Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Maksim Shmakov

SYSTEM FOR EMAIL ANALYSING AND RESPONSE GENERATION BASED ON MACHINE LEARNING

Examiners: Associate Professor Jussi Kasurinen
Associate Professor Saleh H. Mohammed

Supervisors: Associate Professor Jussi Kasurinen
Associate Professor Saleh H. Mohammed

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Maksim Shmakov

System for Email Analyzing and Response Generation Based on Machine Learning

Master’s Thesis

40 pages, 16 figures, 6 tables

Examiners: Associate Professor Jussi Kasurinen
Associate Professor Saleh H. Mohammed

Keywords: Automatic email reply (AMR); Recurrent Neural Network (RNN); Sequence-to-sequence (Seq2seq); Long Short-Term Memory (LSTM).

Email is one of the most popular and effective communication channels in the modern world both for personal tasks and for business. However, the amount of unanswered received emails sometimes can be overwhelming. Systems for automatic email response generation and analysis through unsupervised machine learning technics have recently been proposed and are getting more popular. Unfortunately, these systems mostly partially provide options that work great in services like chat-bots, but not in the business correspondence tasks. The main purpose of the dissertation is to provide an open-source system based on deep learning neural networks, which would be capable of generating automatic replies based on the user’s personal history of email replies. The system is based on a sequence-to-sequence encoder-decoder approach with long short-term memory architecture.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisors Saleh H. Mohammed (HSE, Russia) and Jussi Kasurinen (LUT, Finland) for the great research and implementation guidance along with the organizational help related to the underlined paper. Without it, it would become way harder for me to keep up with all the aspects of work.

Secondly, the teachers and courses at both Universities highly contributed to the personal conviction that I have chosen the right way in life as a Software Engineer and that I deeply love what I do. Thus, I am thankful for all people who contributed to it during my period of education.

Apart from that, I am grateful to both Universities for the given opportunity to take advantage of the double degree program even during this hard COVID-19 pandemic period. The possibility of receiving the scholarship for the whole period of abroad education is also a thing that significantly contributed to making it possible for me to receive a degree.

Finally, I want to thank my parents, Tatiana and Vyacheslav. Without your support, my life would be completely different and for sure not in the best way. In addition, a great thank you to my girlfriend Vasilisa for supporting my choices along with life and for her understanding and trust.

Moscow, 20 June 2021

Maxim Shmakov

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AMR	Automatic Mail Reply system
BoW	Bag of Words
Doc2Vec	Document to Vector models
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory recurrent neural network
NLP	Natural Language Processing
RNN	Recurrent Neural Network
Seq2seq	encoder-decoder based method of sequence translations (Sequence-to-Sequence)
TD-IDF	Term Frequency–Inverse Document Frequency statistic
GUI	Graphical User Interface

1 INTRODUCTION

1.1 Background

The main goal of the current paper is to provide a design and implementation of Automatic Mail Reply (AMR) system architecture which should be capable of generating email replies based on the common replies mixed up with the personalization. Such a kind of task inevitably leads to the need for unsupervised learning since no supervised methods can generate unique long text replies (Feng et al., 2020). As in parallel to the machine translation task deep learning can improve the quality of automatic email replies following their length improvement. Sequence-to-sequence (seq2seq) models' composition is chosen as the main network architecture with two separate neural networks responsible for the encoder and decoder functionality respectively. Enron email dataset should be used for the task of initial neural networks training (Klimt & Yang, 2004). Data should be cleaned using SciKit-learn (Pedregosa et al., 2011) and NLTK (Loper & Bird, 2002) python libraries. Personalization should be achieved through the process of feeding the history of a specific user's messages to both neural networks, which leads to an inference of a specific user's writing style. The results might be compared to the traditional TF-IDF, LSA, and LDA (Schiller, 2015) solutions of the short-reply generation task.

The described architectural approach might be used already in the proprietary software due to the rising popularity of seq2seq models use outside of machine translation, but to the best of the author's knowledge, no open-sourced architecture implementation of such kind exists. Moreover, no ready-to-use system of such kind is present in the open-source community.

Implementation of the system proof-of-concept should be provided as open-source software under the Apache 2.0 license. The objectives of the system are to build separate server-side and client-side applications. The first one should contain trained neural networks along with the SMTP/IMAP subscription handling logic and JSON REST API endpoints for client-server communication purposes. Postgres should be used as the main database to store already-fetched messages from the SMTP/IMAP server. The client-side is intended to be used by the target user to provide an interface to the server. It consists of the

webpage, user input, and HTTP requests handling logic. It can show both the client's inbox with all the messages user currently has and the top suggested answers to a new email. The user is also able to add arbitrary text to a suggested reply or make all necessary corrections to it before the message is sent to the target SMTP server.

1.2 Goals and objectives

The main goal of the whole work is to present the proof-of-concept system helpful in decreasing the time required to handle the emails by the target users. This should be anticipated after the overall system design overview and justification of tools and methods are given.

Key objectives are:

- Give an overview of the current state of such email systems present on the market and key approaches utilized to build such systems;
- Design a system constrained by the requirements;
- Implement a proof-of-concept of such a system.

1.3 Delimitations

The key aim of the current research is to provide the architecture of the discussed system, substantiate the decisions along with the need for specific tools and implement a working prototype. The resulting system is not tested on the real users due to lack of time and resources, just the synthetic benchmarks with the use demonstration are provided.

The paper by any means is not aimed to present a new way to generate the replies by synthesizing a novel neural network architecture or a deterministic approach for text generation. Novel text summarization approach development is not a priority for this paper as well.

1.4 Structure of the thesis

The thesis is structured like the following. The "Theoretical framework" section contains

the background of the research in the field of AMR systems with an overview of the main existing methods. Comparison with existent real-world competitors is present.
The succeeding “Methodology” depicts both functional and non-functional key requirements the system should comply with. This is followed by the detailed description of the dataset used for the initial neural network training together with the specifics about selected models.

The following “System implementation” chapter describes the foundations of the approach and the implementation details of the application including the design of the neural networks and both client- and server-side parts.

The next chapter “System testing and evaluation” contains the performance evaluation of the resulting system and comparison with described in background approaches. The last chapter of the paper is “Conclusion” which is dedicated to sum results up and suggest possible improvements of the resulting system proof-of-concept.

2 THEORETICAL FRAMEWORK

This chapter provides an overview of the main competitors and the theoretical fundamentals of the system being developed.

2.1 Existing solutions

Email reply systems in general are a common thing in the 21st century. Different applications with various features are widely available on the market. Nevertheless, the most robust and functional applications still are developed and supported well only by the biggest competitors. Some of them are already offering the features that the underlying system should implement, so a comparison of them is shown in table 1.

Table 1. Main competitors opposite the offering functionality.

Competitors	Functionality					
	Automatic reply generation	Automatic email tagging	Automatic reply heading generation	Automatic deadlines detection	Available as a self-hosted solution	Open-source solution
Yandex	No	Yes	No	Yes	No	No
Google	Yes	Yes	No	Yes	No	No
Spark	Yes	Yes	No	Yes	No	No
Apple Mail	No	Yes	No	Yes	No	No
Mozilla Thunderbird	No	Yes	No	Yes	No	Partially
Outlook	Yes	Yes	No	Yes	No	No
Spike	No	Yes	No	No	No	No

Mailbird	No	Yes	No	No	No	No
eM client	No	Yes	No	No	No	No

As can be seen from the table, there is no application available on the market which would cover all the defined features.

2.2 Email analysis approaches

The email was opened for the general public in 1971 by Ray Tomlinson (Tomlinson & BBN Technologies, 1971) and it remains one of the most popular communication channels in the modern world. Despite various messaging apps arising like Telegram and Facebook Messenger, there are ~3.9 billion active mail users in the world (Accounts et al., 2019) for both personal and business communication. It is also used as one of the main communication tools in the field of state-to-public communication in various countries. Some statistic agencies are conducting research and present statistics on the average number of emails that an ordinary person receives a day. According to the Radicati Group, these numbers are around 117 emails per day (Accounts et al., 2019). This leads to the phenomenon called “email overload”, which is essentially a situation where the person is lost in the number of emails that are coming into the inbox (Dabbish & Kraut, 2006; Whittaker & Sidner, 1996). Implicatively, some of the emails are marked by a person as not important and the answer to them is postponed for the sake of the most important ones (Hair et al., 2007). In addition, the percentage of mobile devices is growing rapidly and is already overcame the percent of desktop users. That leads to the use of mobile devices for reading and answering email tools, where it is much less convenient to type text fast. A possible way to solve both of the problems is to create a reply generation system that would suggest some of the possible answers that can be sent as-is by a single tap or easily corrected. In this situation, the typing speed would become not a big concern and the number of situations where the person needs to postpone the answer would dramatically reduce.

2.2.1 Automatic response generation systems

These ideas and tasks are not new and there are plenty of approaches already proposed and implemented by various researchers and technological companies. Most of the research is based on the unsupervised learning concept of a bag of words (BoW) and falls into the topic modeling category of Natural Language Processing (NLP) (Moubayed et al., 2020; Nikolenko et al., 2015). Essentially, topic modeling is the direction of machine learning whose key objective is to find patterns in the unlabelled text data, usually called documents, and cluster those documents into a finite set of topics (Jones, 1972). New input documents can further be attributed to one of the defined topics. BoW in this context is the way of representing the data structure of the documents existent in the database. The structure itself is usually represented as a vector of words that are located in the input document, e.g., SciKit implementation through CountVectorizer (scikit-learn developers, 2020).

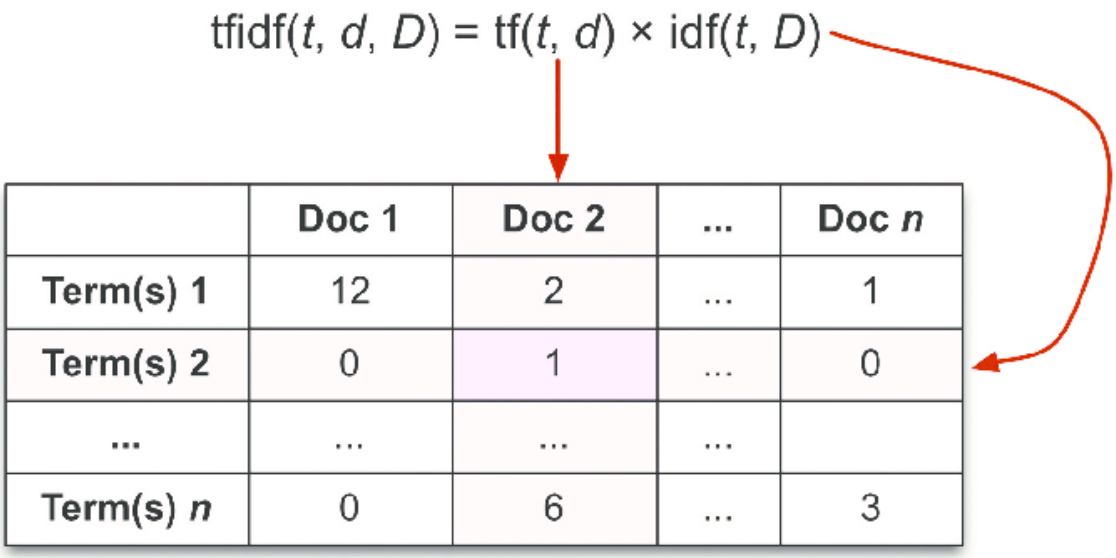
There are several approaches to how to use the BoW vectors to process the documents and find similarities between them. The similarities are necessary to determine the topic of the new document and then extract the similar ones, that can have an email reply which probably can be a suitable email reply suggestion (Ramos, 2003). The first and the most common one is the Term Frequency - Inverse Document Frequency (TF-IDF) model (Ramos, 2003). It consists of respectively Term Frequency and Inverse Document Frequency models.

Term Frequency essentially ranks all the terms that can be found in the document collection by the number of times the word is present in a document (Ramos, 2003). This information can be used later for determining which of the documents most probably corresponds to the query executed on the collection. In this approach, the most frequent terms would correspond to the most valuable features, the presence of which in the document with high probability means that the document relates the topic.

But in the case of the most common words like prepositions or frequent nouns the Term Frequency model tends to misclassify the documents prioritizing not the most meaningful words that are most important to distinguish the right topic. Inverse Document Frequency is the solution to this case. It is used for the task of downrating the most common words

among the collection of documents (Ramos, 2003). Statistically, it is a function that divides the number of documents by the number of documents in which the specified term can be found. Calculated for each of the terms in the collection, it can be then used in prioritization of the terms based on which term contributes the most. The most insignificant words are usually eliminated from the documents, but this model still helps to define the most significant among those that are left.

Combined, these two statistical models form the TF-IDF by multiplication of calculated values for each of the terms present in the collection. In figure 1 the resulting TF-IDF matrix can be seen.



	Doc 1	Doc 2	...	Doc *n*
Term(s) 1	12	2	...	1
Term(s) 2	0	1	...	0
...	...	...	...	
Term(s) *n*	0	6	...	3

Figure 1. Constructed TF-IDF matrix (Al-Khateeb & Epiphaniou, 2016).

The way a query can be executed is by breaking it into a set of terms and checking each term for the top relevant documents, in the end assigning them to the query. Various AMRs (Feng et al., 2020; Schiller, 2015) then use this information to get those documents and extract replies that are then interpreted as the suggestion to the email answer.

Another unsupervised topic modeling approach is the Latent Semantic Analysis (LSA). In this case, the main idea is that the semantically closed terms are located in similar text paragraphs (Avinash, 2018). The way it works is that the initial documents are breaking into paragraphs and those paragraphs are used as the documents later in the analysis. Then

TF-IDF matrix is generated, and a technic called single value decompression (factorization) is executed above the resulting matrix to decompose it into several matrices representing different points views on the dataset. Based on those matrices and the document query we can define the most probable topic from different points of view (e.g., figure 2).

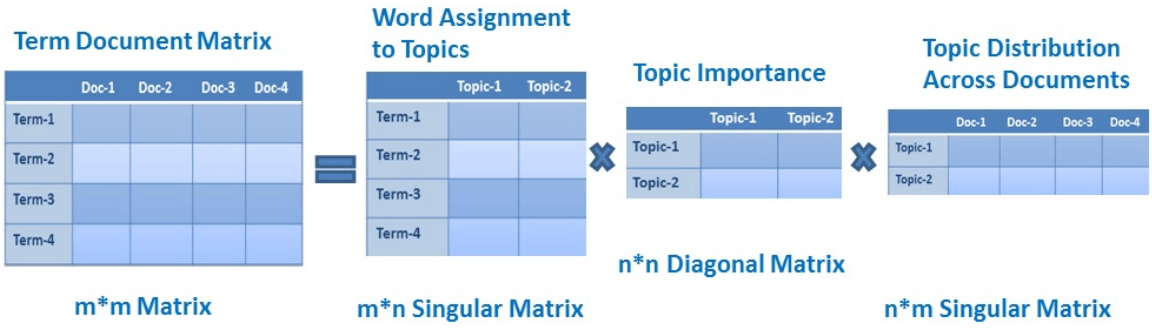


Figure 2. Latent Semantic Analysis matrix definition (Avinash, 2018).

The last approach is the Latent Dirichlet Allocation (LDA). It works differently from the TF-IDF approach in the sense that in this model it is assumed that each of the documents in the collection can belong not to just one topic, but a small set of topics. This comes in handy when some of the topics contain the same number of significant terms and it becomes impossible for the previous model to determine the right topic for the query. This model tries to find the words that are related to the topic, but if all terms related to the topic might be allocated among several documents and the single one of them might not contain all the terms. Algorithmically, the model firstly assigns to each of the documents some set of topics and then several metrics are calculated, such as the number of terms in the specific document per topic and how significant is some term presence of which makes the whole document belong to some specific topic (Lee et al., 2018). Finally, all the documents are assigned with a set of topics where each of them has its probability. AMRs in the same way as previous use this information to extract the most similar documents to get the previous reply and suggest it to the user. In figure 3 the model of the LDA is depicted.

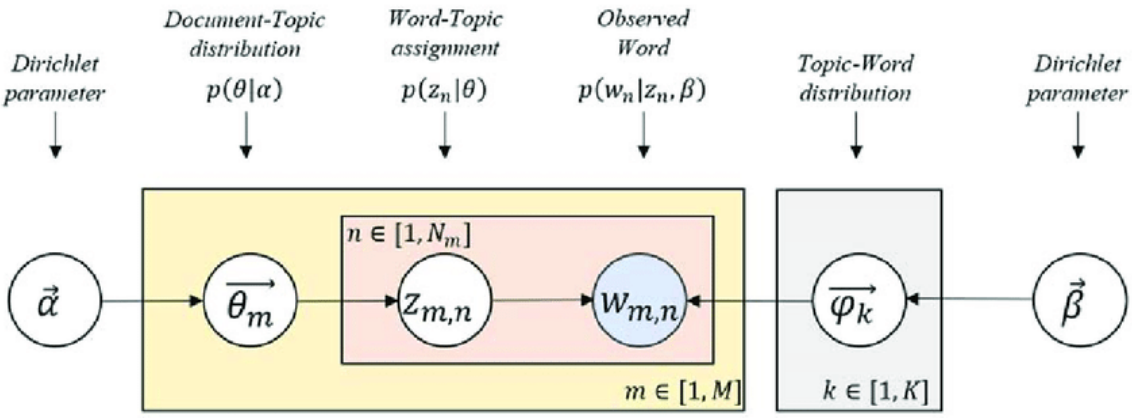


Figure 3. **Model of the statistical Latent Dirichlet Allocation (Lee et al., 2018).**

LDA would not be a very useful solution in the case of email topic modeling, because it considers several topics per one document, whereas most emails contain only one specific topic. There are approaches in the market, such as a single-topic LDA with clustering (stLDA-C) model (Graham, 2020). It considers only one specific topic and was used previously by the author for topic modeling purposes of the Twitter posts dataset, but it can be used in the task of email topic modeling potentially yielding better results than the ordinary multi-topic LDA.

All described models are relying on the bag of words vectors basis. Generally, this is not enough to consider the semantics of the text, features like word order or context significantly contribute to the overall text understanding, which is necessary for accurate language processing and topic modeling. Apart from this, all the systems are relying on the unmodified previous answers of either the client itself, the common general dataset, or the mix of both. This does not give the opportunity of generating ad hoc artificial responses that were not present in the dataset. In addition, the person's style of writing is fully lost if an open-sourced commonly used collection of documents is used as the main dataset.

To address this problem, deep learning approaches are coming on the market. Recently, Facebook Messenger released smart replies for the messages based on recurrent neural networks. In addition, various chat-bots that can dynamically answer questions or personal assistants based on the RNNs are becoming popular. Those systems know much more

about the natural language because they consider much more language features and are capable of text generation. For instance, the long short-term memory (LSTM) model.

There are already existent solutions that use recurrent neural networks on the market to dynamically predict the text that the user would write himself. For instance, Smart Compose by Google uses specially adjusted LSTM-based models (Yonghui, 2018) to predict what the user will type next. It is a completely different task apart from generating the whole response, but the results provided in the latest report tell that there is a potential of using such systems in the field of ad hoc text generation.

The typical architecture of the LSTM model looks like in figure 4. The model processes the input data word-by-word and due to the cells on layers having hidden states, it is capable of ‘remembering’ the previously processed data by preserving the dependencies and then adjust the output accordingly (Kannan et al., 2016). Typical architecture (taken from the machine translation field) consists of the encoder which translates input documents into a vector representation and the decoder which takes as an input a vector and generates the output in a document format. This model does not need to be completely recomposed to put new data into the cell's state. Instead, new documents (emails) can be fed directly to the input of the encoder and then implicitly used by the neural network.

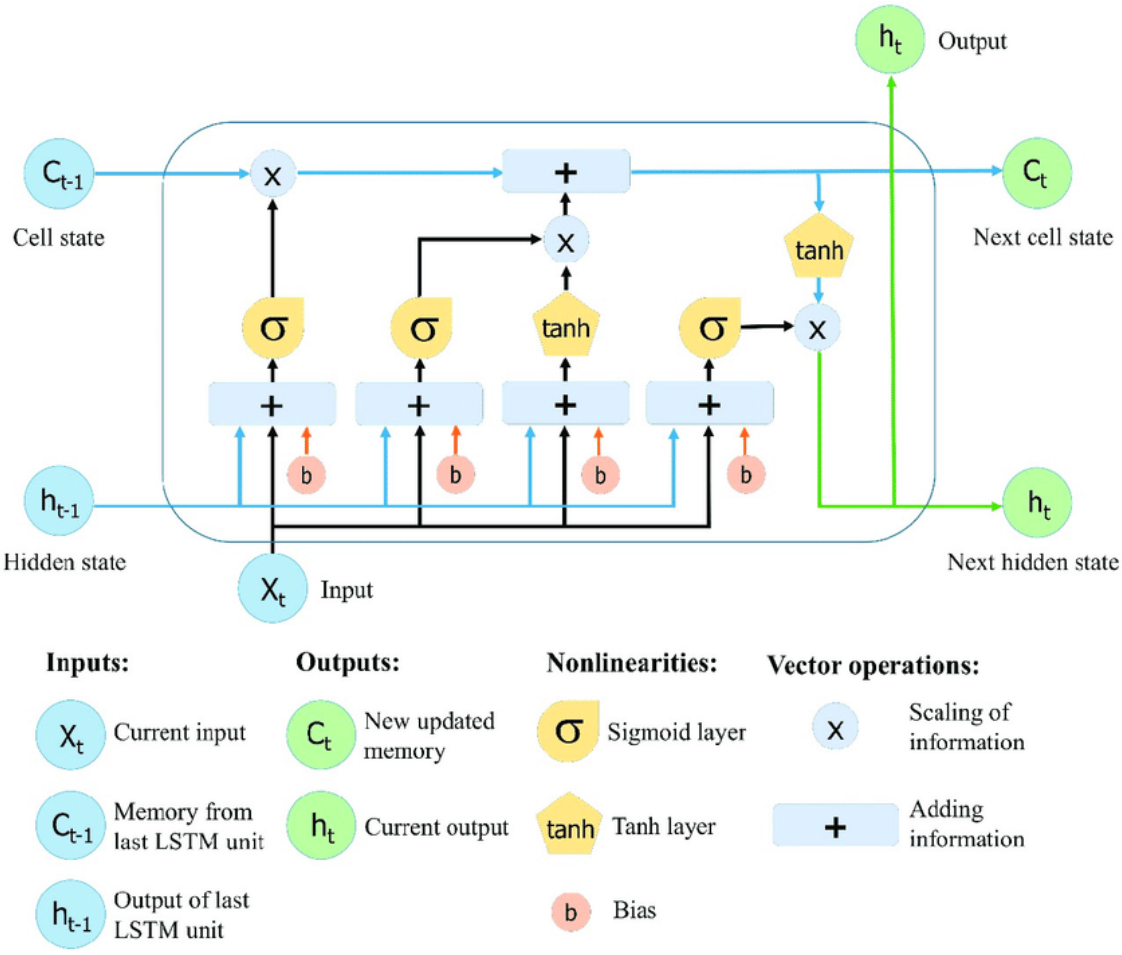


Figure 4. LSTM model architecture.

Other types of RNNs architectures are also present on the market, like transformers (Kannan et al., 2016). They are generally yielding better results, e.g., in the machine translation field of machine learning. But unfortunately, those neural networks require a huge amount of data and GPU time to learn which results in enormous budgets that are not accessible by the author of the current thesis. That is why the following chapter would be dedicated to the detailed description of the LSTM model with the reasons to fit into the performance-cost trade-off.

2.2.2 Abstractive text summarization task

Abstractive text summarization task is different from the previous one and requires the adoption of the previously discussed models to work properly or requires something completely different.

An example of the statistical way of solving this task can be a regression-based summarization (Ulrich et al., 2009). The authors of this research use the system based on the clue words which is useful for the email conversations if they are represented in the graph view with those words being in the adjacent nodes of the built graph. As a model, the regression-based classifier was used, and the results were better than when using the binary classifier.

There are also several main indeterministic ways to solve the task. The first one is called *extractive* and the second one is *abstractive* (Zhong et al., n.d.). They are respectively contextual and non-contextual. The use of LSTM-based neural networks finds its way to exist in both of those ways (Zhang & Tetreault, 2019) and the architecture from the underlined research can be seen in figure 5.

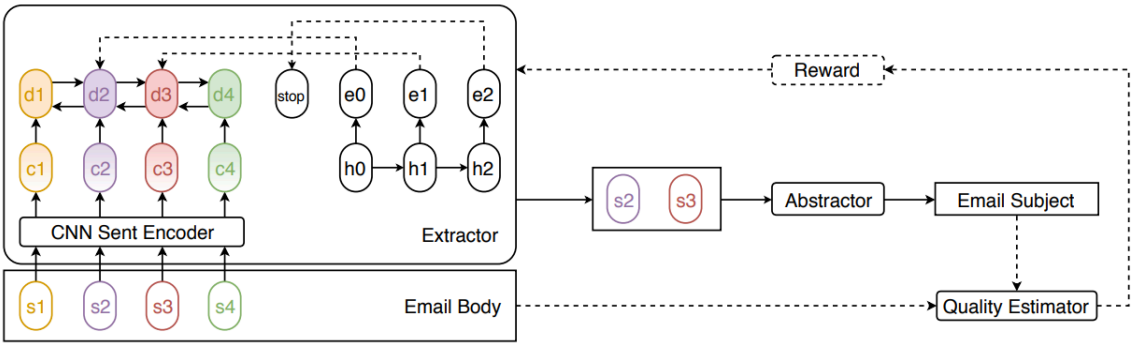


Figure 5. LSTM model architecture (Zhang & Tetreault, 2019)**.**

METHODOLOGY

2.3 Requirement's analysis

To satisfy evolving needs of email users, several core features are proposed to be implemented in the system. All of them are targeted to be used without any of the users' direct input by utilizing neural networks.

2.3.1 Functional requirements

Table 2 contains all the functional requirements that are mandatory for the system to implement.

Table 2. Functional requirements**.**

Code	Functional requirements description
FR1	The system shall be able to connect to any of the email providers that are publicly accessible from the Internet and are supporting SMTP/IMAP subscriptions.
FR2	The system shall be able to automatically (without users' direct input) generate three different variants of responses for a new email.
FR4	The system shall be able to automatically generate email reply heading based on the received email content.
FR5	The system shall be able to automatically detect deadlines and events (both time-based and date-based) and generate the link to a Google Calendar detected event creation.
FR6	The system should provide the user with the ability to respond to the incoming email without the need of using external services.

FR7	The system should provide a way to attach files to the outgoing email and to download all attachments from the incoming emails.

2.3.2 Non-functional requirements

Table 3 contains all the non-functional requirements that are mandatory for the system to implement. Table 4 contains the server requirements under which the system would still be stable.

Table 3. Non-functional requirements**.**

Code	Non-functional requirements description
NR1	The system shall be available as a self-hosted on-premises solution for free.
NR2	The system shall be an open-source solution.
NR3	The system shall be able to handle 10 requests per second without degradation.
NR4	The system shall be easily manually scalable under the high detected load.
NR5	System response time for a typical API call should not be more than 1 second.
NR6	The system shall be modular to improve maintainability.

Table 4. Maximum server requirements**.**

Aspect	Requirement

CPU	Quad-core server one or same power virtually allocated
RAM	1 GB
Disk space	10 GB
OS	Linux-based distributives, but should be also OS-agnostic

2.4 Datasets

There are numerous email-related datasets open for public use on the web (Sathiyanarayanan, 2020). Since the described system can be virtually divided into several subsystems, it is possible to use the most suitable datasets for each of the systems separately.

For the AMR seq2seq system to be trained, it is required to have initial data based on which neural networks can generate the output result. According to base studies discussed in the theoretical framework chapter and (Sites.google.com, n.d.), the most suitable dataset to use for this task is the Enron Email Dataset (Klimt & Yang, 2004) due to the biggest number of emails and previous successful experience with seq2seq models. The example record from this dataset is present in table 5.

Table 5. Example of records from the Enron dataset**.**

Record

Date: Mon, 16 Oct 2000 06:42:00 -0700 (PDT)
From: phillip.allen@enron.com
To: buck.buckner@honeywell.com
Subject: Re: FW: fixed forward or other Collar floor gas price terms
Mime-Version: 1.0
charset=us-ascii
Content-Type: text/plain
Content-Transfer-Encoding: 7bit
X-From: Phillip K Allen
X-To: ""Buckner, Buck"" <buck.buckner@honeywell.com> @ ENRON
X-cc:
X-bcc:
X-Folder: \Phillip_Allen_Dec2000\Notes Folders\'sent mail
X-Origin: Allen-P
X-FileName: pallen.nsf

Mr. Buckner,

For delivered gas behind San Diego, Enron Energy Services is the appropriate Enron entity. I have forwarded your request to Zarin Imam at EES. Her phone number is 713-853-7107.

The task of reply email heading generation is different from the previous one, so the most suitable dataset is different too. One of the best suitable examples is the BC3: British Columbia Conversation Corpora (Ulrich et al., 2008) which was successfully used in (Ulrich et al., 2009) for the same text summarization task solving purposes. This dataset contains annotated emails for more than 40 email threads (more than 3000 sentences) of the respected company which made this dataset public. One record from this dataset is present in table 6.

Table 6. Example of records from the **BC3: British Columbia Conversation Corpora.**

Record

<DOC>
<Received>Mon Mar 31 09:29:15 -0800 2003</Received>
<From>Sharon Laskowski <sharon.laskowski@nist.gov></From>
<To>public-usability-workshop@w3.org</To>
<Subject>W3C UIG informal SIG at CHI––tentative schedule</Subject>
<Text>
<Sent id="6.1">The most convenient time to schedule the W3C UIG informal sig discussion is: Tuesday 4/8 1-2:30pm </Sent>
{…}
<Sent id="6.6">I will email the informal SIG people and get on the schedule this afternoon and confirm via this listserv. </Sent>
<Sent id="6.7">Hope to see you next week.</Sent>
<Sent id="6.8">Sharon</Sent>
</Text>
</DOC>

3 SYSTEM IMPLEMENTATION

3.1 Architecture

The system uses client-server architecture meaning the user is interacting with the front-end part of the system located in the browser which in turn interacts with the back-end via HTTPS protocol. Whereas the front-end is a single system, the back-end consists of multiple subsystems interacting with the main service that is serving the HTTPS requests, handles SMTP/IMAP communication with the external email provider, and communicates with the client. Subsystems are essentially neural network modules responsible for the specific task. The abstract high-level design of the application can be seen in figure 6.

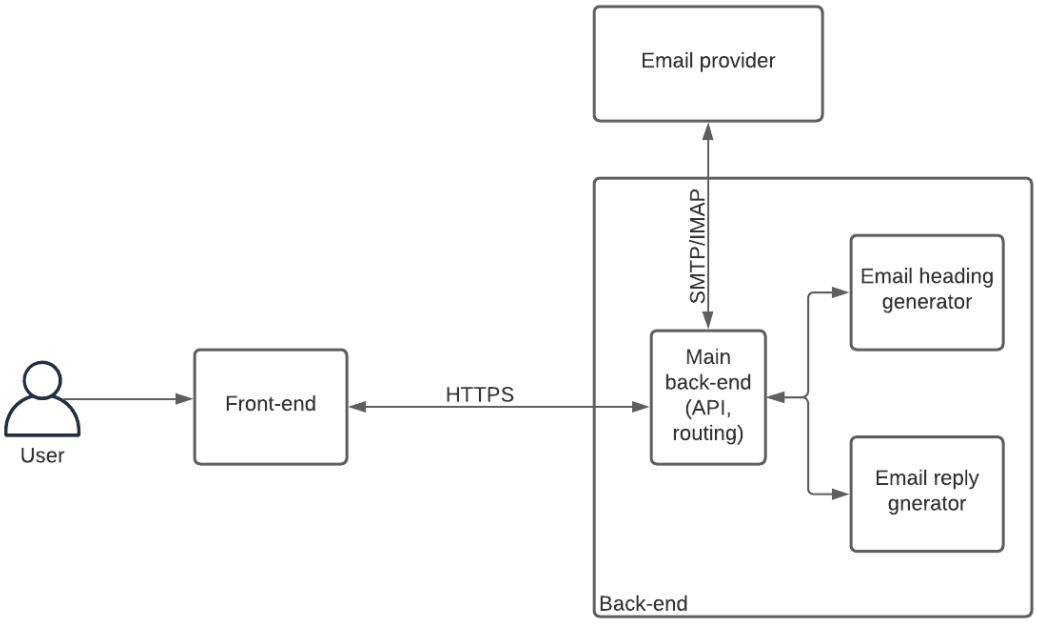


Figure 6. High-level system design.

The main back-end consists of the API endpoints handler, SMTP/IMAP handler, task queue, and the database in which the emails are stored. Task queue is necessary for the communication through the SMTP protocol since the requests are long-running and the best practice is not to enforce the user to wait. The database is used to store the emails and already generated potential responses, email headers, and tags. The detailed design of the

main back-end can be seen in figure 7.

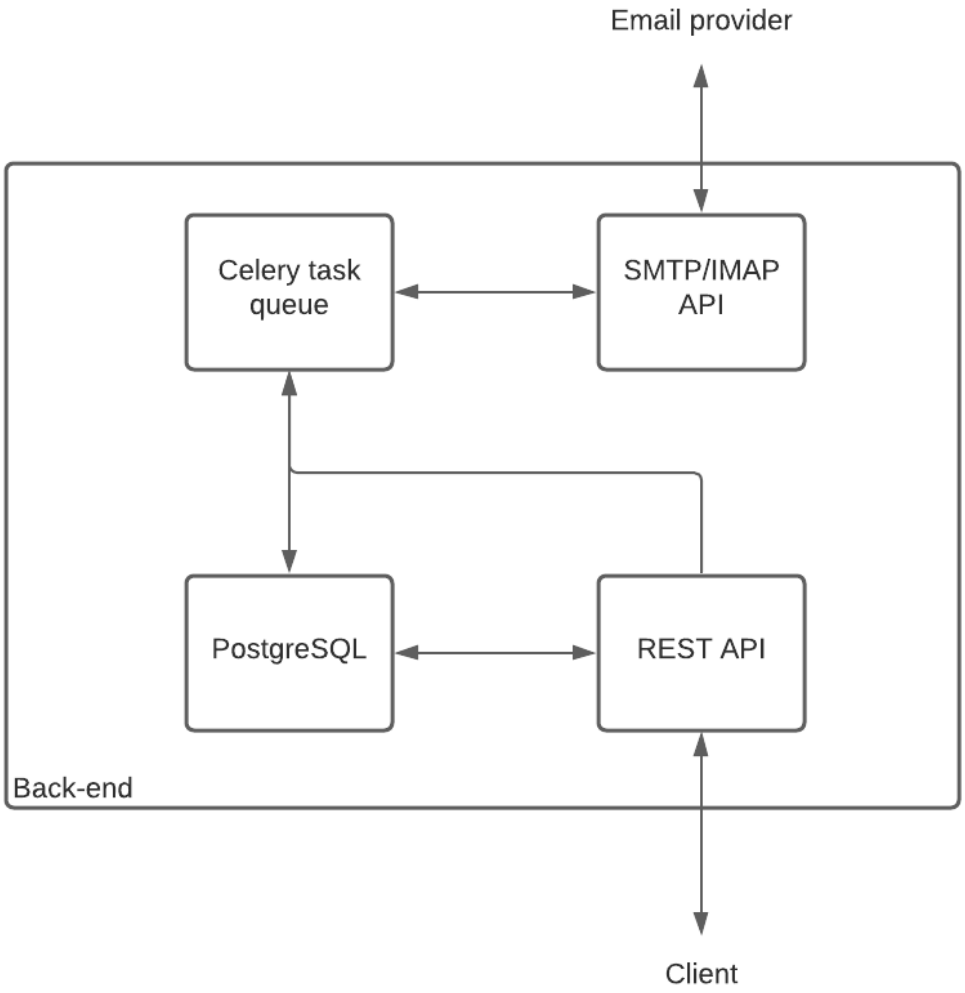


Figure 7. Detailed main back-end design.

The automatic response generation subsystem consists of the seq2seq LSTM neural network which would take users' previous emails from the history as an input and output the suggested reply. The following process is repeated if the email itself is big enough to generate several appropriate suggested replies. The same architecture of the LSTM model (figure 8) is used for the abstractive text summarization task, but the layers would be technically different (details are described in the following sections).

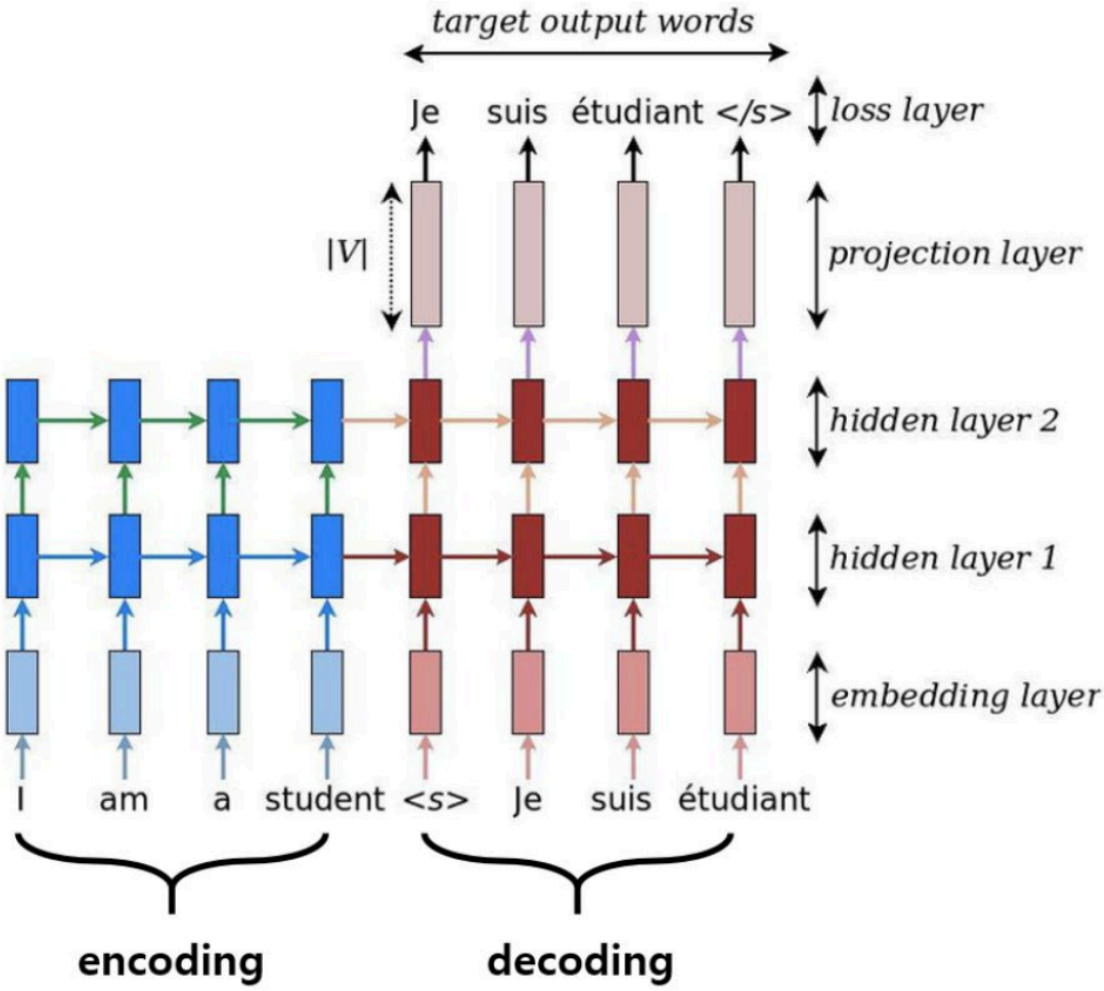


Figure 8. Detailed main back-end design.

3.2 Technologies and tools

To properly implement the described system, it is required to use appropriate tools and technologies. Since the system can be decomposed into several subsystems, the author would describe tools with the respected grouping.

Firstly, on the front-end, an ordinary long poling algorithm of content refreshing is used and nothing complex like WebSockets is implemented. The UI contains several views and almost no complex interactions, so it is enough to use just plain HTML/CSS/JS stack instead of the full-fledged framework solution. Nevertheless, it is more convenient to use some templating engine to bind data extracted from the database to the final HTML files,

so it was decided to use Jinja (Pallets, 2010). EcmaScript 2015 is used as the JavaScript standard to ensure a high level of compatibility with browsers. *Editor.js* library is used to implement a WYSIWYG editor inside the client application.

Secondly, the main backend is built with the use of *Python 3.9* and Flask as the framework. Justification for the latter is the small amount of code overhead required to bootstrap a prototype and a variety of available community-driven packages that cover all of the system needs. API handling is provided by Flask's built-in functionality, whereas the communication with the email providers is handled by the external *smtplib* and *imaplib* python libraries. Sending and receiving an email might be a time-consuming task, so Celery (Solem & contributors, 2018) was used as a task queue capable of handling a heavy asynchronous load. Also, there is a need to store already downloaded emails and all of the related content: generated replies, headers, and tags for which is enough to use PostgreSQL.

```
app = Flask(__name__)


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/poll_emails', methods=['GET'])
def poll_sensors():
    return render_template(dashboard.html, sensors=[])


@app.route(/ping, methods=['GET'])
def switch_button():
    return make_response("<h2>200 OK</h2>", 200)
```

Figure 9. A part of the server-side Flask API code.

As far as the neural networks are concerned, tools used to implement both of them are

similar in order not to make it harder to maintain. TensorFlow is used as a framework for building the neural networks and the architecture was built using such pre-built components as LSTMCell, DropoutWrapper, bidirectional_dynamic_rnn, etc. The example of the LSTM encoding layer can be seen in figure 10.

```
def encoding_layer(rnn_size, sequence_length, num_layers, rnn_inputs, keep_prob):
    for layer in range(num_layers):
        with tf.variable_scope('encoder_{}'.format(layer)):
            cell_fw = tf.contrib.rnn.LSTMCell(rnn_size,
                                              initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=2))
            cell_fw = tf.contrib.rnn.DropoutWrapper(cell_fw,
                                                    input_keep_prob=keep_prob)

            cell_bw = tf.contrib.rnn.LSTMCell(rnn_size,
                                              initializer=tf.random_uniform_initializer(-0.1, 0.1, seed=2))
            cell_bw = tf.contrib.rnn.DropoutWrapper(cell_bw,
                                                    input_keep_prob=keep_prob)

            enc_output, enc_state = tf.nn.bidirectional_dynamic_rnn(cell_fw,
                                                                    cell_bw,
                                                                    rnn_inputs,
                                                                    sequence_length,
                                                                    dtype=tf.float32)


    enc_output = tf.concat(enc_output, 2)

    return enc_output, enc_state
```

Figure 10. The example of the LSTM encoding layer.

In addition, for the data changes tracking and experiment checkpoints DVC (Iterative Inc., 2021).

3.3 User interface

As the user interface is concerned, most of the competitors are constantly working on improving it along with the user experience, because it dramatically affects the users' involvement. The current subchapter is intended to give the overall view of the UI implemented in the system.

3.3.1 Use cases

Overall main system use cases can be seen in figure 11.

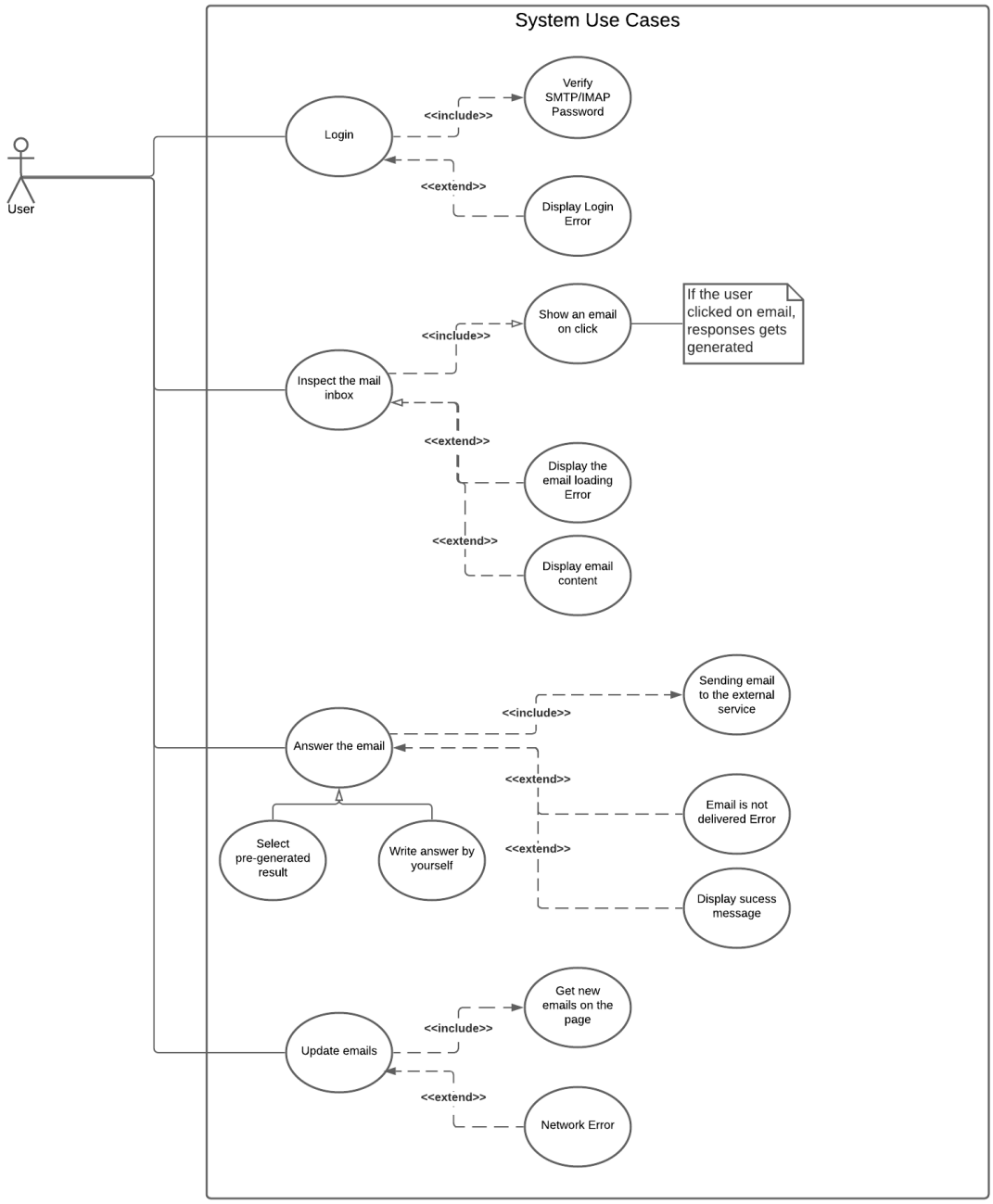


Figure 11. **Login view of the client-side application.**

A detailed description of the use cases:

1. When the user first opens the application, the Login View is shown. The user logs into the application with the login/pass pair or registers using this information.

Then the application asks to provide the access to the email inbox using credentials from the email provider (e.g., Mai.ru, Yandex, Google, etc.), the incoming and outgoing mail server addresses required by IMAP protocol. On the successful login, the user is redirected to the Dashboard View, where all the messages from the inbox are shown. The application then starts to fetch all the emails that are located in the mail inbox of this account. Fetching is a long, ongoing process and the notification is sent to the user when it is finished. Current progress in percentages is displayed. SMTP subscription is created and any new message (not old ones) received is supplemented by the notification.

2. The user can view the already-fetched messages by clicking on their name/preview. In this case, the user gets directed to the Email View, where the message content is displayed and the “answer” button is located. By pressing the answer button, the user is directed to the Email Answering View page and there the message entering textarea is shown, under which there are several (e.g., three) generated answers. By clicking on the generated answer, its content is displayed in the message entering textarea. The user can make any corrections/add any kind of text/attachment he wants and then click the “send” button to send the message.
3. The user can manually set the greetings and signatures on the Settings View that are then automatically added to any of the replies the user wants to send (even the auto-generated ones).
4. The user can pin any message he wants by pressing on the email menu on the Dashboard View and selecting “important message”. Then those messages are displayed on the top of the email feed on the Dashboard View in a separate section.
5. The user can attach the files to the email on the Email Answering View by clicking on the “attach file” button. The limits are 20MB for one message by default, but if the email provider supports uploading the attaches to the cloud storages (e.g., Mail.ru uploads files to the proprietary cloud), then the file would be sent to the target receiver by an HTTPS-accessible download link.

3.3.2 UI guide

In figure 12 the initial screen displayed for the user is depicted. Here the user is asked to enter the credentials of the IMAP inbox and the SMTP Outbox, namely: username,

password, server address, and port. If something is not provided, the back-end would try to fall back on the defaults (e.g., IMAP 993 port, SMTP 587 port, etc.), but if there is no possibility to connect to the defined servers, then the error would be shown right on this page.

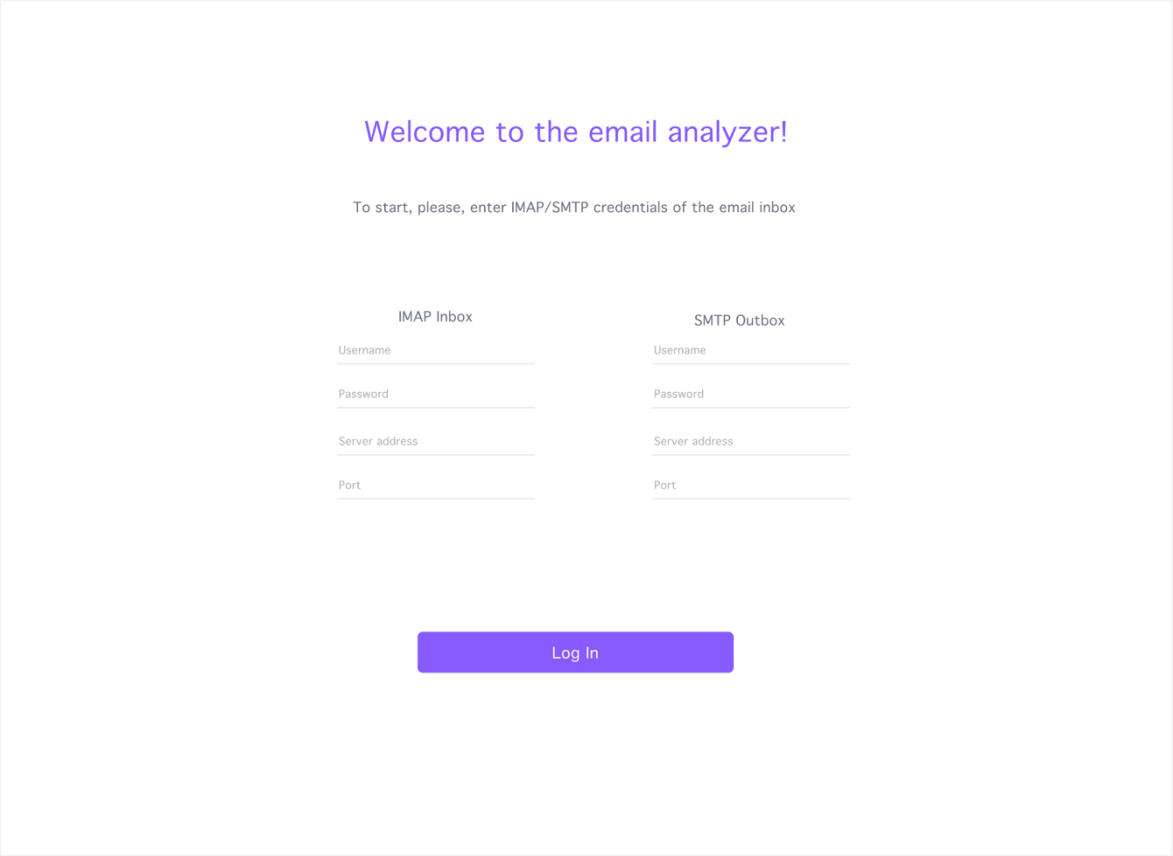


Figure 12. Login view of the client-side application.

The next view the user sees is depicted in figure 13. This is called the main dashboard and it consists of three columns. The first column contains folders used inside the app: ordinary emails, ones that the user has sent himself, important ones, unsent emails, and thrown away. There is also a possibility to log out of the account by pressing the “Log out” button.

The second column consists of a header with a search bar and filters followed by the list of emails downloaded from the email server. Brand new emails, that are not yet read are marked with the blue line, which would disappear when the user clicks on the specific email. Emails detected as important ones contain the bookmark sign on them.

The last column renders the HTML-formatted content of the email along with metainformation and action buttons (reply, delete, show next email). Attachments are displayed as a separate row inside this column. The user can download them all at once.

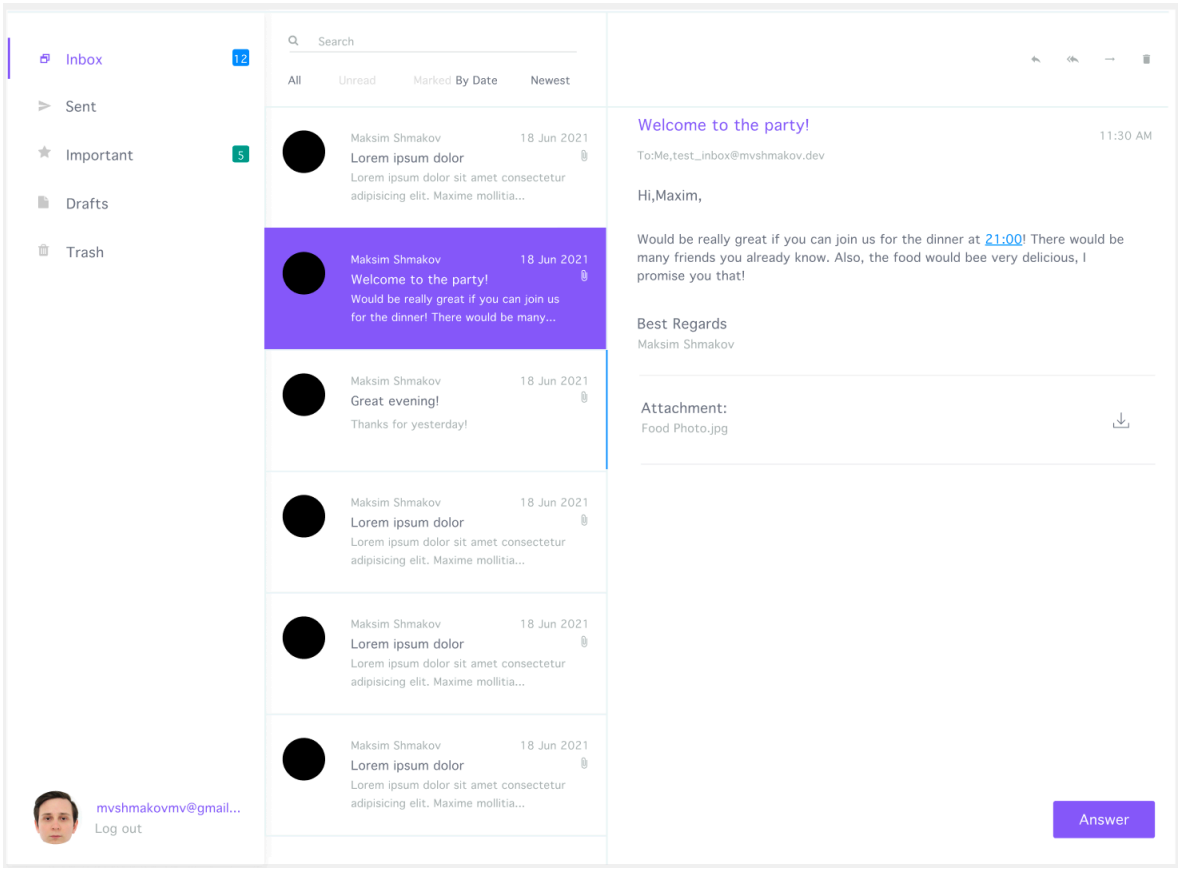


Figure 13. Main dashboard of GUI of the client-side application.

When the user clicks the "Answer" button, he gets directed to the next view shown in figure 14. This view contains two columns: the first one is the menu described in the previous paragraph, the second one includes the reply editing functionality. The application shows recipients and the subject, which is possible to regenerate from the source email content summarization. The user can either select any of the generated responses or write a manual one if nothing is suitable or even makes sense. In addition, there is an ability to supply the email with the attachment up to 20MB.

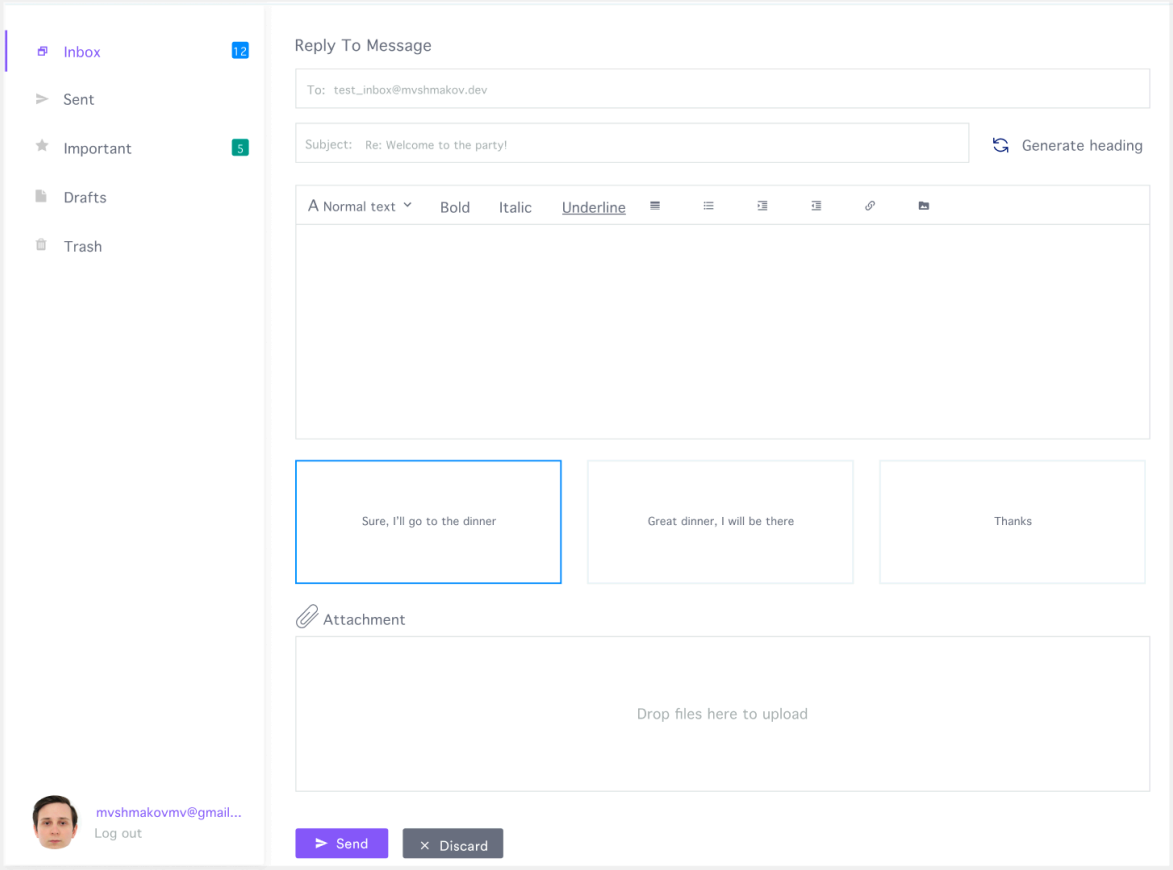


Figure 14. Email reply view of GUI of the client-side application.

3.4 Deployment

One of the non-functional requirements is the need to yield an available self-hosting solution. To satisfy this requirement, it was decided to wrap system components in Docker containers supplied with the *docker-compose.yaml* file, so the developed system would be containerized and host system-agnostic. A simple *docker-compose up* command would spin up the whole system due to proper configuration (see a part of the configuration file in figure 15).

Figure 15. Part of the *docker-compose.yaml* file.

During testing, the system was deployed on the Heroku platform (Middleton & Schneeman, 2013) by using free-tier dyno hours and running through the *docker-compose*. The deployment diagram of the working system looks like in figure 16.

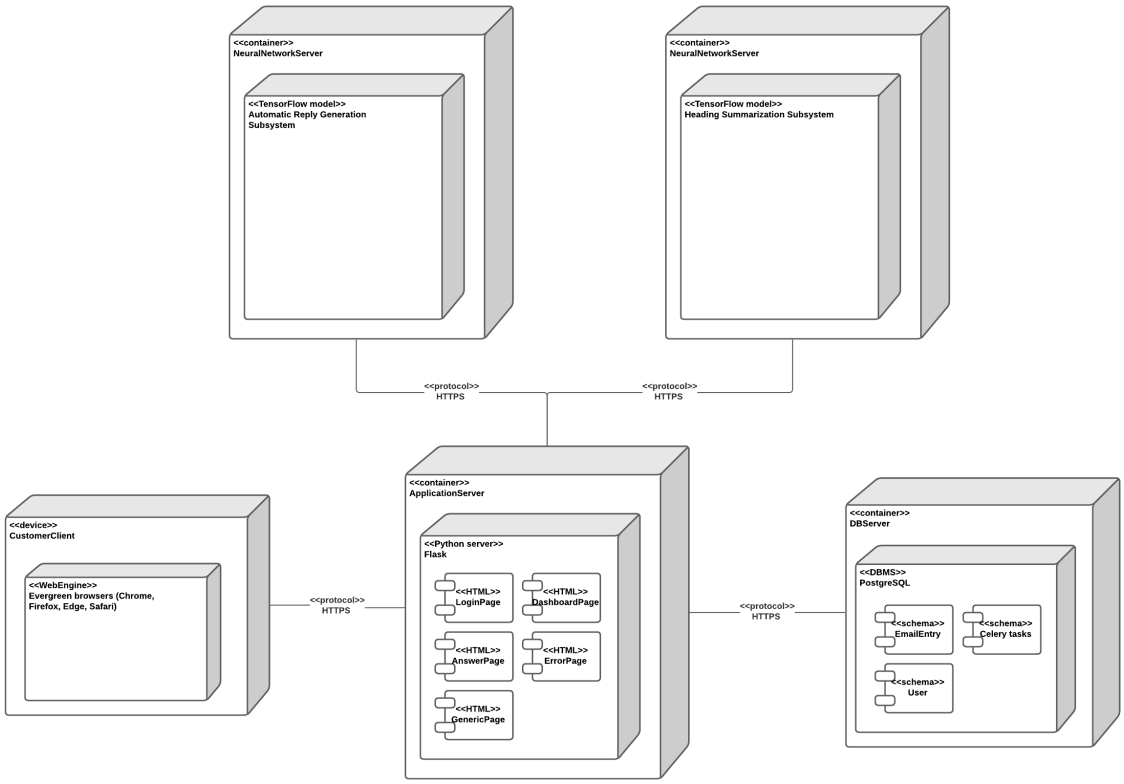


Figure 16. Deployment diagram of the system.

4 SYSTEM TESTING AND EVALUATION

The system was tested against the key functional and non-functional requirements and all of them were satisfied. Functional requirements were tested by manually going through the list and trying to implement the functionality with real-world data. Real email servers and email account were used and the application was reacting on the real-world data.

Some of the non-functional requirements were verified during the system implementation and deployment, like maintainability and scalability. In addition, load testing was performed with the Yandex.Tank (Yandex LLC, n.d.) load testing tool, and the results were less than 700ms for 10 RPS metric as was stated in the requirements.

4.1 Limitations of the testing

Despite the tests were conducted and the system is formally satisfying the requirements based on those, the tests are artificial since no real users are present in the system. Thus, the first public system launch might still be problematic due to unrevealed issues.

In addition, there is no way to test all of the possible emails, so the errors during the summarization and response generation are still possible.

5 DISCUSSION AND CONCLUSIONS

Summing everything up, the initial goal and all of the objectives are satisfied by this research. An overview of the current state of competition in the field of email assistants and core approaches is provided. The functional and non-functional requirements of the initial system proposal were satisfied during the work on this paper. The analysis of the existent solutions and modern network approaches was conducted leading to the system design proposal. The resulting system design was evaluated and tested during the actual system implementation phase.

Although the approaches used in this paper are not the cutting-edge ones, they are yielding pretty good results. One of the main points of improvement is to use other more powerful models like transformers, but for this experiment, sufficient funding is needed.

Unfortunately, there was not enough time to make the system to production, because of the thesis time limit and the amount of work to finish the current state.

5.1 Future research suggestions

First, since the proposed system currently exists just as a proof-of-concept, it is a great idea to deploy it to the production environment and start testing it on real users. This should reveal potential performance and security issues discovery and bugs; fixing those would increase the overall quality of the system.

Considering the models' accuracy, as discussed, on the market currently more performant neural network architectures are arising, but the cost of training such neural networks is much higher than the ordinary student can afford. Replacing the NN with a more performant one may dramatically increase the user experience and the actual use of generated answers.

From the infrastructural point of view, the system is still using the long polling approach to fetch the emails, which is not the best way of doing that. WebSockets or Server Push technologies should be used in the future to handle this part of the application components

communication.

In addition, no research was conducted from the b2b sector, meaning the implemented system might not be the perfect fit for self-hosting. By researching with the target business users, more precise requirements can be elicited, and the respected features might be implemented in the system. For instance, implementing the functionality of regenerating the responses can be helpful for some slices of users.

SUMMARY

The key goal and all the objectives that were stated at the very beginning of the current thesis are all fulfilled as expected. The system architecture is proposed and verified. The appropriate tools that made it possible to implement the proof-of-concept are justified and properly used. Subsequent system testing is conducted, and the overall system is verified.

The current work also faces some problems that should be solved as a part of the future topic and software development.

REFERENCES

Accounts, C. E., Traffic, E., Sent, D. E., Email, A., Requirements, S., Spam, A., & Pacific, A. (2019). Email Statistics Report, 20 19 - 2023. *The Radicati Group*, *44*(0), 3.

Al-Khateeb, H. M., & Epiphaniou, G. (2016). How technology can mitigate and counteract cyber-stalking and online grooming. *Computer Fraud and Security*, *2016*(1), 14–18. https://doi.org/10.1016/S1361-3723(16)30008-2

Avinash, N. (2018, October 9). *Latent Semantic Analysis using Python - DataCamp*. Datacamp.Com. https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python

Dabbish, L. A., & Kraut, R. E. (2006). Email overload at work: An analysis of factors associated with email strain. *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, 431–440. https://doi.org/10.1145/1180875.1180941

Feng, Y., Naeem, M. A., Mirza, F., & Tahir, A. (2020). Reply using past replies—a deep learning-based e-mail client. *Electronics (Switzerland)*, *9*(9), 1–20. https://doi.org/10.3390/electronics9091353

Graham, T. (2020). *g-Tierney/stLDA-C_public: Single-topic LDA (DMM) with unsupervised clustering*. GitHub. https://github.com/g-tierney/stLDA-C_public

Hair, M., Renaud, K. V., & Ramsay, J. (2007). The influence of self-esteem and locus of control on perceived email-related stress. *Computers in Human Behavior*, *23*(6), 2791–2803. https://doi.org/10.1016/j.chb.2006.05.005

Iterative Inc. (2021). *Data Version Control · DVC*. https://dvc.org/

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*(1), 11–21. https://doi.org/10.1108/eb026526

Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Lukács, L., Ganea, M., Young, P., & Ramavajjala, V. (2016). Smart Reply: Automated response suggestion for email. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-Augu*, 955–964. https://doi.org/10.1145/2939672.2939801

Klimt, B., & Yang, Y. (2004). Introducing the Enron Corpus. *Machine Learning*, *stitutep1*, wwceasccaers2004168.

http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Introducing+the+Enron+Corpus#0

Lee, J., Kang, J. H., Jun, S., Lim, H., Jang, D., & Park, S. (2018). Ensemble modeling for sustainable technology transfer. *Sustainability (Switzerland)*, *10*(7). https://doi.org/10.3390/su10072278

Loper, E., & Bird, S. (2002). NLTK: The Natural Language Toolkit. *Arxiv.Org*. https://doi.org/10.3115/1225403.1225421

Middleton, N., & Schneeman, R. (2013). *Heroku: up and running: effortless application deployment and scaling*. O'Reilly Media, Inc. https://books.google.com/books?hl=en&lr=&id=IS4KAgAAQBAJ&oi=fnd&pg=PR2&dq=Heroku&ots=eFI0N5Njzz&sig=0ej_Jotw80zbvQVm5T3DIPZD1TU

Moubayed, N. Al, Mcgough, S., & Hasan, B. A. S. (2020). Beyond the topics: How deep learning can improve the discriminability of probabilistic topic modeling. *PeerJ Computer Science*, *6*, 1–32. https://doi.org/10.7717/PEERJ-CS.252

Nikolenko, S., Koltcov, S., & Koltsova, O. (2015). Topic modeling for qualitative studies. *Journal of Information Science*, *5*, 1–15. https://doi.org/10.1177/0165551515617393

Pallets. (2010). *Templates — Flask Documentation (2.0.x)*. Flask Documentation. https://flask.palletsprojects.com/en/2.0.x/tutorial/templates/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ramos, J. (2003). Using TF-IDF to Determine Word Relevance in Document Queries. *Proceedings of the First Instructional Conference on Machine Learning*, *242*(1), 29–48.

Sathiyanarayanan, M. (2020). *Mithileysh/Email-Datasets: Email Datasets can be found here*. GitHub. https://github.com/Mithileysh/Email-Datasets

Schiller, Z. L. (2015). Email Similarity Matching and Automatic Reply Generation Using Statistical Topic Modeling and Machine Learning. *Electronic Theses and Dissertations*, *2379*.

scikit-learn developers. (2020). *6.2. Feature extraction — sci-kit-learn 0.24.2 documentation*. Scikit-Learn. https://scikit-

learn.org/stable/modules/feature_extraction.html

Sites.google.com. (n.d.). *Datasets - Email Research*. Google Sites. Retrieved June 21, 2021, from https://sites.google.com/site/emailresearchorg/datasets

Solem & contributors. (2018). *Celery - Distributed Task Queue — Celery 5.1.1 documentation*. https://docs.celeryproject.org/en/stable/#

Tomlinson, R., & BBN Technologies. (1971). The First Network Email. *BBN Technologies*, 10–11. http://openmap.bbn.com/~tomlinso/ray/firstemailframe.html

Ulrich, J., Carenini, G., Murray, G., & Ng, R. (2009). *Regression-Based Summarization of Email Conversations Introduction and Related Work*. http://www.cs.ubc.ca/labs/lci/bc3.html.

Ulrich, J., Murray, G., & Carenini, G. (2008). *A Publicly Available Annotated Corpus for Supervised Email Summarization*. http://caloproject.sri.com

Whittaker, S., & Sidner, C. (1996). Email overload: exploring personal information management of email. *Conference on Human Factors in Computing Systems - Proceedings*, 276–283. https://doi.org/10.4324/9781315806389-24

Yandex LLC. (n.d.). *Yandex/Yandex-tank: Load and performance benchmark tool*. Retrieved June 21, 2021, from https://github.com/yandex/yandex-tank

Yonghui, W. (2018). *Google AI Blog: Smart Compose: Using Neural Networks to Help Write Emails*. Google AI Blog. https://ai.googleblog.com/2018/05/smart-compose-using-neural-networks-to.html

Zhang, R., & Tetreault, J. (2019). *This Email Could Save Your Life: Introducing the Task of Email Subject Line Generation*. https://github.com/

Zhong, M., Liu, P., Chen, Y., Wang, D., Qiu, X., & Huang, X. (n.d.). *Extractive Summarization as Text Matching*. Retrieved June 21, 2021, from https://github.com/