds are associated with the size of the processed document, and the speeds are measured in words/minute. This was done to make them comparable to the models required speed of at least 450 words/minute, see Section 7.4 for details. For reference, a red line representing the minimum requirement of 450 words/minute was included. As mentioned in Section 9.1, the size of the document is recorded in words and shown as a percentage of the largest text, which was 5801 words. This means that a 0 in the graph represents 0 words and 1 represents 5801 words.

Compared to their references, both models execute their respective tasks much faster than anticipated. These results also seemed to scale very well, since execution time increased slower than the size of the workload. This means that a large portion of the execution time lies in the model's initialization and that these models would be highly viable for larger documents. Provided the accuracy did not decrease with size, which it does at least in the QA model's case as demonstrated in Section 9.1.1. The summarization model's result was unexpected, and may also support the hypothesis presented in Section 9.1.2. The expectation was that the workload would grow exponentially with longer texts, as the entire texts needs to be processed together to form the summary. However, if the model only processes the early parts thoroughly when the text grows too big, then this would give the impression that the model became more efficient with longer texts.



(a) Question-Answering model **(b)** Abstractive summarization model

Figure 9 Measured speed with respect for the two NLP models. The red line is the minimum required speed in words/minute for each model

10 Conclusions

In this project we have shown how an online tool can help people with digital text comprehension. We made this contribution to test whether or not such a tool can be

accomplished with currently existing technologies, and limited hardware. We have also demonstrated what difficulties lie ahead in producing this as a real world application.

The application managed to solve basic problems with short to medium sized texts. Longer texts proved to yield inconsistent results. The application also proved to be reasonably reliable when processing texts with simpler content, while being inaccurate with more complex texts. The results presented was accomplished with minimal hardware resources and free to use libraries, frameworks and tools.

Test groups were presented with the application, with a mixed reception. Many users saw the potential use of the tool, but did not express trust in the current implementation. The motivation many participants had was that the time gain would not compensate for the amount of mistakes made by the algorithm.

In summary, our application showed that solving the problems of text comprehension is indeed feasible, even for smaller systems. However, we have also seen that there needs to be a more sophisticated central model in place to deal with text analysis. We also need a better way to communicate the decision process of the machine to its users.

11 Future Work

While the application serves as an optimistic prototype, there are a couple of improvements that would be necessary to make this a usable application in the real world. In this section, we outline these steps as a proposal for future work.

11.1 GPU Support

The first step towards making the application more usable as a product is changing the hardware it runs on. Instead of simply adding more CPU cores, moving the application to a hosting service with dedicated CUDA enabled GPU support would make the NLP models much faster [72]. It would also potentially enable the usage of other NLP models or implementations that were deemed to slow or had too many parameters to be useful in this project. Variations and successors to the BERT transformer would have been especially interesting to compare. Models such as ALBERT [52], RoBERTa [57], BART [53], XLNet [103], and others [75] would have been interesting to compare against our current model implementation.

11.2 Re-train the Models

Another possible improvement could be to manually re-train the models currently used in the application. For instance, one such improvement could be to train the summarizer to yield longer summaries (see Section 9.1). We noted that the length of the document negatively effected how well the summaries and QA performed. This could potentially be solved in part by re-training on larger text with better hardware. We also considered training the models to separate the texts into smaller parts, analyze each part, and cleverly combine those into one text for its predictions. We also noted that the grammatical correctness of the questions confused the QA system's predictions. We believe that a possible solution to this could be to re-train the models with questions posed with spelling errors or grammatical errors, to make the models more robust. We also considered to include pre-trained models[5],[6] or re-train the summarizer and QA model to work better on more diverse fields, such as on medical data[7,8,9], legal documents[10], scientific text[11,12], etc. This would potentially make the models better on more diverse types of documents and improve their overall performance. These aspects were not explored in the project due to time constraints.

11.3 Better Metrics

As of now, we are using our own evaluation for how well the models perform on different tasks. Moving forward, we would need to utilize standardized metrics for evaluating how well these models perform. Additionally, the QA system's confidence score is based on how many potential answers it finds to a question. We noted that the confidence score sometimes corresponds well to the provided answers. Other times it is very confident, but the answers clearly distinguished as incorrect. A better solution to the confidence score would need to be implemented. A possible solution to this is to train a separate model on the confidence score the model predicts. In that way, we could provide an additional model which predicts the confidence, based on its initial score and adjust its confidence.

[5] https://github.com/dmis-lab/biobert
[6] https://github.com/georgwiese/biomedical-qa
[7] https://github.com/zhangsheng93/cMedQA
[8] https://github.com/abachaa/MedQuAD
[9] https://github.com/abachaa/VQA-Med-2019
[10] https://github.com/siatnlp/LegalQA
[11] https://github.com/allenai/scibert
[12] https://github.com/allenai/scifact

11.4 User Features and Security

As of now, users are provided with an unique Session ID Token each time they are using the application. We thought about adding user login as an optional feature, where users could store previous session activities and documents. This would, require the inclusion of a authentication system, such as OAuth2 and increased vulnerability for hackers. Additionally, we would need to provide a secure data storage solution for users, sessions, and data. We are not sure if this is a needed feature, but have been discussed as a potential future inclusion. We would also like to investigate more in-depth the types of vulnerabilities of the application. We also discussed the ability to allow user to ask a batch of questions to multiple documents. This was not implemented due to time constraint and the increased server-workload.

11.5 Increase the Number of Supported File Formats

The system in its current state supports the file extensions *TXT*, *PDF* and *STORY*. For future improvements, we would like to extend the capabilities of the file formats and types of documents that can be parsed, such as *DOC*, *DOCX*, *ODT*, *MD*, and *CSV*. We also considered support for URLs[13], which could parse the text content from a website. Furthermore, the ability to include and parse the content from an image, such as the text from an image (OCR[14,15]) and/or parsing image obejects to text (VQA[16]) was discussed. We decided not to implement these features, due to time constraints and the increased model complexity.

11.6 Dedicated Mobile Application

We reasoned that users might use the application on a mobile phone and therefore a mobile version could be useful. As of now, the client-side view is rendered in a browser, which is designed to work both on mobiles, tablets, and computers. Moving forward, we could potentially improve support for tablet and mobiles by implementing the client-side view as a Progressive Web App (PWA) or a dedicated native app using React Native. We considered this because of many smart phone user's prevalence to prefer using a dedicated mobile app and the the ability to send notifications, which could notify a user when a summary is completed.

[13] https://github.com/jjangsangy/ExplainToMe
[14] https://github.com/gasparian/CRNN_OCR_lite
[15] https://github.com/fengxinjie/Transformer-OCR
[16] https://github.com/abhshkdz/neural-vqa-attention

References

[1] B. Alshemali and J. Kalita, "Improving the reliability of deep neural networks in NLP: A review," *Knowledge-Based Systems*, vol. 191, 2020.

[2] D. Anderson, "A deep dive into BERT: How BERT launched a rocket into natural language understanding," https://searchengineland.com/a-deep-dive-into-bert-how-bert-launched-a-rocket-into-natural-language-understanding-324522, Search Engine Land, Nov. 2019, accessed 2020-04-17.

[3] M. Arvola, *Interaktionsdesign och UX: om att skapa en god användarupplevelse*. Studentlitteratur AB, 2014.

[4] ASGI Team, "ASGI (Asynchronous Server Gateway Interface)," https://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/deployment/asgi.html, Pyramid, accessed 09 May 2020.

[5] ASGI Team, "ASGI Read the Docs: Introduction," https://asgi.readthedocs.io/en/latest/introduction.html, ASGI Team, accessed 09 May 2020.

[6] E. Atto, "Understanding the fundamentals of state in React," https://medium.com/the-andela-way/understanding-the-fundamentals-of-state-in-react-79c711be677f, Medium, accessed 24 April 2020.

[7] Bootstrap Team, "Home Page," https://getbootstrap.com, Bootstrap Team, accessed 09 April 2020.

[8] E. Brown, E. Epstein, J. W. Murdock, and Tong-Haing, "IBM Research Report," *IBM Research Report RC25356*, 2013.

[9] M. Brysbaert, "How many words do we read per minute? A review and meta-analysis of reading rate," *Journal of Memory and Language*, vol. 109, 2019.

[10] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading Wikipedia to Answer Open-Domain Questions," *CoRR*, vol. abs/1704.00051, 2017. [Online]. Available: http://arxiv.org/abs/1704.00051

[11] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018, OCLC: ocn982650571.

[12] N. Chomsky and D. Lightfoot, *Syntactic structures*. Walter de Gruyter, 2002.

[13] T. S. community, "Numpy Polyfit," https://numpy.org/doc/1.18/reference/generated/numpy.polyfit.html, The SciPy community, accessed 13 May 2020.

[14] S. Daityari, "Angular vs Vue vs React," https://www.codeinwp.com/blog/angular-vs-vue-vs-react/, Codeinwp, accessed 24 April 2020.

[15] L. Deng and Y. Liu, *Deep learning in natural language processing*. Springer, 2018.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[17] H. Dhaduk, "5 popular JavaScript frameworks, when to use," https://dev.to/hirendhaduk_/5-popular-javascript-frameworks-when-to-use-46le, dev.to, accessed 24 April 2020.

[18] DigitalOcean, "Home Page," DigitalOcean, DigitalOcean, accessed 03 April 2020.

[19] P. J. Eby, "PEP 3333 – Python Web Server Gateway Interface v1.0.1," https://www.python.org/dev/peps/pep-3333, Python Software Foundation, accessed 10 May 2020.

[20] L. Education-Ecosystem, "React History," https://www.education-ecosystem.com/guides/programming/react-js/history, Ledu Education-Ecosystem, accessed 24 April 2020.

[21] J. Ellingwood and K. Juell, "How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 18.04," https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04, Digital Ocean, accessed 09 April 2020.

[22] L. Encrypt, "Getting Started," https://letsencrypt.org/getting-started, Let's Encrypt, accessed 09 April 2020.

[23] "Cookies, the GDPR, and the ePrivacy Directive," https://gdpr.eu/cookies/, European Union, accessed 2020-04-28.

[24] A. Farias, "Return confidence probability of the answer," https://github.com/cdqa-suite/cdQA/issues/195/, july 2019, accessed 2020-05-08.

[25] A. M. Farias, "How to create your own Question-Answering system easily with Python," https://towardsdatascience.com/how-to-create-your-own-question-answering-system-easily-with-python-2ef8abc8eb5, Towards Datascience, july 2019, accessed 2020-04-20.

[26] FastAPI, "Alternatives, Inspiration and Comparisons," https://fastapi.tiangolo.com/alternatives, FastAPI, accessed 09 April 2020.

[27] FastAPI, "Deployment," https://fastapi.tiangolo.com/deployment, FastAPI, accessed 09 April 2020.

[28] FastAPI, "Features," https://fastapi.tiangolo.com/features, FastAPI, accessed 09 April 2020.

[29] FastAPI, "Home Page," https://fastapi.tiangolo.com, FastAPI, accessed 09 April 2020.

[30] D. Ferrucci, A. Levas, S. Bagchi, D. Gondek, and E. Mueller, "Watson: Beyond Jeopardy!" *Artificial Intelligence*, pp. 93–105, 2017.

[31] J. Findlay, "Philosophical Investigations," *Philosophy*, vol. 30, no. 113, pp. 173–179, 1955.

[32] Flask, "User's Guide," https://flask.palletsprojects.com, Flask, accessed 24 April 2020.

[33] M. Garbade, "A Quick Introduction to Text Summarization in Machine Learning," https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f, Towards Datascience, accessed 2020-04-12.

[34] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75 – 88, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731518308773

[35] Geeks For Geeks, "Client-Server model," https://www.geeksforgeeks.org/client-server-model/, Geeks For Geeks, accessed 03 April 2020.

[36] A. Goel, "10 Best Web Development Frameworks," https://hackr.io/blog/top-10-web-development-frameworks-in-2020, hackr.io, accessed 09 April 2020.

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[38] Google, "Angular," https://opensource.google/projects/angular, Google, accessed 24 April 2020.

[39] Google Brain Team, "About the Team," https://research.google/teams/brain/, Google Brain Team, accessed 09 April 2020.

[40] Gunicorn, "How Many Workers?" https://docs.gunicorn.org/en/stable/design.html#how-many-workers, Gunicorn, accessed 09 April 2020.

[41] S. Hajian, F. Bonchi, and C. Castillo, "Algorithmic Bias: From Discrimination Discovery to Fairness-aware Data Mining," https://dl.acm.org/doi/pdf/10.1145/2939672.2945386, Eurecat, ISI Foundation, Aug. 2016, accessed 2020-04-28.

[42] M. Hamedani, "What JavaScript Framework You Should Learn to Get a Job in 2019?" https://programmingwithmosh.com/javascript/what-javascript-framework-you-should-learn-to-get-a-job-in-2019/, Programming With Mosh, accessed 8 May 2020.

[43] A. Han, D. Wong, L. Chao, and L. He, "Automatic machine translation evaluation with part-of-speech information," in *International Conference on Text, Speech and Dialogue*. Springer, 2013, pp. 121–128.

[44] Heroku, "Heroku is for Developers," https://www.heroku.com/developers, Heroku, accessed 30 Mars 2020.

[45] A. Holzinger, M. Plass, K. Holzinger, G. C. Crisan, C.-M. Pintea, and V. Palade, "A glass-box interactive machine learning approach for solving NP-hard problems with the human-in-the-loop," 2017.

[46] A. Hopkins, "Google's 9 Principles of Material Design," https://blog.prototypr.io/googles-9-principles-of-material-design-fb3fef64dcf, Prototyper.io, accessed 08 May 2020.

[47] Huggingface, "Huggingface Summary Documentation," https://github.com/huggingface/transformers/tree/master/examples/summarization/bertabs, Huggingface, accessed 28 April 2020.

[48] Huggingface, "Transformers," https://huggingface.co/transformers/, Huggingface, accessed 17 April 2020.

[49] C. M. Jeffrey Pennington, Richard Socher, "GloVe documentation," https://nlp.stanford.edu/projects/glove/, Stanford, accessed 28 April 2020.

[50] U. Kamath, J. Liu, and J. Whitaker, *Deep Learning for NLP and Speech Recognition*. Springer, 2019.

[51] G. Klare, "Assessing readability," *Reading research quarterly*, pp. 62–102, 1974.

[52] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERTA: A lite BERT for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[53] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[54] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of summaries," in *Proceedings of the ACL Workshop: Text Summarization Braches Out 2004*, Jan. 2004, p. 10, accessed 2020-04-10.

[55] T. Liu, "CSS Framework," https://www.mockplus.com/blog/post/css-framework, Mockplus, accessed 24 April 2020.

[56] Y. Liu, "Fine-tune BERT for extractive summarization," *arXiv preprint arXiv:1903.10318*, 2019.

[57] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *arXiv preprint arXiv:1907.11692*, 2019.

[58] K. Loginova, "Attention in NLP," https://medium.com/@joealato/attention-in-nlp-734c6fa9d983, Medium, june 2018, accessed 2020-04-17.

[59] M. Makai, "WSGI Servers," https://www.fullstackpython.com/wsgi-servers.html, Full Stack Python, accessed 10 May 2020.

[60] T. Maronick, "Do Consumers Read Terms of Service Agreements When Installing Software? A Two-Study Empirical Analysis," *International Journal of Business and Social Research*, 2014.

[61] M. Masterman, "Semantic message detection for machine translation, using an interlingua," in *Proc. 1961 International Conf. on Machine Translation*, 1961, pp. 438–475.

[62] MDN contributors, "Fetch API," https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API, Mozilla, accessed 11 April 2020.

[63] R. Mihalcea and P. Tarau, "TextRank: Bringing Order into Texts," in *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing*, July 2004.

[64] T. Mobily, "Interview with Igor Sysoev, author of Apache's competitor NGINX," http://fsmsh.com/3657, Free Software Magazine, accessed 09 April 2020.

[65] M. Nehra, "Top 10 best backend frameworks for web development in 2020," https://www.decipherzone.com/blog-detail/top-10-best-backend-frameworks-for-web-development-in-2020, Decipherzone, accessed 24 April 2020.

[66] Netcraft, "Web Server Survey," https://news.netcraft.com/archives/category/web-server-survey, Netcraft, accessed 09 April 2020.

[67] Nginx, "What Is a Reverse Proxy Server?" https://www.nginx.com/resources/glossary/reverse-proxy-server, Nginx, accessed 09 April 2020.

[68] NLP Zurich, "Co-founder Thomas Wolf on Huggingface," https://youtu.be/rEGB7-FlPRs?t=1399, Huggingface, accessed 29 April 2020.

[69] "NLTK documentation," https://www.nltk.org/, NLTK, accessed 28 April 2020.

[70] Node, "Home Page," https://nodejs.org/en/, Node, accessed 24 April 2020.

[71] M. Nowak, "Vue vs React 2020," https://www.monterail.com/blog/vue-vs-react-2020, monterail, accessed 26 April 2020.

[72] Nvidia, "Nvidia GPU acceleration," https://www.nvidia.com/en-us/deep-learning-ai/solutions/data-science/, Nvidia, accessed 7 May 2020.

[73] J. Patterson and A. Gibson, *Deep learning: A practitioner's approach*. O'Reilly Media, Inc., 2017.

[74] P. Plavén-Sigray, G. J. Matheson, B. C. Schiffler, and W. H. Thompson, "The readability of scientific texts is decreasing over time," https://elifesciences.org/articles/27725, Karolinska Institutet, Feb. 2017, accessed 2020-04-27.

[75] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained Models for Natural Language Processing: A Survey," *arXiv preprint arXiv:2003.08271*, 2020.

[76] P. Rajpurkar, "SQuAD," https://rajpurkar.github.io/SQuAD-explorer/, Stanford University, accessed 10 May 2020.

[77] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[78] React, "React | A JavaScript library for building user interfaces," https://reactjs.org, Facebook Open Source, accessed 09 April 2020.

[79] Rotten Tomatos, "Blade Runner (The Final Cut) (1982)," https://www.rottentomatoes.com/m/blade_runner_1982, Rotten Tomatoes, accessed 2020-04-12.

[80] V. Sanh, "Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT," https://medium.com/huggingface/distilbert-8cf3380435b5, Huggingface, accessed 10 May 2020.

[81] J. Schmidhuber, "Deep Learning," *Scholarpedia*, vol. 10, no. 11, 2015, revision #184887.

[82] T. J. Sejnowski, "The unreasonable effectiveness of deep learning in artificial intelligence," *Proceedings of the National Academy of Sciences*, 2020.

[83] Y. Seth, "Attention in NLP," https://yashuseth.blog/2019/06/12/bert-explained-faqs-understand-bert-working/, Blog, jun 2019, accessed 2020-04-14.

[84] K. Shuang, Z. Zhang, J. Loo, and S. Su, "Convolution–deconvolution word embedding: An end-to-end multi-prototype fusion embedding method for natural language processing," *Information Fusion*, vol. 53, pp. 112 – 122, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253518306171

[85] T. Suzor, "The defining moments of IBM Watson last decade," https://www.ibm.com/blogs/watson/2020/01/the-defining-moments-of-ibm-watson-last-decade, IBM, accessed 03 April 2020.

[86] T. Mudau (https://ai.stackexchange.com/users/7550/tshilidzi-mudau), "Is there any open source counterpart to the IBM Watson?" https://ai.stackexchange.com/questions/4219/is-there-any-open-source-counterpart-to-the-ibm-watson, AI Stackexchange, accessed 26 April 2020.

[87] TechEmpower, "Web Framework Benchmarks," https://www.techempower.com/benchmarks/#section=test&runid=7464e520-0dc2-473d-bd34-dbdfd7e85911&hw=ph&test=query&l=zijzen-7, TechEmpower, accessed 09 April 2020.

[88] O. Tezer, "How to Deploy Python WSGI Apps Using Gunicorn HTTP Server Behind Nginx," https://www.digitalocean.com/community/tutorials/how-to-deploy-python-wsgi-apps-using-gunicorn-http-server-behind-nginx, Digital Ocean, accessed 09 April 2020.

[89] The Pandas Development Team, "Pandas Dataframe Documentation," https://pandas.pydata.org/docs/reference/frame.html, The Pandas Development Team, accessed 2020-05-10.

[90] M. Tinker, *Legibility of print*. Iowa State University Press, 1963.

[91] B. Traversy, "Node.js Deployment," https://gist.github.com/bradtraversy/cd90d1ed3c462fe3bddd11bf8953a896, Brad Traversy, accessed 09 April 2020.

[92] Tutorailsteacher, "What is a Web API," https://www.tutorialsteacher.com/webapi/what-is-web-api, Tutorialsteacher, accessed 26 April 2020.

[93] United Nations Department of Public Information, "Sustainable Development Goals," https://sustainabledevelopment.un.org/?menu=1300, United Nations, accessed 20 April 2020.

[94] Uvicorn, "Deployment," https://www.uvicorn.org/deployment, Uvicorn, accessed 09 April 2020.

[95] Uvicorn, "Home Page," https://www.uvicorn.org, Uvicorn, accessed 09 April 2020.

[96] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[97] Vue, "Vue," https://vuejs.org/v2/guide/team.html, Vue, accessed 24 April 2020.

[98] W3School, "CSS Selectors," https://www.w3schools.com/css/css_selectors.asp, W3School, accessed 09 April 2020.

[99] W3Schools, "JS JSON intro," https://www.w3schools.com/js/js_json_intro.asp, W3Schools, accessed 24 April 2020.

[100] W3Schools, "Node.js Intro," https://www.w3schools.com/nodejs/nodejs_intro.asp, W3schools, accessed 24 April 2020.

[101] W3Techs, "Usage statistics of web servers," https://w3techs.com/technologies/overview/web_server, W3Techs, accessed 09 April 2020.

[102] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144

[103] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *Advances in neural information processing systems*, 2019, pp. 5754–5764.

[104] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligene Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[105] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu, "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization," 2019.

Appendix

A User Test Script

Kort undersökning innan vi börjar:

Kön:

Ålder:

Egen bedömd teknisk kompetens:

Testperson (alias går bra):

Testledare:

Förberedelse:

Sätt upp hemsidan (om den körs lokalt), eller ha en URL redo (om den körs online). Ladda ner textfilerna "UnionJack" och "BatmanLong", och lägg dom i någon mapp som är lättillgänglig.

Instruktioner till testledaren:

- Du får endast ge de instruktioner som finns här.
- Du får inte svara på följdfrågor gällande applikationen.
- Du ska så gott som möjligt vara neutral till testpersonens framsteg eller misslyckanden.
- Om du upplever att testpersonen säger/gör någonting som kan vara värt att anteckna i koppling till något moment så använd fältet för kommentarer

Instruktioner:

Uppladdning av dokument

Du ska nu få använda ett verktyg som ska hjälpa dig hitta information i texter. Verktyget gör detta genom att du ska kunna ställa frågor om innehållet i texten. För att veta ungefär vad texten handlar om så kan systemet sammanfatta texten åt dig. I mappen <något mappnamn> ligger filen UnionJack. Var god och ladda upp den till hemsidan. Du vill klicka i att den ska använda den experimentella sammanfattningen

[Här betygsätter testledaren hur bra testpersonen lyckas ladda upp filen 3 = med stor säkerhet 0 = lyckades inte alls]

Betyg:

Figure 10 Usertest script part 1

Kommentarer på detta moment:

Använda sammanfattningen

För att få reda på vad texten handlar om kan du använda sammanfattningen, ta fram sammanfattningen och läs den.

[Här betygsätter testledaren hur bra testpersonen lyckas använda sammanfattningen 3 = med stor säkerhet 0 = lyckades inte alls]

Betyg:

Kommentarer på detta moment:

Använda QA

Du vill nu tar reda på **Vilka länder var med i skapandet av Union Jack?** och **Vilken kung skapade flaggan?** (skriv upp två saker här som testledaren ska be användaren ta reda på)

[Här betygsätter testledaren hur bra testpersonen lyckas använda QA 3 = med stor säkerhet 0 = lyckades inte alls]

Betyg:

Kommentarer på detta moment:

Uppladdning av ett nytt dokument

Nu vill du ladda upp den andra filen BatmanLong för att undersöka den, gör detta nu.

[Här betygsätter testledaren hur bra testpersonen lyckas ladda upp den andra filen 3 = med stor säkerhet 0 = lyckades inte alls]

Betyg:

Kommentarer på detta moment:

Använda sammanfattningen igen

Ta fram en sammanfattning på den här texten.

[Här betygsätter testledaren hur bra testpersonen lyckas använda sammanfattningen 3 = med stor säkerhet 0 = lyckades inte alls]

Figure 11 Usertest script part 2

Betyg:

Gå tillbaka till första texten och ställa en tredje fråga

Nu vill du ställa frågor till den första texten du laddade upp, ta reda på vad **Vilket år skapades Union Jack?**.

[Här betygsätter testledaren hur bra testpersonen lyckas välja den första filen igen och ställa frågor till korrekt text 3 = med stor säkerhet 0 = lyckades inte alls]

Betyg:

Färdig!

Frågor:

- **Hur bra upplevde du sammanfattningen var?**
- **Hur bra upplevde du svaren på frågorna var?**
- **Ser du en användning för applikationen i ditt dagliga liv? (arbete/studier) Isf hur?**
- **Vad tycker du kan förbättras?**

Avslutande kommentarer från testpersonen?

Figure 12 Usertest script part 3

B Code

B.1 React

```
import React, { Component } from 'react'
import Form from './Form'
import FileForm from './FileForm'

class FormHandler extends Component {

    state = {
        isFetching: false,
        answer: '',
        summarization: ''
    }

    handleQuestion = (text) => {
        this.changeState()
        fetch("/api", {
            method: 'post',
            headers: {
                "Content-type": 'application/json'
            },
            body: JSON.stringify(text)
        })
        .then(resp => resp.json())
        .then(data => {
            this.setState({
                isFetching: false,
                answer: data.answer
            })
        })
    }
    handleFileUpload = (url, file) => {
        this.changeState()
        fetch(`/${url}`, {
            method: 'post',
            body: file
        })
        .then(resp => resp.json())
        .then(data => {
            this.setState({
                isFetching: false,
                summarization: data.sum
```

```
            })
        })
    }
    changeState = () => {
        this.setState({ isFetching: !this.state.isFetching })
    }

    render() {
        const fetching = this.state.isFetching
        const comps = <div>
            <p>{this.state.summarization}</p>
            <Form className="btn btn-primary"
                answer={this.state.answer}
                text="Submit Here"
                sendQuestion={this.handleQuestion} />
            <h1>You can also upload Files</h1>
            <FileForm sendFile={this.handleFileUpload} />
        </div>;
        const res = (!fetching ? comps : <h1>fetching</h1>)
        return (
            <div>
                {res}
            </div>
        )
    }
}

export default FormHandler;
```

B.2 CSS

```
/* element selector */
h1 {
  font-size: 48px;
}

/* class selector */
.green-text {
  color: green;
}

/* id selector*/
#Welcome-page {
  margin: 0 auto;
  background-color: #a2536c;
  width:720px;
}
```

C Surmize UI Components

These different sections define the UI of different aspects of the website. The first section depicts the three main sections on the landing page, the next showcases some of the design principles used for the upload/landing page. The last section defines UI and design principles for the file management/workspace page.

C.1 Landing Page

Figure 13 All pages of the single-page application

C.2 Design Principles for the Landing Page

This section showcases the main upload page and some of the design patterns implemented to help the user and reduce clutter

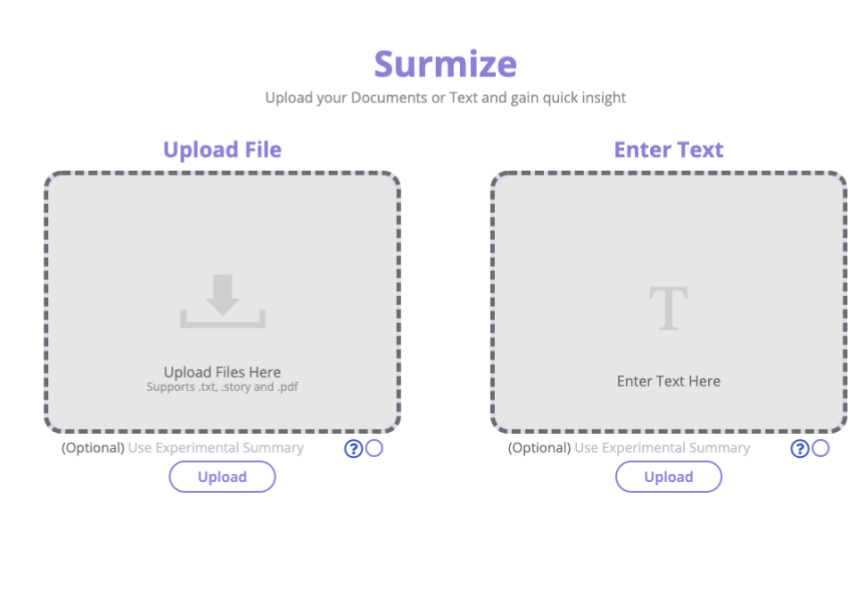
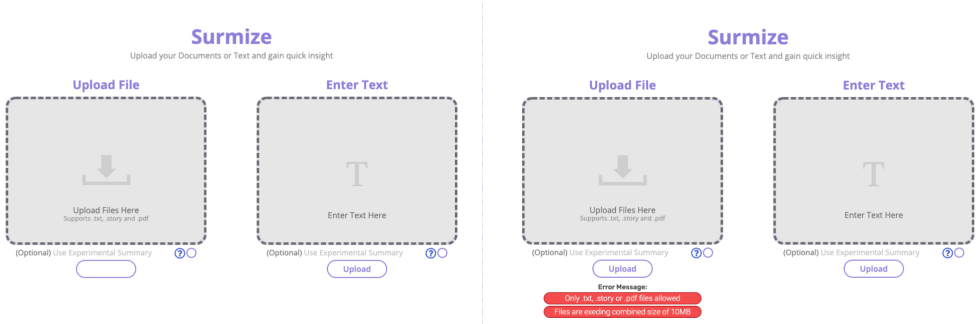


Figure 14 Upload Section



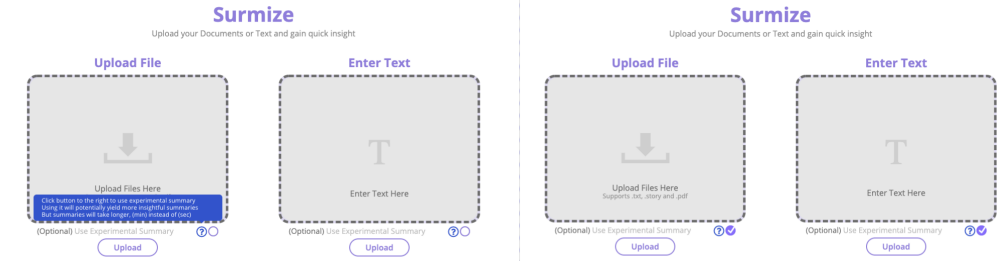
(a) Upload button disabled if no file/text is selected for upload

(b) If incorrect file types, files have too many pages or similar, an error message will be displayed

Figure 15 Design principles to guide users on what and what not to upload

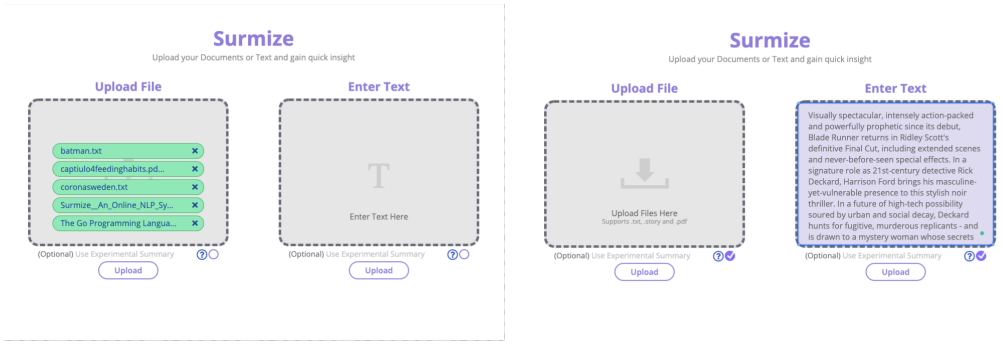
A help icon is implemented to be used near sections that could require more more in-depth explanation. This was implemented to reduce the amount of text shown, as well as making sure to only show text when a user want to understand a section more. We

reasoned that once a user has read and understood the help information, it takes up valuable screen real estate.



(a) Help button to hover over **(b)** Check button

Figure 16 Design principles to inform about a specific part and hide info



(a) Files **(b)** Text

Figure 17 UI of file upload and text upload section

C.3 Design Principles for the Workspace

This section depicts the UI and design principles of the file/workspace page.

The first image depict a text uploaded via the text form (see fig. 17b), which is Rotten Tomato's summary of Blade Runner (The Final Cut) [79]. The currently active file is highlighted with a purple color to the left. The summary of that file is displayed in the window titled "Summary" and the questions posed to the system are displayed in the window titled "Question-Answering". Underneath each window, the name of the file currently used and asking questions about are also displayed as a subtitle to each window. The summary aims to be a guide to what might be worth asking about. The answers and questions are displayed in a dialogue system. Each answer yielded back to the user are marked by the model with a corresponding confidence score.

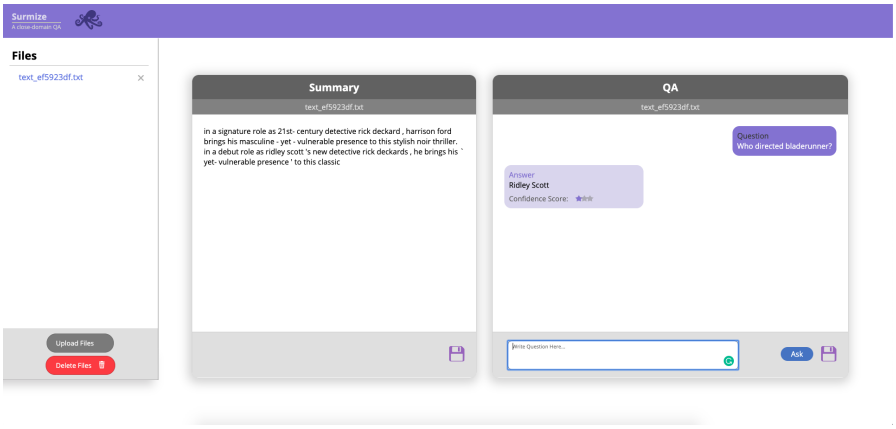


Figure 18 Illustration of the workspace, and the corresponding summary and QA conversation for that file

The next image depicts when a user has uploaded multiple files into their workspace. Hovering over a file will add a purple notch to the left of it. Each file has a small cross to the right of it to indicate that an individual file can be removed. Hovering over any file will turn the cross red to indicate that they can remove that file. A user can also remove all files by pressing the large red button in the bottom left corner.

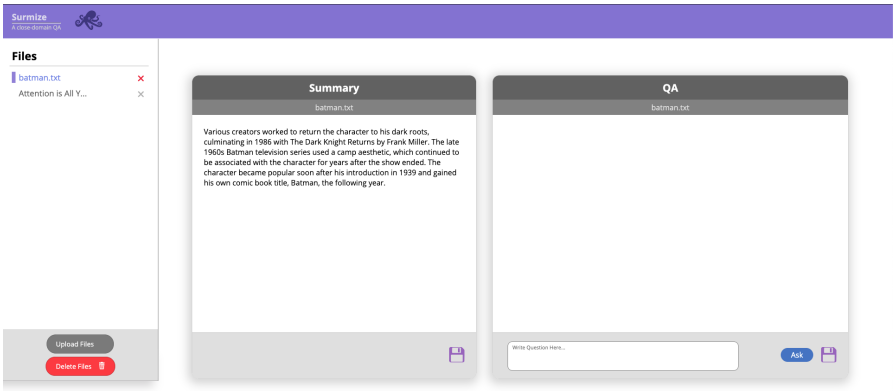


Figure 19 Having multiple files in the workspace

A user can also upload new files or text by pressing the large gray button in the bottom left corner. This will open the same viewing area as on the landing page. This is to ensure that the user quickly gains familiarity with the UI. A user can either upload files/text or press the bottom left button again to go back to the workspace.

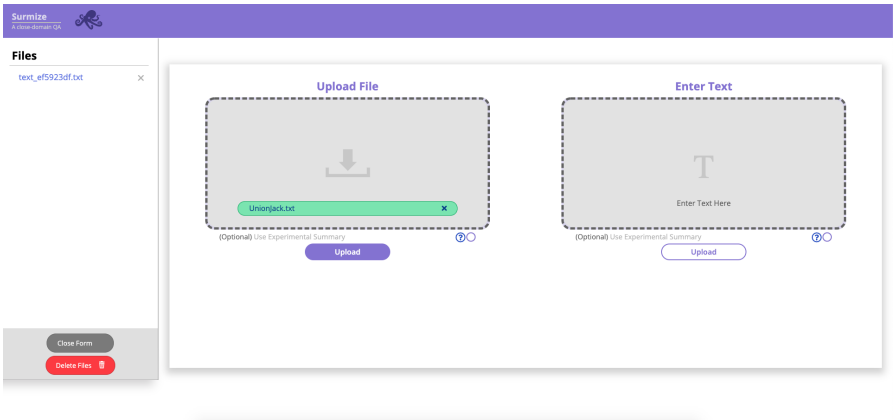


Figure 20 Upload other files or text from the workspace

D Surmize API End-Points

The following illustrates the web API end-points, when viewed from either the path `/docs` or `/redoc`. Each end-point is documented and testable trough the browser.

Figure 21 API end-point documentation viewed from the `/docs` path

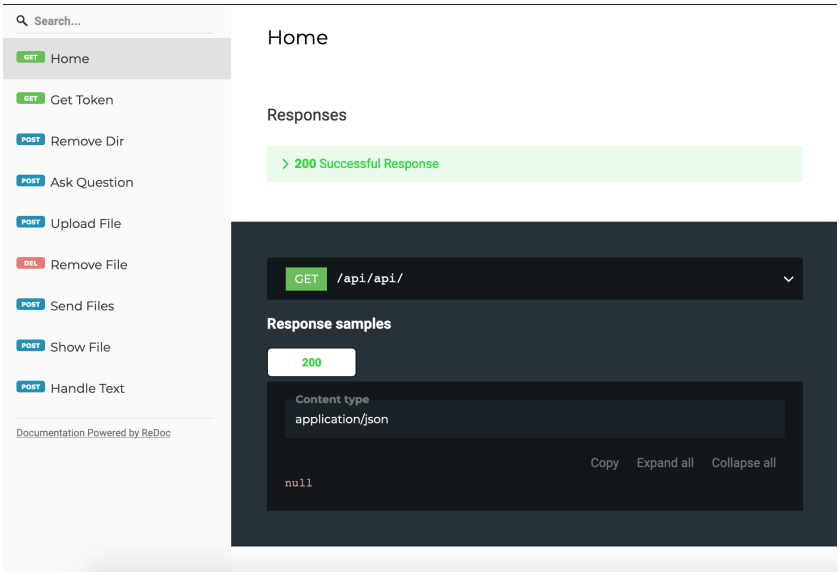


Figure 22 API end-point documentation viewed from the `/redoc` path