ELEC 490/498 Final Report: Extractive Summarization of Chat Box Dialogue

Submitted By: Group 12

Veerasundar, Shrish

Maduabuchi, Iffy

Rao, Hai-Ling

Edhlund-Roy, Jaden

Faculty Supervisor(s): Xiaodan Zhu

Executive Summary

The purpose of this project is to design a tool which can perform extractive summarization on chat box dialog using machine learning models. Chat bots have become increasingly common in online platforms and customer services, so there arises a need to efficiently obtain the key points of any given dialogue to optimize analyses. The tool is intended to provide more privacy, visualization, and a client-centric focus to summarization than that provided by tools such as ChatGPT.

The goal of the tool is to perform extractive summarization of text files or raw text, where the summarization results indicate which messages contributed to the summary. The tool was designed as a Frontend Component (FEC) and Network Component (NC) which handles user input and summarization processing respectively, independently from one another.

The tool implementation closely follows this design approach and is implemented as a web application. The frontend of the web app represents the FEC and was implemented with React.js and CSS. It handles user input and converts them into an API request sent to the backend, which represents the NC. The API is built with Flask and services requests by invoking the implemented network component object that contains the summarization model. The network component uses the model to perform the summary, and packages the model output for the FEC. The FEC renders the response as a full summary paragraph and a chat pane of the input transcript which highlights messages used in the summary.

The ML model is implemented with PyTorch and is an extension RoBERTa, the robustly optimized BERT language model. The model identifies the *k* most important sentences in the dialogue, which are combined to form the summary. Sentence importance is determined using ROUGE scores, a training metric that compares the model's selected sentences against sentences selected for a reference summary. This was used with MSE loss and the SAMSum dataset to train the model.

The tool was able to meet speed requirements, with an end-to-end response time of 10 seconds. The FEC and NC were tested and verified to appropriately handle erroneous user input and requests, as well as block malicious attacks. As for the model, its performance was evaluated using ROUGE metrics, where it achieved a recall of 0.32, precision of 0.17 and f1 measure of 0.22. This is significantly below SOTA performance, which averages 0.4 to 0.5 in f1.

While the tool satisfies the defined goals, there is room for improvement. The web app GUI can be optimized for mobile devices, and API performance can be improved by resolving resource contention in request servicing. The training pipeline and loss function of the summarization model can also be refined to improve its summarization quality, while more capable hardware can improve its processing speed.

Table of Contents

1 Motivation and Background

1.1 Motivation

Chatbox dialogues have become increasingly common in various online platforms, be it conversations with your bank advisor, customer service, or even a conversation with your insurance provider [1]. Research indicates that chatbots are already very prevalent and will continue to be more prevalent as time goes on. In fact, the government of Canada recently invested $1.4M in Proto Research Inc, a company that builds multilingual chatbots used by central banks, clinics, and companies in the digital financial services space [2]. According to Juniper research, between 75-90% of queries in healthcare/banking sectors are estimated to be handled by automated chatbots in the next 5 years saving about $0.70 per interaction [3]. With the increasing amount of information, there arises a need to have a tool that can summarize the key points in these conversations. Extractive summarization is a technique that can extract essential information from a text and present it in a concise form [4].

This project describes a tool that the team has built to assist in extractive summarization of chatbox dialogue using machine learning models such as BERT. These models have been shown to be effective in natural language processing tasks, making them a suitable option for building an extractive summarization tool [5].

While there are existing tools in the market such as ChatGPT [6] that can perform a similar function, this project aims to build a summarizer with privacy in mind, and a very client-centric focus. This could mean training the dataset on topics that the client is interested in, as opposed to ChatGPT, which is trained on a large set of data. This allows the project to be informationally efficient while also providing a good base for the machine learning models to perform summarization.

Some traditional challenges faced when performing summarization tasks include but are not limited to inability to handle complex language structures such as idiomatic expressions, sarcasm, or irony, leading to incomplete or misleading summaries [7]. Another challenge could be lack of context understanding, which could lead to inaccurate or irrelevant summaries [8]. Using machine learning models such as BERT can help overcome these challenges by providing more coherent and accurate summaries of chatbox dialogues. The underlying machine learning model can be trained to identify the most significant points in the conversation depending on the training dataset and present them in a way that is easily digestible to the users.

2 Project Design

2.1 Requirements

The goal of the project is to use a machine learning model trained on extractive summarization to summarize and highlight key topics of a given chat dialogue. Usage of the model will be facilitated through a graphical user interface, and the summary must be visualized in such a way that it highlights messages in the source chat dialogue that were used to generate the summary.

The tool should support file uploads and direct text input for maximum user accessibility. Additionally, the summarization and visualization process should limit resource usage on user devices while being completed in no more than 10 seconds.

Table 1 states these goals as the functional, interface and performance requirements of the tool.

Table 1 - System Specifications

1	**Functional requirements**
1.1	Extractive Summarization of chat dialog (transcript between 2 entities)
1.2	Visualization of results to highlight summary sentences within dialog transcript
2	**Interface requirements (A GUI prototype is included, see Figure 19 in the Appendix)**
2.1	Users can upload dialog transcript file
2.2	Users can paste chat transcript text
2.3	Summarization Rendered as a chat conversation with key sentences highlighted.
3	**Performance requirements**
3.1	End-to-end response time within summarization and processing tolerances (10 seconds)
3.2	Limit user resource usage expand applicability to wide range of devices

2.2 Design Approach

The design approach was to build the tool/application as two distinct components that communicate with each other. These are the Network Component and the Frontend Component.

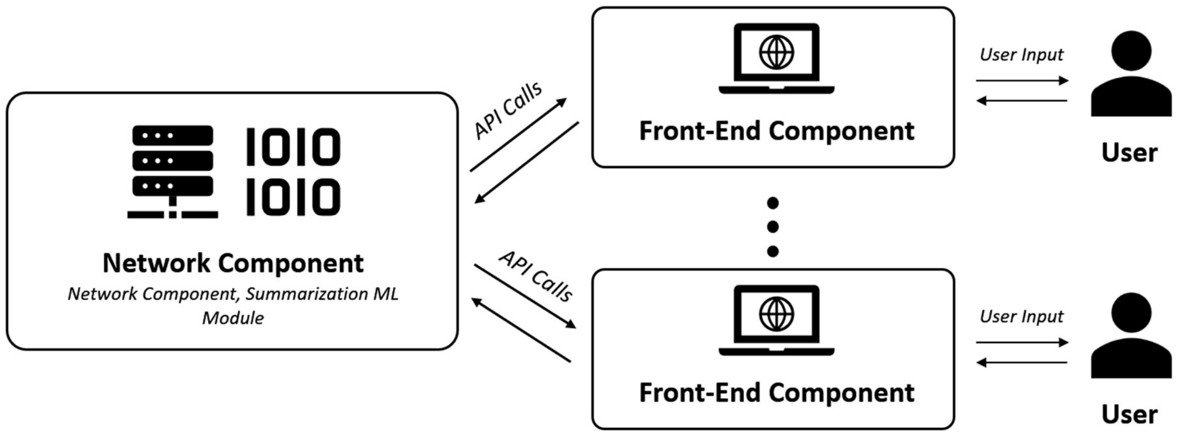


Figure 1 - High-level Application Architecture

The Network Component (NC) houses the machine learning model and provides wrapper functions and classes to encapsulate the model and facilitate summarization requests to it. It parses summarization requests into the model input, and processes the model output into request responses. This encapsulation approach abstracts the model's implementation details, allowing it to be modified without requiring changes to external systems.

The Frontend Component (FEC) serves as the interface between the user and the network component. It provides the required graphical user interface and handles translating user input into requests sent to the NC. The GUI is also responsible for rendering the request responses to satisfy the system interface requirements.

The communication protocol between the components is standardized and easily configurable, easing the addition of more functionality to the application. To implement a feature, the new content in the protocol will be determined and can intra-component changes can be implemented independently.

A client-server architecture was also chosen to reduce device resource usage for users. User devices only run instances of the frontend component, which make calls to a remote network component running on a suitable server. For higher volume demands, multiple network components can be used and FECs will instead make calls to a server which handles dispatching requests to the network components.

2.3 Work Distribution and Breakdown Structure

The project was broken down and assigned to team members using a primary and secondary manager approach. The primary manager was directly responsible for their assigned task while the secondary manager acted as help to the primary manager when required.

Table 2 lists the primary and secondary manager assignments for the project.

Table 2 – Project Work Distribution

Project Aspect	**Primary Manager**	**Secondary Manager**
Frontend Component	Iffy Maduabuchi	Shrish Veerasundar
Network Component API	Iffy Maduabuchi	Jaden Edhlund-Roy
Network Component Model Encapsulation	Jaden Edhlund-Roy	Hai-Ling Rao
Machine Learning Model	Hai-Ling Rao	Jaden Edhlund-Roy

A work breakdown structure was also designed by the team to address the main design phases of the project: Research and Design Specification, Component Implementation, Component Unit Testing, Component Integration, and Integration Testing. These were further broken down into specific problems and tasks assigned to each team member. The WBS diagram can be found in the Appendix (Figure 18).

2.4 Design Process

Project development began by defining the basic design approach to build the tool as two independent components. After researching available tools, frameworks and models for the problem, the desired specifications were identified, and the basic design approach was refined.

The communication protocol between the components was implemented first, which allowed for concurrent development of the FEC and NC. Later, the two branches of development were merged for component integration and integration testing.

2.4.1 Implementing the Frontend Component and Component Communication

To satisfy the client-server architecture discussed in the design approach, the application was implemented as a simple web application. The FEC would be the application frontend, with the NC component running on the backend. A web application implementation also simplifies implementing component communication since standard REST APIs [9] can be used.

2.4.1.1 API

Development began with the API as it determines the structure of the communication protocol, which once defined would allow for the concurrent development of the FEC and NC. Flask [10], a python micro-web framework, was chosen to implement the API.

Flask was chosen for two main reasons. Since it is a framework implemented in Python, the model and NC can be invoked and run in the API framework directly, instead of having to implement another messaging protocol between the API and the NC. Secondly, Flask is a micro-web framework, only being suitable for handling HTTP [11] API requests and serving static HTML pages. This makes it an optimal solution for the API, as other frameworks (such as Django [12]) can provide more functionality that is not needed and would bloat the tool.

The Flask API was implemented as part of the network component. The initial implementation consisted of the API routes, as well as a set of static HTML pages that were designed to satisfy the interface requirements for the application (Figure 20 in the Appendix). The HTML pages were used to test the functionality of the API calls (handling requests and rendering responses) and familiarized the team with HTML, preparing them for the web-based GUI for the FEC.

JSON [13] was used within the HTTP requests, forming the communication protocol. It was chosen as it is easily modified by editing its key-value pairs, satisfying the desired configurability for the protocol. For a given new feature or change, the JSON would be modified as required and feature support can be implemented in the components independently based on the modified JSON.

2.4.1.2 FEC Basic GUI

With the API implemented, the frontend component would have to implement the GUI to take in user input and convert them into requests sent to the NC API. The FEC would also have to be implemented as a web frontend component, following the web application implementation.

This makes React.js [14] an excellent candidate for implementing the frontend component. React.js is a JavaScript front-end framework for building web applications. With React, web sites are built as a single page with dynamic elements, instead of loading new pages for user input. Implementing the FEC with React allows it to be as dynamic as an on-device application while minimizing usage of device resources as it is still a website.

The first implementation of the FEC web app was a functional skeleton GUI in React (Figure 21). This included the HTML elements required to take in user input and satisfy interface specifications as well as the underlying functionality to package requests for the NC API and render the API responses. The GUI was not well designed as the goal was to implement the functionality of the application first, then add styling and design afterwards.

2.4.1.3 FEC Final GUI

The final stage of implementing the frontend GUI is to style the web app with CSS to achieve the user-friendly interface. CSS [15] is a styling language for HTML, which can be added as attributes for HTML elements. This allows the interface design for the web app to be completed after implementing functionality, as the CSS can be written independently and then applied to the elements by adding it as an attribute.

First, the initial GUI mock-up discussed in the Blueprint Report (Figure 19 in the Appendix) was refined to include and highlight the HTML elements used in the React app (Figure 22 in the Appendix). Additional mock-ups were also created to show different states of the web app (loading, no content, errors, etc.). These can be found in Section 10.2 in the Appendix.

The mock-ups were used as reference to write the appropriate CSS for each element in the web app, completing the implementation of the frontend GUI and therefore the frontend component.

2.4.2 Implementing the Summarization Model

The development of an extractive dialogue summarization model involved a multi-stage iterative process and was performed concurrently with the development of the FEC.

The team consulted the team supervisor, who identified the SAMSum dataset [16], a dataset containing handwritten dialogues and corresponding abstract summaries, as an ideal resource for training the model.

RoBERTa [17], a robustly optimized BERT model, was chosen over other natural language models such as GPT-3 due to its better performance on Natural Language Understanding (NLU) tasks. This is understood to be partly due to the base BERT employing bidirectional masking unlike GPT, which is unidirectional [17].

With more consultation from the team supervisor, ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [18] scores between a dialogue sentence and dialogue abstract summary were chosen to fine-tune the model. ROUGE scores are a metric used to evaluate the quality of summaries automatically generated by AI or algorithms, by comparing them with human-generated reference summaries. They measure the overlap of words or phrases between the generated summary and the reference, with higher scores indicating better match and thus better performance of the summarizing system.

Through a series of iterative refinements, the model selects k sentences as the final output, providing concise and coherent dialogue summaries. The final design represents the culmination of a data-driven approach that leveraged cutting-edge natural language processing techniques and technologies.

2.4.3 Implementing The Network Component

The design process for the network component has three main requirements for its implementation. These are to receive JSON object packages from the calling API, call the extractive summarization machine learning model, and return summary messages to the API.

The NC was designed as an OOP class to facilitate multiple instances of the component to be invoked by the API. This allows for increased program flexibility and running multiple summarization requests in parallel.

The network component class was designed to receive a passed JSON chat package, isolate the transcript messages for summarization, and call the summarization model on the extracted chat messages.

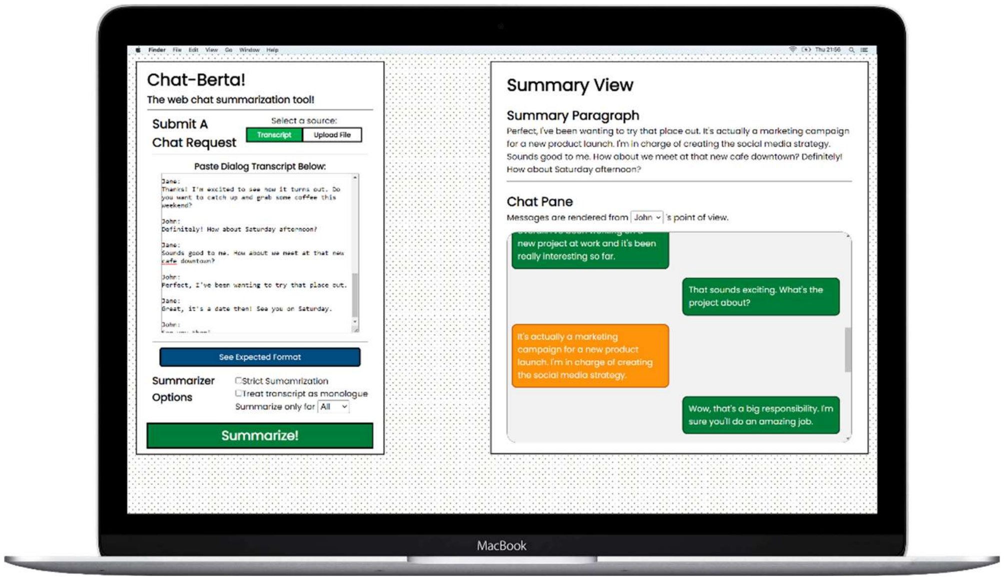
Once the model performs the summarization computations and returns the top sentences to be used in the extractive summary, the NC will update the JSON package with a new field containing the summary messages to be returned to the API.

2.4.4 Component Integration

Integration of the NEC and FEC was made simple since the structure of the JSON used in communication was standardized and supported by both components. Integration was achieved by invoking the network component within the Flask API route to handle the summary request package. Since the NC returned information in JSON, it simply had to be included in the API response to the FEC.

3 Solution

The summarization tool is implemented as a web app (as discussed in Design Process) allowing it to be accessible on any device with a browser. The layout of the web app is shown in Figure 2 below.



Chat-Berta!

The web chat summarization tool!

Submit A Chat Request

Select a source:

Paste Dialog Transcript Below:

```
Jane:
Thanks! I'm excited to see how it turns out. Do
you want to catch up and grab some coffee this
weekend?

John:
Definitely! How about Saturday afternoon?

Jane:
Sounds good to me. How about we meet at that new
cafe downtown?

John:
Perfect, I've been wanting to try that place out.

Jane:
Great, it's a date then! See you on Saturday.

John:
```

Summarizer Options

☐ Strict Sumamrization

☐ Treat transcript as monologue

Summarize only for: All

Summary View

Summary Paragraph

Perfect, I've been wanting to try that place out. It's actually a marketing campaign for a new product launch. I'm in charge of creating the social media strategy. Sounds good to me. How about we meet at that new cafe downtown? Definitely! How about Saturday afternoon?

Chat Pane

Messages are rendered from John 's point of view.

new project at work and it's been really interesting so far.

That sounds exciting. What's the project about?

It's actually a marketing campaign for a new product launch. I'm in charge of creating the social media strategy.

Wow, that's a big responsibility. I'm sure you'll do an amazing job.

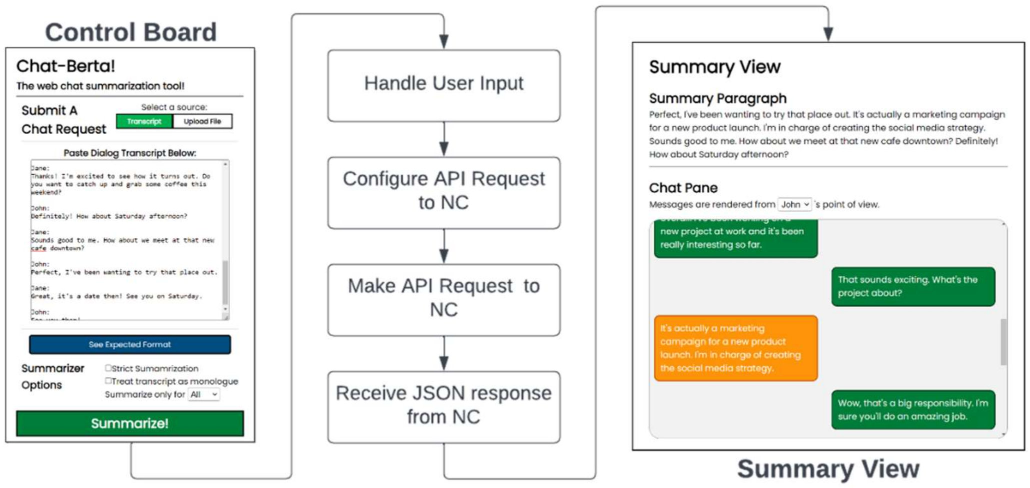
Figure 2 - Summarization Tool Web App

The frontend component and network component are hosted as two separate cloud servers, chat-berta.vercel.app and chat-berta-api.vercel.app respectively. Users visit chat-berta.vercel.app to use the summarization tool, which makes calls to the API server for summarization request[1].

3.1 Frontend Component

The FEC is the React.js web application. It consists of two main components: the Control Board and the Summary View. The Control Board receives the user input and makes the request the NC API, while the Summary View renders the API response in the GUI for the user. Figure 3 visualizes this control flow.

[1] Size limitations on the cloud hosting service have broken production deployments for the NC and FEC. The size limitations were only encountered at the very end of the project due to the model needing to be deployed as well. The application works appropriately on development servers.



```
Jane:
Thanks! I'm excited to see how it turns out. Do
you want to catch up and grab some coffee this
weekend?

John:
Definitely! How about Saturday afternoon?

Jane:
Sounds good to me. How about we meet at that new
cafe downtown?

John:
Perfect, I've been wanting to try that place out.

Jane:
Great, it's a date then! See you on Saturday.

John:
```

Perfect, I've been wanting to try that place out. It's actually a marketing campaign for a new product launch. I'm in charge of creating the social media strategy. Sounds good to me. How about we meet at that new cafe downtown? Definitely! How about Saturday afternoon?

new project at work and it's been really interesting so far.

That sounds exciting. What's the project about?

It's actually a marketing campaign for a new product launch. I'm in charge of creating the social media strategy.

Wow, that's a big responsibility. I'm sure you'll do an amazing job.

Figure 3 - Control Board and Summary View Control Flow

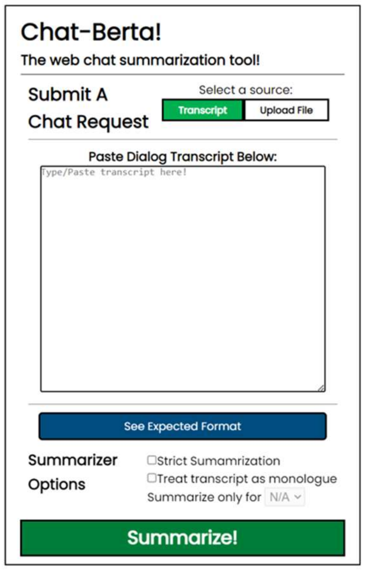
The Control Board and Summary View independent React components (*ControlBoard.js and SummaryView.js*) which are placed within *App.js*, the highest-level component that represents the web application. Figure 24 in the Appendix shows the implemented code for the organization.

3.1.1 Control Board

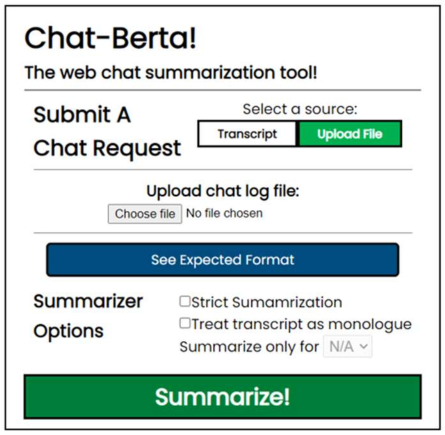
3.1.1.1 Input Sources

Within the control board is the *SubmitChat.js* component, which contains all the user input elements and makes the summary request to the network component.

Users can select between uploading a transcript text file (*Upload File*) or pasting the raw transcript (*Transcript*) directly into the web app using the *Select a source* toggle at the top of the control board. The appropriate toggle button for a source option becomes green when it is selected by the user. Figure 4 shows the states of the control board based on the selected source.



Paste Transcript Text



Upload A File

Figure 4 - Control Board Input Source Options

File uploads are restricted to .txt files to prevent attacks by uploading malware or unidentified file types. Similarly, the contents of the transcript text box are never executed to prevent attacks by embedding malicious HTML and/or JavaScript.

3.1.1.2 Parsing The Transcript

For both input sources, the transcript text is extracted and parsed to identify the messages and parties involved in the chat dialogue. This is done using the function *chatTextToChatJSON* which receives a given text and attempts to parse it into the JSON structure used when sending the request to the Network Component.

The JSON structure is referred to as the chat package. Figure 5 shows the standard format of a chat package.

```
{
    "config": {
        "parties": [
           {
                "id": 0,
                "name": "John"
            },
            {
                "id": 1,
                "name": "Jane"
            }
        ]
    },
    "messages": [
        {
            "id": 0,
            "pid": 0,
            "text": "Apples are my favourite fruit, what are yours?"
        },
        {
            "id": 1,
            "pid": 1,
            "text": "I like oranges better, apples be gross sometime"
        },
        . . .
    ]
}
```

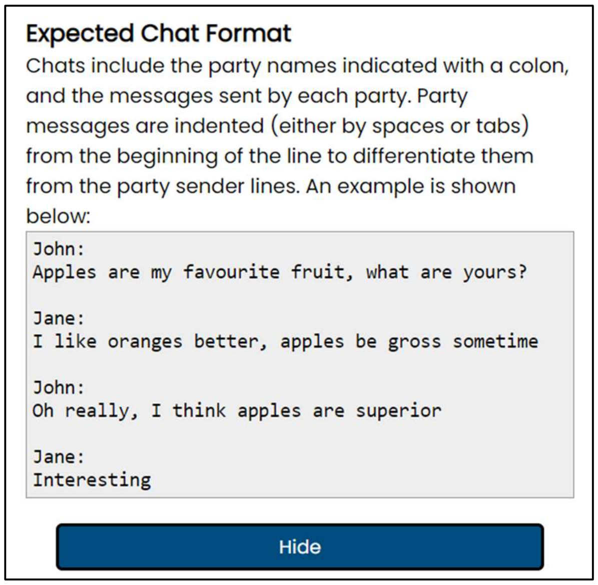
Figure 5 - Standard Format of Chat Package

The chat package has the *config* field, which contains *parties*, a list of parties that were involved for the given chat dialogue. Each party is assigned a party ID, that is used to track the messages they contributed to the chat dialogue.

There is also the *messages* field, which is a list of all the messages sent in the chat dialogue. Each *message* object has a message ID (*id*), a party ID (*pid*) for which party sent the message, and the associated text of the message.

The chat package is embedded as its own field (*chat_package)* within the JSON request sent to the network component.

Users must be made aware of the expected transcript format, as the parsing in *chatTextToChatJSON* is only successful if the input text is in the correct format. This is provided with the *ExpectedTranscriptFormat.js* component (as seen in Figure 6) which is rendered when users click the *See Expected Format* button.



Expected Chat Format

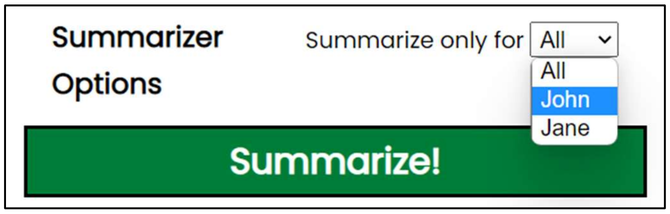
Chats include the party names indicated with a colon, and the messages sent by each party. Party messages are indented (either by spaces or tabs) from the beginning of the line to differentiate them from the party sender lines. An example is shown below:

```
John:
Apples are my favourite fruit, what are yours?

Jane:
I like oranges better, apples be gross sometime

John:
Oh really, I think apples are superior

Jane:
Interesting
```

Figure 6 - Rendered ExpectedTranscriptFormat.js Component

3.1.1.3 Summarization Options

Once a valid transcript is uploaded, users can optionally change the nature of the summarization by editing the summarization options. The main summarization option is to selectively summarize the chat dialogue for only a specific party.



Summarizer Options

Summarize only for: All

Figure 7 - Summarization Options

The party options in the dropdown are populated from the *parties* field in the chat package, which is obtained by calling *chatTextToChatJSON* on the given transcript text. The party options are repopulated on any change to the source transcript (i.e. changes to the transcript text box, or a new uploaded file). If no valid transcript text is detected, the dropdown is disabled in the web application.

Users can keep the dropdown as *All* if they desire no specific party to be summarized, otherwise they can select the name of the party from the dropdown. The option is packaged in the API request as a *summarize_only_for* field within the main *summary_options* field (Figure 8). If a specific party is selected, their party ID is the value of the field, otherwise it is set to -1.

```
"summary_options": {
        "basic_options": []
        "summarize_only_for": -1
    }
```

Figure 8 - Summary Options Field in the API Request

3.1.1.4 Sending the Summary Request

Users can send a summary request by clicking the *Summarize!* button. This triggers validation of the user input before making the API request to the network component. The validation checks include:

- Ensuring that a text source is selected.
- Ensuring that a valid transcript text is detected (i.e. can be parsed by *chatTextToChatJSON*).

Input validation ensures that all requests sent to the NC API are correct. If the validation fails, the web application responds with an alert indicating the issue (see Web App Request Validation Failure Messages in the Appendix).

If input validation passes, then the *chat_package* and *summary_options* field discussed previously are included as the JSON body of an HTTP API request made to the URL *chat-berta-api/api/submit-chat.* This is done using the function *apiJSONFetch* which takes the route for the API, the HTTP method, and the JSON body to be sent. The function abstracts configuring and sending the HTTP request, as well as retrieving the JSON in the HTTP API response.

Sending the API request and receiving the response can take a user perceivable amount of time. To accommodate for this, a loading page is rendered in the Summary View (as shown in Figure 9) to provide users with an indication that their request is being processed.

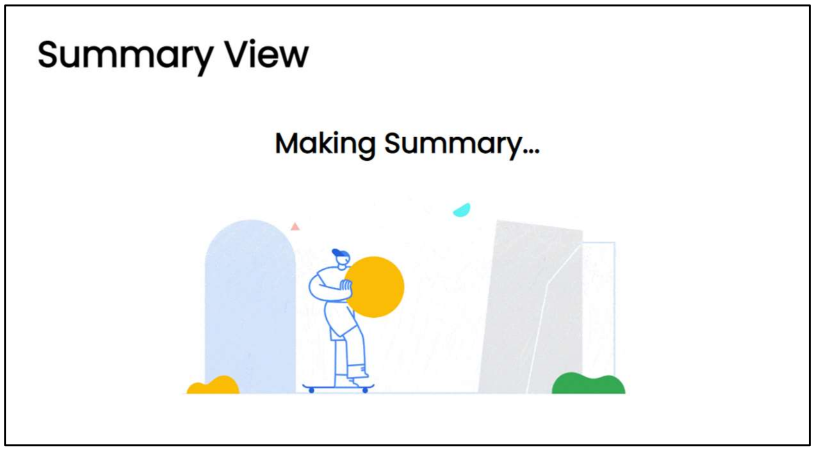


Figure 9 - Summary View Loading Page

When the response has been received from the network component, the JSON is sent to the Summary View component for rendering.

3.1.2 Summary View

The API response JSON contains the fields sent in the request as well as a new *summary* field within *chat_package* that contains the summarization information. Figure 10 shows the general structure of the summary response.

```
{
    "summary_options": {
        ...
    },
    "chat_package": {
        "config": {
            "parties": [
                ...
            ]
        },
        "messages": [
            ...
        ],

        "summary": {
            "paragraph" : "Apples are my favourite fruit, what are yours? Oh really, I think apples are superior.
Interesting",
            "message_ids" : [
                0,2,3
            ]
        }
    }
}
```

Figure 10 - Response JSON Format

The *summary* field contains the fields *paragraph* and *message_ids*. The *paragraph* field contains the full summary paragraph that was generated by the model. This would be the set of sentences extracted from the chat dialogue and stitched together to form a paragraph. The field *message_ids* contain the IDs of the messages whose sentences were used in the extractive summary.

3.1.2.1 Erroneous Responses

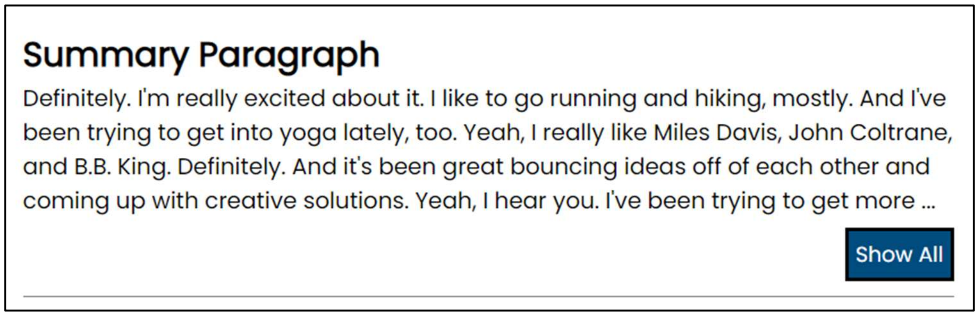
In addition to making requests and preparing the response, *apiJSONFetch* also handles erroneous responses. If the received API response is erroneous, the function first determines if the response was sent from the server by checking the HTTP status code in the response. Status codes of 400-599 represent client-side and server-side errors, so if the status code is within these ranges, then the response was sent from the server running the network component, which always responds using a specific JSON format. Otherwise, the response did not reach the server, so the error message is retrieved by looking at the text version of the HTTP response.

For either case, *apiJSONFetch* indicates an erroneous response by setting the *success* field of the returned object to false, and the *content* field to the error message. For successful responses, the *content* field would contain the JSON body of the NC API response.

The summary view checks the *success* flag in the response object returned by *apiJSONFetch*, and if the flag is false then the error component is rendered in the summary view with the corresponding error message. The rendered component for a sample error can be seen at Figure 26 in the Appendix.

3.1.2.2 Summary Paragraph

When a successful API response is received, the included chat package is sent to *SummaryParagraph.js* which is responsible for rendering the summary paragraph included in the response (Figure 11).



Summary Paragraph

Definitely. I'm really excited about it. I like to go running and hiking, mostly. And I've been trying to get into yoga lately, too. Yeah, I really like Miles Davis, John Coltrane, and B.B. King. Definitely. And it's been great bouncing ideas off of each other and coming up with creative solutions. Yeah, I hear you. I've been trying to get more ...

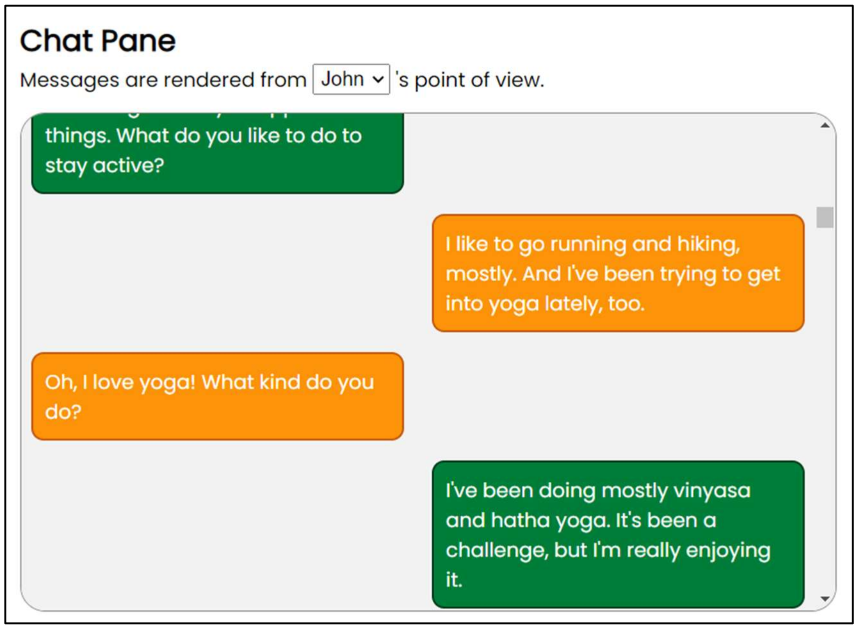
Figure 11 - SummaryParagraph.js Component

The text for the summary paragraph is obtained directly from the *paragraph* field in the response, but is limited to 350 characters by default to maintain the app layout. The user can toggle between the full and truncated paragraph using the *Show All* button (which becomes *Show Less* when the full paragraph is visible).

The summary paragraph allows the user to get the summary information they need immediately, for when they do not care to know which messages were used in the summary.

3.1.2.3 Chat Pane

The successful API response is also sent to the *ChatPane.js* component, which makes up the other part of the Summary View. The chat pane renders the chat dialogue into a text conversation seen on most messaging apps (Figure 12).


Chat Pane

Messages are rendered from John 's point of view.

things. What do you like to do to stay active?

I like to go running and hiking, mostly. And I've been trying to get into yoga lately, too.

Oh, I love yoga! What kind do you do?

I've been doing mostly vinyasa and hatha yoga. It's been a challenge, but I'm really enjoying it.

Figure 12 - ChatPane.js Component

The component works by getting the message objects included in the *message* field of the chat package, and rendering them into the message HTML elements shown above. Summary messages (messages whose sentences contributed to the summary paragraph) are determined by going through the *message_ids* sub-field in the *summary* field. If a message has its ID in the list, then its CSS is changed to render it as orange, indicating that it is part of the summary.

The component also selects one of the parties as the primary party, which determines the alignment of the messages in the chat pane. Messages from the primary party will be right aligned, while other messages will be left-aligned. This mirrors the behaviour of messaging apps, where the messages are rendered from the primary party's point of view. A user can also change the party POV of the chat pane using the dropdown.

3.2 Component Communication

As discussed in the design process, the network component and frontend component communicate using an HTTP API implemented in Flask and running as part of the network component.

The API only has one route *submit-chat/* which handles a summarization request received from the calling FEC. Figure 25 in the Appendix shows the source code for the API route.

When a request it received, the API performs validation to make sure it has all the required fields to service the request. This involves checking if the request was made with the appropriate HTTP method if request has a JSON body and if it contains the required *summary_options* and *chat_package* field. If validation fails, it sends an erroneous response with the status code 400, indicating that it is a client-side error.

If request validation is successful, the value of the fields are pulled from the request JSON and passed into network component object to perform the summarization. The content returned by the network component is the chat package updated with the *summary* field, as well as the status of the operation. If the status is non-zero, then an error occurred during the summarization process. In that case, it sends an erroneous response with status code 500, indicating that it is a server-side error.

Otherwise, the updated chat package and summary options are packaged into the successful response JSON which is sent back to the calling frontend component.

3.3 Summarization ML Model

Figure 13 below shows the high level diagram of the summarization process.

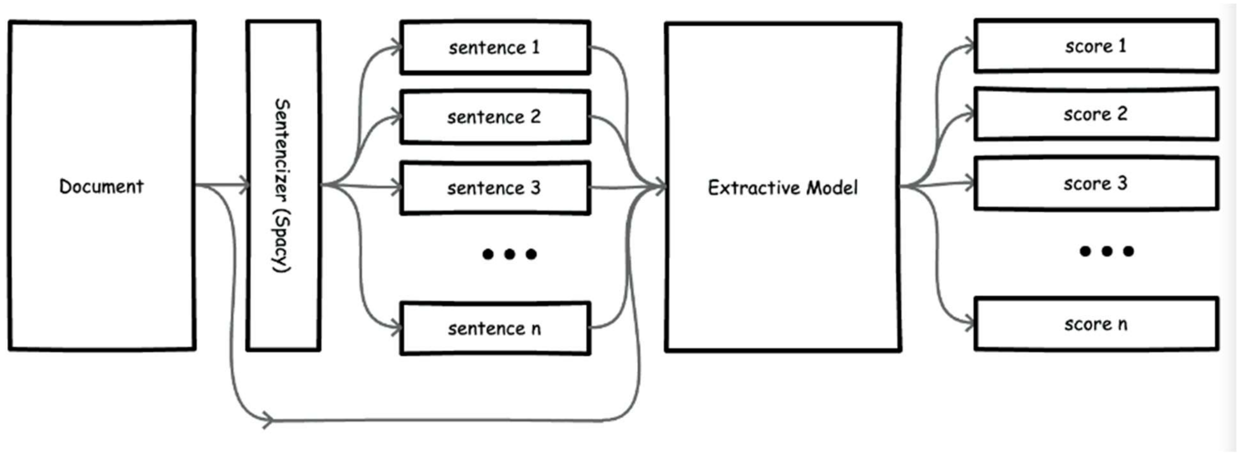


Figure 13 - Diagram of the Summarization Process

The final summarization ML model is a robustly optimized BERT model, trained on the SAMSum dataset specifically designed for dialogue summarization tasks.

RoBERTa, a variant of BERT, is a transformer-based neural network with attention mechanisms, multiple hidden, and many nodes. The regression head added on top of the RoBERTa model is a fully connected layer designed to output continuous values.

The loss function during training is the mean squared error (MSE) of the model's predicted output and the sentence's calculated ROUGE score between the sentence and the abstract gold summary.

For training the model, parameters were largely adjusted to fit the limitations of the computer training. To speed up training, a batch size of 4 and a gradient accumulation of 8 was chosen (an effective batch size of 32). Evaluation and saving of the model occurred at 1000 and 10,000 steps respectively. At the end of training with the dataset, the training reaches around 60,000 steps.

For summarization the model receives a Python dictionary from the NEC containing dialogue ‘turns’ which represent each speaker's contribution in the conversation in chronological order. Subsequently, the model employs Spacy's sentencizer [19], a sophisticated statistical sentence segmentation library, to further dissect the dialogue turns into distinct sentences. The model is then fed the sentences in batches, where each sentence is assigned a score: a higher score indicates the model predicts the sentence is more relevant to the final summary. The model ultimately selects the top *k* sentences, which exhibit the highest scores, to comprise the final summarized output. Throughout this process, the model’s wrapping code records the message and party id of chosen sentences to feed back to the network component along with the summary.

3.4 Network Component

T implementation of the network component (NC), which satisfies the requirements outlined in Section 2.4.3, has been completed to enable full integration of the extractive summarization model with the FEC. This integration facilitates the transfer of JSON packages between the backend machine learning model and the frontend, where user transcripts are inputted, and summary outputs are rendered.

The NC is an object initialized at API start-up. This means the same instance of the network component will be called each time to service summarization requests to the *submit-chat* API route. A single object was used due to the significant object initialization delay, which would increase service times if new objects were initialized for every service request. The API calls the NC object *summarize* method, passing *summary_options* and *chat_package* from the FEC request as parameters.

Each network component object initializes the summarization model, which is called in the *summarize* method and takes the series of messages from *chat_package.messages*, as well as *summary_options* to perform the extractive summarization. The model returns a summary paragraph, which is a concatenation of the *top k* selected sentences, as well as the list of message IDs which correspond to the IDs of messages whose sentences were used in the summary.

The NC then takes the returned paragraph and message IDs and adds them as the fields *summary.message_ids* and *summary.paragraph* respectively to the original chat package, which creates the summary chat package discussed in Section 3.1.2. The updated chat package is returned to the API to complete the service request.

3.5 Bill of Materials

The project accrued no costs as all resources, frameworks and tools used to build the application are free to use. Table 3 shows the bill of materials for the application.

Table 3 - Bill of Materials

	Tool	**Cost (CAD)**	**Justification**
1	Flask	0.00	Free, open-source software [10]
2	React	0.00	Free, open-source software [14]
3	HTML	0.00	Free, open-source software [20]
4	CSS	0.00	Free, open-source software [15]
5	JSON	0.00	Free, open-source software [13]
6	PyTorch	0.00	Free, open-source software [21]
7	HuggingFace	0.00	Free, open-source software [22]
8	Standard Python	0.00	Free, open-source software [23]
9	GPU-Enabled Computer	0.00	Running on team member Jaden's computer
10	Vercel Web Hosting	0.00	Hobby Account is free to use [24]

It is important to note that costs can be incurred if the application would need to be deployed on a larger scale, as that could incur costs for web hosting and servers.

4 Testing and Evaluation

4.1 Frontend Component Testing

The FEC web app was tested for the following categories:

- User Input Testing: Testing the web app on handling user input.
- Layout Testing: Testing the web app response to window scaling and resizing.
- Security Testing: Testing the web app on handling/preventing malicious attacks.
- Volume Testing: Testing the web app limits for handling transcript text.

4.1.1 User Input Testing

Table 4 below shows the erroneous user inputs tested, and the web app response to the user input.

Table 4 - User Input Testing Cases

Erroneous User Input	**Application Response**
Clicking *Summarize* immediately	Request validation blocks request since no input is selected and responds with alert (Figure 27).
Attempting to summarize without a transcript	Request validation blocks request since transcript text box is empty and responds with alert (Figure 28).
Attempting to summarize without a file uploaded	Request validation blocks request since no file has been uploaded and responds with alert (Figure 29).
Attempting to summarize with both a transcript and a file upload	Impossible user input as toggle forces user to select only one input source.
Uploading an invalid file (unallowed file extension)	File is blocked at upload time and app responds with alert (Figure 30).
Uploading an invalid file (no file extension)	
Uploading an invalid file (file cannot be parsed)	Request validation on request submission blocks the request and responds with alert (Figure 31).
Invalid transcript that cannot be parsed	
Invalid summarization options	Impossible user input since dropdown only provides a specific set of options and is disabled if the transcript could not be parsed.

An additional test was also performed to test a user toggling between the different source options, to ensure that the content the user put in (the text in the transcript text box, or the uploaded file) is not removed when the user toggles to a different option.

4.1.2 Layout Testing

The application was also tested to see how the application layout is affected by resizing of the browser window.

By default, the Control Board component and Summary View component are anchored to the left and right ends of the application window respectively with empty space that varies based on the size of the window (as shown in Figure 2).

As the browser window reduces, the empty space between the components is reduced to accommodate. If the browser window falls under a certain threshold, the Summary View falls under the Control Board component, as shown in Figure 32 in the Appendix. This allows the application to be used at smaller window sizes on laptops and PCs.

Unfortunately, the application layout is negatively impacted due to the limited screen width on mobile devices (see Figure 33 in the Appendix). The Summary View component does not fit within the window width and therefore must be scrolled horizontally by the user. The layout for the source toggle buttons has also changed because of the reduced width. While the application is still usable in this state, it does not provide the best user experience. Unfortunately, the required changes to implement a better user experience on user devices could not be completed within the project timeline.

4.1.3 Security Testing

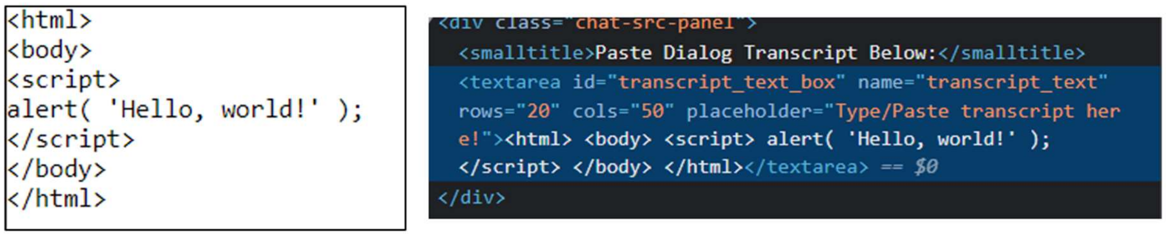
The transcript text box is the only source of direct user input on the site and can be used to inject malicious HTML and JavaScript as an attempt to hijack the application [25].

Injection attempts such as reflected HTML injection attacks [25], are not possible as the transcript textbox uses the HTML *textarea* element only, which is not embedded in a form element. Furthermore, the contents of the text box are never treated as executable HTML and/or JavaScript code within the web app, but instead as standard string which is parsed. As a result, no malicious code contained in the text is ever executed on the application and will be blocked due to parsing failures in *chatTextToChatJSON*.

Some HTML injection attacks were attempted manually to verify that the web app responds appropriately.

4.1.3.1 Standard HTML

This injection attempt is to hijack the web application to run the JavaScript and send an alert for "Hello World" to the browser. Figure 14 below shows the text box content on the web app, along with its location in the HTML source code.



```
<html>
<body>
<script>
alert( 'Hello, world!' );
</script>
</body>
</html>
```

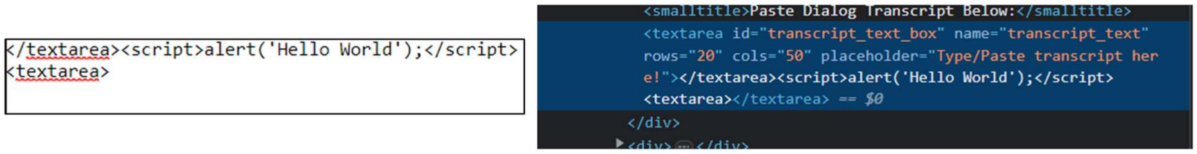
```
<div class="chat-src-panel">
  <smalltitle>Paste Dialog Transcript Below:</smalltitle>
  <textarea id="transcript_text_box" name="transcript_text"
  rows="20" cols="50" placeholder="Type/Paste transcript her
  e!"><html> <body> <script> alert( 'Hello, world!' );
  </script> </body> </html></textarea> == $0
</div>
```

Figure 14 - Standard HTML Injection in GUI (left) and source code (right)

As expected, there is no alert made when the HTML is pasted into the text box, even when the summarize button is clicked. The application responds with a parsing failure for the transcript (Figure 31).

4.1.3.2 Wrapped HTML

For the previous case, the HTML injection was embedded within the tags of the *textarea* element, which could automatically force it to be treated as non-executable text. This injection attempts to enclose the *textarea* tag for the textbox, so that the script can be seen as valid HTML element that can be executed. Figure 15 shows the source code.



```
</textarea><script>alert('Hello World');</script>
<textarea>
```

```
  <smalltitle>Paste Dialog Transcript Below:</smalltitle>
  <textarea id="transcript_text_box" name="transcript_text"
  rows="20" cols="50" placeholder="Type/Paste transcript her
  e!"></textarea><script>alert('Hello World');</script>
  <textarea></textarea> == $0
</div>
▶<div>…</div>
```

Figure 15 - Wrapped HTML Injection in GUI (left) and source code (right)

This injection attempt also fails, resulting in the same response as the previous injection attempt.

4.1.4 Volume Testing

The web application was also to tested to find the largest transcript that can be handled without perceivable latency (approximately 10ms [26]). This is both parsing latency (the amount of time required to populate the parities in the summary options dropdown) and typing latency in the transcript text box.

Trial and error were used to determine the transcript size required to reach 10ms of latency. A sample transcript was duplicated to generate a transcript of the desired size, and latency tests were performed 10 times with the generated transcript. The average value of the latency tests were assigned to the generated transcript, and if it was under 10ms a larger transcript was generated and tested.

Testing was also done at different levels of throttling to simulate the processing power available for different classes of devices. Table 5 contains the results of the tests.

Table 5 - Maximum Transcript Sizes for 10ms Parsing and Typing Latency

Performance Level	**Maximum Parsing Latency Transcript Size (lines)**	**Maximum Typing Latency Transcript Size (lines)**
Standard Laptop	40000	5000
Mid-tier Mobile Phone	2000	1500
Low-end Mobile Phone	1000	1000

These results are acceptable as it is reasonable that transcripts of 1000 lines or longer will most likely be pasted into the text box or uploaded as a file, rather than being typed by the user directly.

4.2 API Testing

API testing was similar to user input testing for the web application. A variety of invalid requests were sent to the API route to ensure that it rejects them appropriately. Table 6 shows the invalid requests tested and the API responses.

Table 6 - Request Testing for API

Invalid Request	**API Response**
Request to URL other than *api/submit-chat*	Responds with code 404, *Not Found* error to client
Request without JSON body	Responds with code 400, *Bad Request* error: "Did not attempt to load JSON data because the request Content-Type was not application/json".
Request without *summary_options* field	Responds with code 400, and message *Required keys are missing*.
Request without *chat_package* field	Responds with code 400, and message *Required keys are missing*.
Invalid content in *summary_option* field	Responds with code 500, and message *Summarization process failed on server*.

Invalid content in *chat_package* field	Responds with code 500, and message *Summarization process failed on server.*

4.3 Model Testing

In order to evaluate the performance of an extractive summarization machine learning model, two main metrics were used: loss on the training and evaluation datasets and ROUGE metrics including F1 measure, recall, and precision. ROUGE1 scoring was selected for this problem to measure and evaluate the overlap of individual words between the generated extractive summaries and the provided abstractive summaries in the dataset.

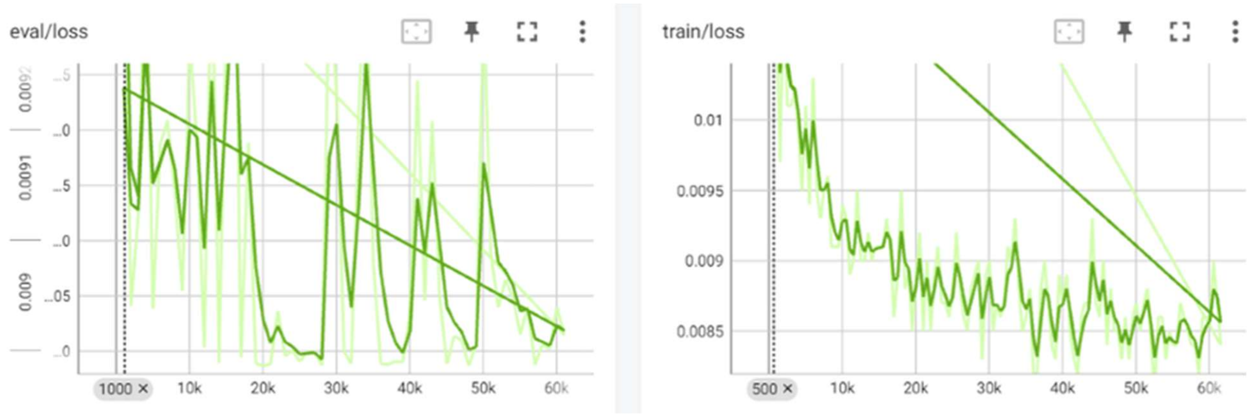


Figure 16 - Loss statistics through 60000 steps of training

The model loss metric was evaluated through 60 thousand steps of training. Model training functioned by having the model learn to predict the ROUGE score a sentence in a dialogue would have with respect to the abstractive reference summary. The loss function used the predicted and actual ROUGE1 scores on each dialogue sentence to find the mean squared error of the model after each step of the training. Figure 16 depicts how the evaluated loss decreased to below 1% on both the evaluation and training datasets. This is a promising outcome as it is indicative of the model's ability to generalize appropriately to new data without overfitting to the training dataset.

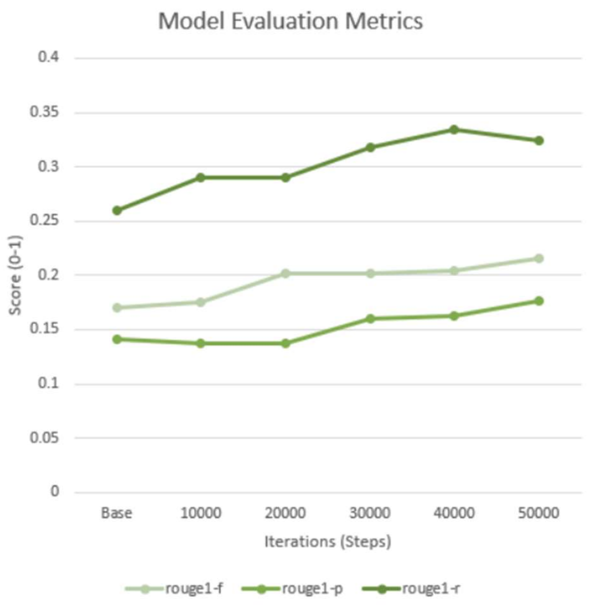


Figure 17 - ROUGE evaluation metrics for extractive summarization model

The performance of the model was also evaluated using ROUGE metrics. Through 50K steps of training, using the SAMSum testing dataset, testing of ROUGE1 scores versus the provided abstract summaries for given dialogues was performed. Referring to Figure 17, the model achieved a score of 0.32 in recall, 0.22 in f1, and 0.17 in precision. Here, "recall" refers to the proportion of overlapping words between the aggregation of chosen summary messages and the provided abstractive summary, relative to the length of the abstractive summary. The term "precision" refers to the same overlap, but relative to the length of the generated summary. Last, “f1” refers to the harmonic mean of precision and recall, and is the metric used to compare the model’s summarization quality to state of the art (SOTA) performance.

Current SOTA average 0.4 to 0.5 in f1 with ROUGE1 scoring [27]. The current model performance is significantly below SOTA with an f1 of 0.22, indicating a need to improve the model's training and summarization quality.

5 Stakeholder Analysis

The main concern was privacy in handling user information since the tool could potentially be used to for sensitive client specific information. To mitigate this, the tool was designed to never store any data between sessions, either locally or on the cloud. This would mean that as soon as the summarization task was complete, the user would have the option to copy their extractive summary to a more secure location and then delete the contents on the user interface.

Privacy is also ensured in component communication by using the HTTPS protocol [28] on the Vercel hosting service. This would help prevent unauthorized access to chat dialog transcripts in transmission.

From a cost perspective, we aimed to create a cost-effective solution by automating the summarization process, reducing the need for manual labor, and saving businesses money on hiring and training employees for manual summarization tasks.

6 Specifications Compliance

6.1 Functional requirements

The main requirements for performance outlined in the project's proposal phase were to enable the extractive summarization of transcriptions of dialogues and to provide users with a formal visualization of the summarization result.

The summarization model is able to successfully accept a transcript from the user, parse and compute the relevant sentences and return the series of summary messages to the front end where these sections of the dialogue will be rendered and visualized by the user.

6.2 Interface requirements

The frontend user interface is robustly designed, only accepting file and text inputs which follow a standardized format. Rendered summaries are easily digestible, as the extracted summaries are highlighted with various options for user convenience. Users have the options to enter in either *.txt* transcript files or directly input transcripts in a designated chat text field.

6.3 Performance requirements

The network component (NC), incorporating the extractive summarization model, meets the required speed specifications for summarization and end-to-end render times of less than 10 seconds. This was tested using dialogue transcripts of a variety of lengths.

However, the original performance objectives for summary quality were not satisfied by the current state of the model. This is due to physical hardware computational limitations, which have hindered the team's ability to fine-tune the RoBERTa model for extractive summarizations by markedly increasing the model training times. As a result, the average ROUGE1 F1 scores are lower than the targeted state of the art (SOTA) performance.

Table 7 below summarizes the requirements achieved by the final solution.

Table 7 - Software Specifications Table with Achieved Specifications

Specification		**Specification Met**
1	**Functional requirements**	
1.1	Extractive Summarization of chat dialog (transcript between 2 entities)	Yes
1.2	Visualization of results to highlight summary sentences within dialog transcript	Yes
2	**Interface requirements (A GUI prototype is included, see Figure 19 in the Appendix)**	
2.1	Users can upload dialog transcript file	Yes
2.2	Users can paste chat transcript text	Yes
2.3	Summarization Rendered as a chat conversation with key sentences highlighted.	Yes
3	**Performance requirements**	
3.1	End-to-end response time within summarization and processing tolerances (10 seconds)	Yes
3.2	Limit user resource usage expand applicability to wide range of devices	Yes

7 Conclusions and Recommendations

7.1 Conclusion and Future Work

Following the design approach, the tool was implemented as a web application with a React frontend and a Flask backend. The React frontend represents the network component and contains the GUI that handles user interaction and communication with the backend. The backend consists of the Flask API which invokes the implemented network component class to service summarization requests from the frontend component. The network component provides a method which calls the model to perform the summarization request and uses the model output to create the summary chat package sent back to the React web app (via the API) for rendering to the user.

The tool satisfies all defined requirements as it can summarize a given file or text input under 10 seconds, and its visualization indicates key messages that were used in the summary. However, there are some aspects of its implementation that can be improved.

Future implementations of the frontend component could implement the CSS to fix layout issues on mobile devices or design a new application layout optimized for mobile device platforms. The FEC could also be implemented as native applications for mobile devices, which would improve processing performance allowing for larger transcripts to be parsed without perceivable latency.

As for the API, a possible bottleneck is the use of only one network component object to service all API requests. This can result in queueing of requests due to resource contention for the single object. The resource contention can be resolved by running the API and NC on more capable hardware, which would reduce the initialization delay and allow objects to be created during service. A pool of NC objects could also be initialized at API startup and requests can be dispatched to those as required.

Several recommendations can also be considered to enhance the performance of the training pipeline and the model optimization process. One limitation encountered was the constraint on batch size and training speed, which resulted from the use of an NVIDIA RTX 3060 GPU. To address this issue, utilizing more advanced hardware or distributed training techniques could lead to improvements in training efficiency and model performance. Additionally, the model's effectiveness in dialogue summarization could potentially be improved by refining the loss function. This can be achieved by exploring alternative metrics and incorporating them into the model, thereby increasing the complexity of the loss function and enabling more nuanced optimization. Furthermore, the current method of selecting the top *k* sentences for summarization could be replaced by more sophisticated heuristics, such as Maximal Marginal Relevance (MMR) [29] or other dialogue-specific approaches. Implementing these advanced heuristics may result in more accurate and relevant summarizations, ultimately enhancing the overall quality and usefulness of the model's output.

Other general recommendations include designing the tool to comply with data protection codes and standards, such as GDPR and CCPA [30]. Features such as data anonymization and consent management can be implemented to protect user privacy. Another consideration could be building a bank of ethical guidelines for the use of the extractive summarization tool, which could include transparency in the use of machine learning models, obtaining consent from stakeholders when processing sensitive information, and ensuring that the tool does not introduce biases or unfairness in the summaries.

7.2 Project Market Analysis and Potential Impact

7.2.1 Demand

The demand for extractive summarization tools has been ever increasing in recent years as more and more organizations seek to automate the process of summarizing large volumes of text data [4]. This is primarily driven by the need for faster ways to extract key information from large documents, news articles and more information sources. For example, the AI in the computer market is expected to reach $99.9 billion by 2025, growing at a CAGR (Compound Annual Growth Rate) of 42.2% from 2020 to 2025 [31]. Since extractive summarization is a subsegment of AI, we can use this information to estimate the market size.

To estimate the market size, we can use the overall AI market size and the proportion of the extractive summarizer market within it. Assuming that the extractive summarization market accounts for 5% of the AI market (this is a speculative assumption, as there is no readily available data), we can calculate the market size as follows:

$$Market\ size\ (2025)\ =\ AI\ market\ size\ (2025)\ *\ proportion\ of\ extractive\ summarizer\ market$$

$$Market\ size\ (2025)\ =\ \$99.9\ billion\ *\ 5\%$$

$$Market\ size\ (2025)\ =\ \$4.995\ billion$$

Based on this speculative analysis, the extractive summarizer market size could be around $4.995 billion by 2025. However, it's essential to note that this is a rough estimate based on available data and assumptions. Further research and data analysis would be required to arrive at a more accurate estimate.

7.2.2 Target Audience

The target audience for this extractive summarization tool includes businesses in the financial services space, government agencies, educational institutions, and individuals [32]. This tool could be particularly useful for professionals in fields such as journalism, marketing, law and research.

7.2.3 Competition

The market for extractive summarization tools is highly competitive with a range of established players and new entrants vying for market share. Some of these key players include Google, Microsoft, IBM, and OpenAI [6]. In addition to these larger companies, there are also several smaller startups and niche players offering specialized summarization tools.

7.2.4 Positive and Negative Impacts

Some of the positive effects of an extractive summarizer are:

1. Time saving: These can drastically reduce the time required to read and understand lengthy documents, allowing stakeholders to focus on decision making and other more important tasks from an entity perspective.
2. Cost effective: By automating the summarization process, businesses can save money on hiring and training employees who would otherwise be performing manual summarization tasks.
3. Consistency: An extractive summarizer tool could also provide consistent summaries without being influenced by emotions or personal biases to ensure that the stakeholders receive objective information that they can base their decisions on
4. Scalability: These tools can handle large volumes of data, allowing businesses to process and analyze information as efficiently as they would want it.

Some of the negative effects of an extractive summarizer could be:

1. Loss of context: Extractive summarizers may not always capture the full context or meaning of the original text, leading to potential misunderstandings or misinterpretations.
2. Job displacement: While it was noted that this is a cost-effective method for businesses, it might also negatively impact employees who perform manual summarization tasks such as writers, editors, and researchers.
3. Ethical concerns: Some stakeholders might have ethical concerns about the use of machine learning models to process and summarize sensitive information, such as confidential business information, or personal data.

4. Perception of machine leaning models: The use of extractive summarizers could contribute to the perception that machine learning models are replacing human intelligence and creativity, potentially leading to resistance or any skepticism among stakeholders.

7.2.5 Impact Mitigation

To mitigate some of the negative effects of the effective summarizer, a variety of solutions can be looked at:

1. Loss of context: To mitigate the loss of context, it is essential to combine extractive summarization with human review and validation. This ensures that the summary captures the intended meaning and context of the original text.
2. Job displacement: Businesses can focus on upskilling and reskilling employees to adapt to new roles that complement the use of extractive summarizers. This could include training employees in data analysis, critical thinking, or machine learning model development and maintenance.
3. Ethical concerns: Develop and implement ethical guidelines and best practices for the use of extractive summarizers, particularly when dealing with sensitive information. This may include anonymizing data, obtaining consent from stakeholders, and maintaining transparency regarding the use of machine learning models.
4. Perception of machine learning models: Educate stakeholders about the benefits and limitations of machine learning models, emphasizing that these tools are designed to augment human intelligence and creativity, not replace them. Encourage collaboration between human experts and machine learning models to achieve the best results.

7.2.6 Potential Revenue Streams

Extractive summarization tools can generate revenue through a variety of channels, including subscriptions, licensing fees, and pay-per-use models. Some companies also offer custom development services, where they work with clients to create bespoke summarization tools tailored to their specific needs. Another potential revenue stream is the sale of data insights generated through the use of these tools, which can be valuable to businesses and researchers.

Overall, the market for extractive summarization tools is expected to continue growing as more organizations seek to streamline and extract the use case from their text analysis workflows. This would enable them to extract key insights from large volumes of data. However, competition in this space is fierce and companies will need to differentiate themselves through advanced features, ease of use and strong customer support to succeed.

8 Team Expended Effort

Name	**Overall Effort Expended (%)**
Ifeanyi Maduabuchi	100
Shrish Veerasundar	100
Jaden Edhlund-Roy	100
Hai-Ling Rao	100

9 References

[1] "AI Chat Bots May be the Financial Savers Hospitals are Seeking," *MC AnalyTXs*, Feb. 28, 2019. https://mcatx.com/ai-chat-bots-may-be-the-financial-savers-hospitals-are-seeking/ (accessed Apr. 09, 2023).
[2] K. Gilchrist, "Chatbots expected to cut business costs by $8 billion by 2022," *CNBC*. https://www.cnbc.com/2017/05/09/chatbots-expected-to-cut-business-costs-by-8-billion-by-2022.html (accessed Oct. 25, 2022).
[3] "Chatbots." https://www.juniperresearch.com/researchstore/operators-providers/chatbots-trends-research-report (accessed Apr. 09, 2023).
[4] Sciforce, "Towards Automatic Text Summarization: Extractive Methods," *Sciforce*, Feb. 02, 2022. https://medium.com/sciforce/towards-automatic-text-summarization-extractive-methods-e8439cd54715 (accessed Mar. 07, 2023).
[5] Y. Liu and M. Lapata, "Text Summarization with Pretrained Encoders." arXiv, Sep. 05, 2019. doi: 10.48550/arXiv.1908.08345.
[6] "Introducing ChatGPT." https://openai.com/blog/chatgpt (accessed Apr. 09, 2023).
[7] S. Padó, U. Padó, and K. Erk, "Flexible, Corpus-Based Modelling of Human Plausibility Judgements," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, Czech Republic: Association for Computational Linguistics, Jun. 2007, pp. 400–409. Accessed: Apr. 09, 2023. [Online]. Available: https://aclanthology.org/D07-1042
[8] R. Nallapati, B. Zhou, C. N. dos santos, C. Gulcehre, and B. Xiang, "Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond." arXiv, Aug. 26, 2016. doi: 10.48550/arXiv.1602.06023.
[9] "What is a REST API? | IBM." https://www.ibm.com/topics/rest-apis (accessed Apr. 10, 2023).
[10] "API — Flask Documentation (2.2.x)." https://flask.palletsprojects.com/en/2.2.x/api/ (accessed Apr. 10, 2023).
[11] "Hypertext Transfer Protocol -- HTTP/1.1." https://www.w3.org/Protocols/rfc2616/rfc2616.html (accessed Apr. 10, 2023).
[12] "Django," *Django Project*. https://www.djangoproject.com/ (accessed Apr. 10, 2023).
[13] "JSON." https://www.json.org/json-en.html (accessed Apr. 10, 2023).
[14] "React – A JavaScript library for building user interfaces." https://reactjs.org/ (accessed Oct. 24, 2022).
[15] "Cascading Style Sheets." https://www.w3.org/Style/CSS/Overview.en.html (accessed Apr. 10, 2023).
[16] "samsum · Datasets at Hugging Face," Dec. 23, 2022. https://huggingface.co/datasets/samsum (accessed Apr. 09, 2023).
[17] Y. Liu *et al.*, "RoBERTa: A Robustly Optimized BERT Pretraining Approach." arXiv, Jul. 26, 2019. doi: 10.48550/arXiv.1907.11692.
[18] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. Accessed: Apr. 10, 2023. [Online]. Available: https://aclanthology.org/W04-1013
[19] "Sentencizer · spaCy API Documentation," *Sentencizer*. https://spacy.io/api/sentencizer/ (accessed Apr. 10, 2023).
[20] "Learn HTML," *Codecademy*. https://www.codecademy.com/learn/learn-html (accessed Nov. 20, 2022).
[21] "PyTorch." https://www.pytorch.org (accessed Oct. 24, 2022).
[22] "Hugging Face – The AI community building the future." https://huggingface.co/ (accessed Nov. 22, 2022).
[23] "Python," *Python.org*. https://www.python.org/ (accessed Oct. 24, 2022).

[24] "Pricing – Vercel." https://vercel.com/pricing (accessed Apr. 10, 2023).
[25] Vijay, "HTML Injection Tutorial: Types & Prevention with Examples," *Software Testing Help*, Mar. 12, 2023. https://www.softwaretestinghelp.com/html-injection-tutorial/ (accessed Apr. 10, 2023).
[26] W. L. in R.-B. U. Experience, "Response Time Limits: Article by Jakob Nielsen," *Nielsen Norman Group*. https://www.nngroup.com/articles/response-times-3-important-limits/ (accessed Oct. 25, 2022).
[27] "Papers with Code - CNN / Daily Mail Benchmark (Extractive Text Summarization)." https://paperswithcode.com/sota/extractive-document-summarization-on-cnn (accessed Apr. 10, 2023).
[28] "What is HTTPS? | Cloudflare." https://www.cloudflare.com/learning/ssl/what-is-https/ (accessed Apr. 08, 2023).
[29] Y. Mao, Y. Qu, Y. Xie, X. Ren, and J. Han, "Multi-document Summarization with Maximal Marginal Relevance-guided Reinforcement Learning." arXiv, Sep. 30, 2020. doi: 10.48550/arXiv.2010.00117.
[30] "General Data Protection Regulation (GDPR) – Official Legal Text," *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/ (accessed Apr. 08, 2023).
[31] "Artificial Intelligence Market Trends, Drivers & Opportunities | MarketsandMarkets^{TM}," *MarketsandMarkets*. https://www.marketsandmarkets.com/Market-Reports/artificial-intelligence-market-74851580.html (accessed Apr. 09, 2023).
[32] D. Dhagarra, M. Goswami, and G. Kumar, "Impact of Trust and Privacy Concerns on Technology Acceptance in Healthcare: An Indian Perspective," *International Journal of Medical Informatics*, vol. 141, p. 104164, Sep. 2020, doi: 10.1016/j.ijmedinf.2020.104164.

10 Appendix

10.1 Work Breakdown Structure

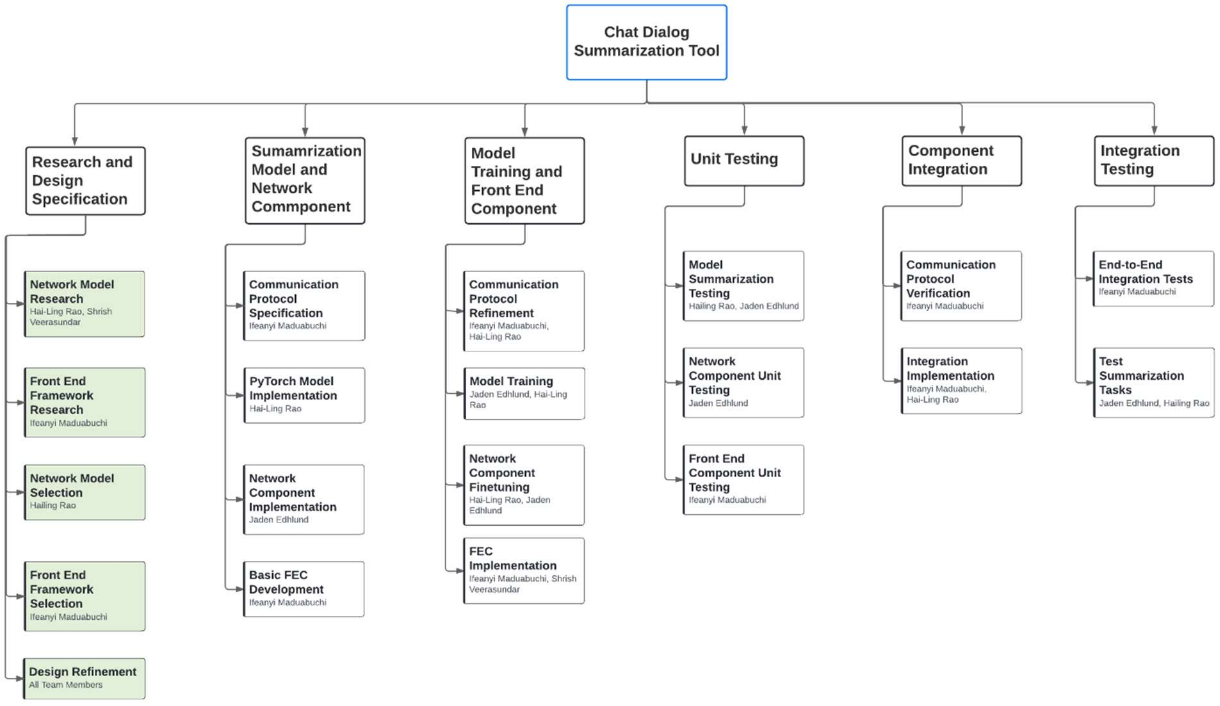

Chat Dialog Summarization Tool
Research and Design Specification
Network Model Research
Hai-Ling Rao, Shrish Veerasundar
Front End Framework Research
Ifeanyi Maduabuchi
Network Model Selection
Hailing Rao
Front End Framework Selection
Ifeanyi Maduabuchi
Design Refinement
All Team Members
Sumamrization Model and Network Commponent
Communication Protocol Specification
Ifeanyi Maduabuchi
PyTorch Model Implementation
Hai-Ling Rao
Network Component Implementation
Jaden Edhlund
Basic FEC Development
Ifeanyi Maduabuchi
Model Training and Front End Component
Communication Protocol Refinement
Ifeanyi Maduabuchi, Hai-Ling Rao
Model Training
Jaden Edhlund, Hai-Ling Rao
Network Component Finetuning
Hai-Ling Rao, Jaden Edhlund
FEC Implementation
Ifeanyi Maduabuchi, Shrish Veerasundar
Unit Testing
Model Summarization Testing
Hailing Rao, Jaden Edhlund
Network Component Unit Testing
Jaden Edhlund
Front End Component Unit Testing
Ifeanyi Maduabuchi
Component Integration
Communication Protocol Verification
Ifeanyi Maduabuchi
Integration Implementation
Ifeanyi Maduabuchi, Hai-Ling Rao
Integration Testing
End-to-End Integration Tests
Ifeanyi Maduabuchi
Test Summarization Tasks
Jaden Edhlund, Hailing Rao

Figure 18 - Work Breakdown Structure

10.2 Frontend Component Prototypes

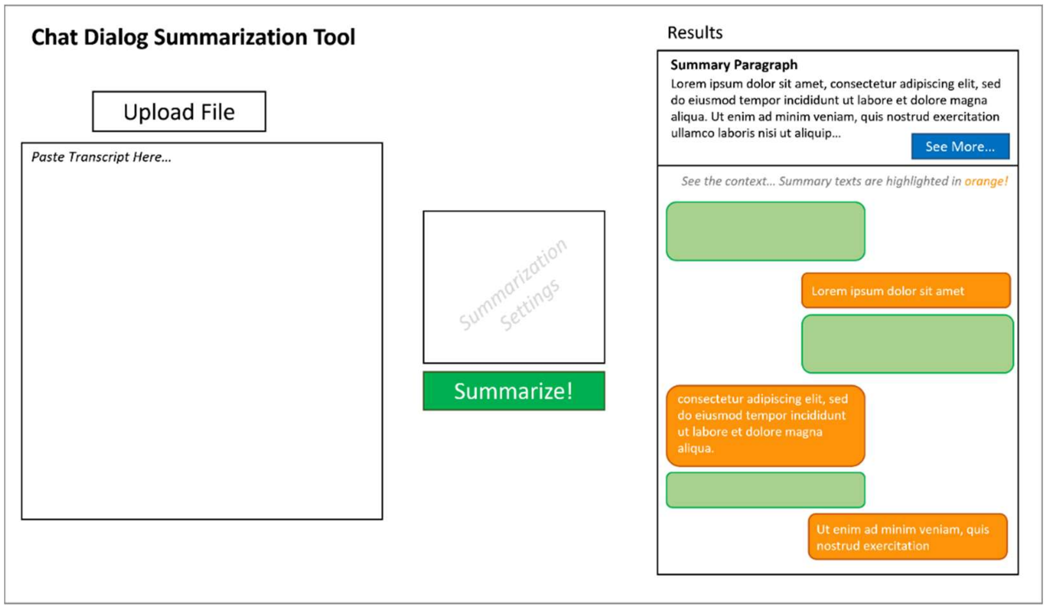
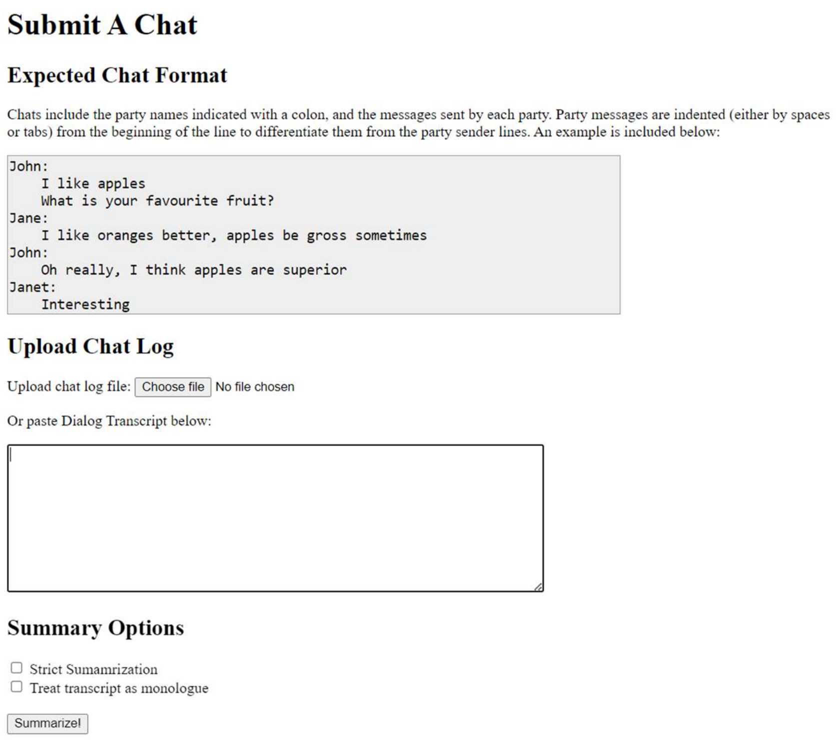

Chat Dialog Summarization Tool
Upload File
Paste Transcript Here...
Summarization Settings
Summarize!
Results
Summary Paragraph
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip...
See More...
See the context... Summary texts are highlighted in orange!
Lorem ipsum dolor sit amet
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation

Figure 19 - Initial GUI Prototype


Submit A Chat

Expected Chat Format

Chats include the party names indicated with a colon, and the messages sent by each party. Party messages are indented (either by spaces or tabs) from the beginning of the line to differentiate them from the party sender lines. An example is included below:

```
John:
    I like apples
    What is your favourite fruit?
Jane:
    I like oranges better, apples be gross sometimes
John:
    Oh really, I think apples are superior
Janet:
    Interesting
```

Upload Chat Log

Upload chat log file: Choose file No file chosen

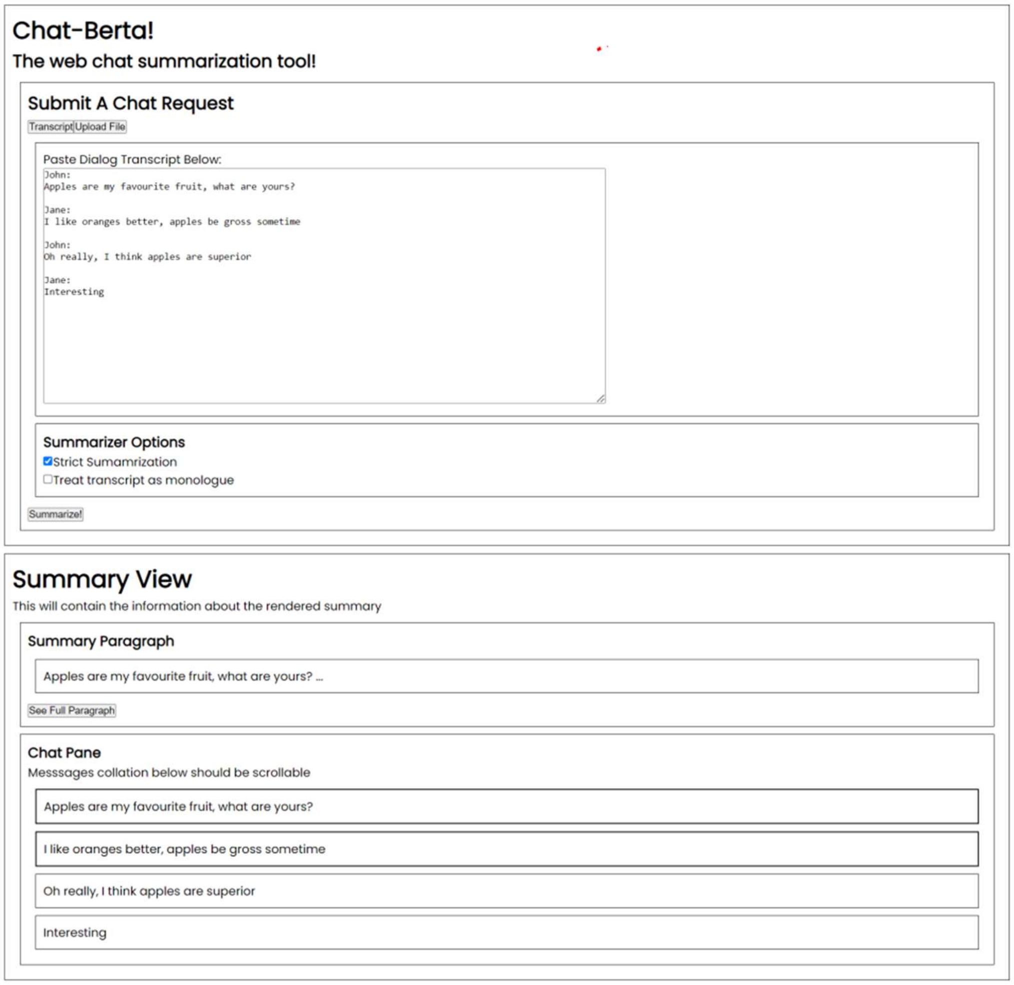
Or paste Dialog Transcript below:

Summary Options

☐ Strict Sumamrization
☐ Treat transcript as monologue

Summarize!

Figure 20 - Basic web site for API testing



Chat-Berta!

The web chat summarization tool!

Submit A Chat Request

Paste Dialog Transcript Below:

```
John:
Apples are my favourite fruit, what are yours?

Jane:
I like oranges better, apples be gross sometime

John:
Oh really, I think apples are superior

Jane:
Interesting
```

Summarizer Options

☑ Strict Sumamrization

☐ Treat transcript as monologue

Summary View

This will contain the information about the rendered summary

Summary Paragraph

Apples are my favourite fruit, what are yours? ...

Chat Pane

Messsages collation below should be scrollable

Apples are my favourite fruit, what are yours?

I like oranges better, apples be gross sometime

Oh really, I think apples are superior

Interesting

Figure 21 - FEC React Barebones GUI

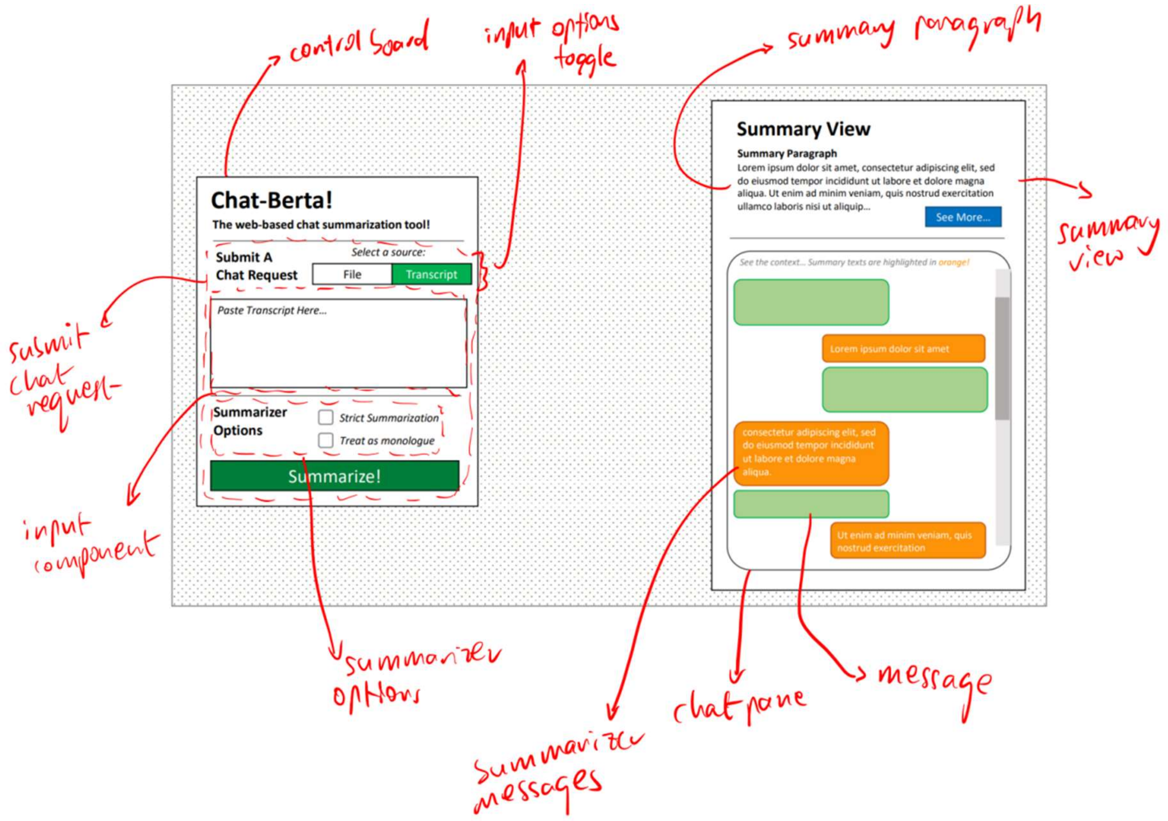

control board
input options toggle
summary paragraph
Chat-Berta!
The web-based chat summarization tool!
Submit A Chat Request
Select a source:
File
Transcript
Paste Transcript Here...
Summarizer Options
Strict Summarization
Treat as monologue
Summarize!
Summary View
Summary Paragraph
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip...
See More...
See the context... Summary texts are highlighted in orange!
Lorem ipsum dolor sit amet
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation
summary view
submit chat request
input component
summarizer options
summarizer messages
chat pane
message

Figure 22 - Refined FEC GUI Mock-Up indicating underlying React Components

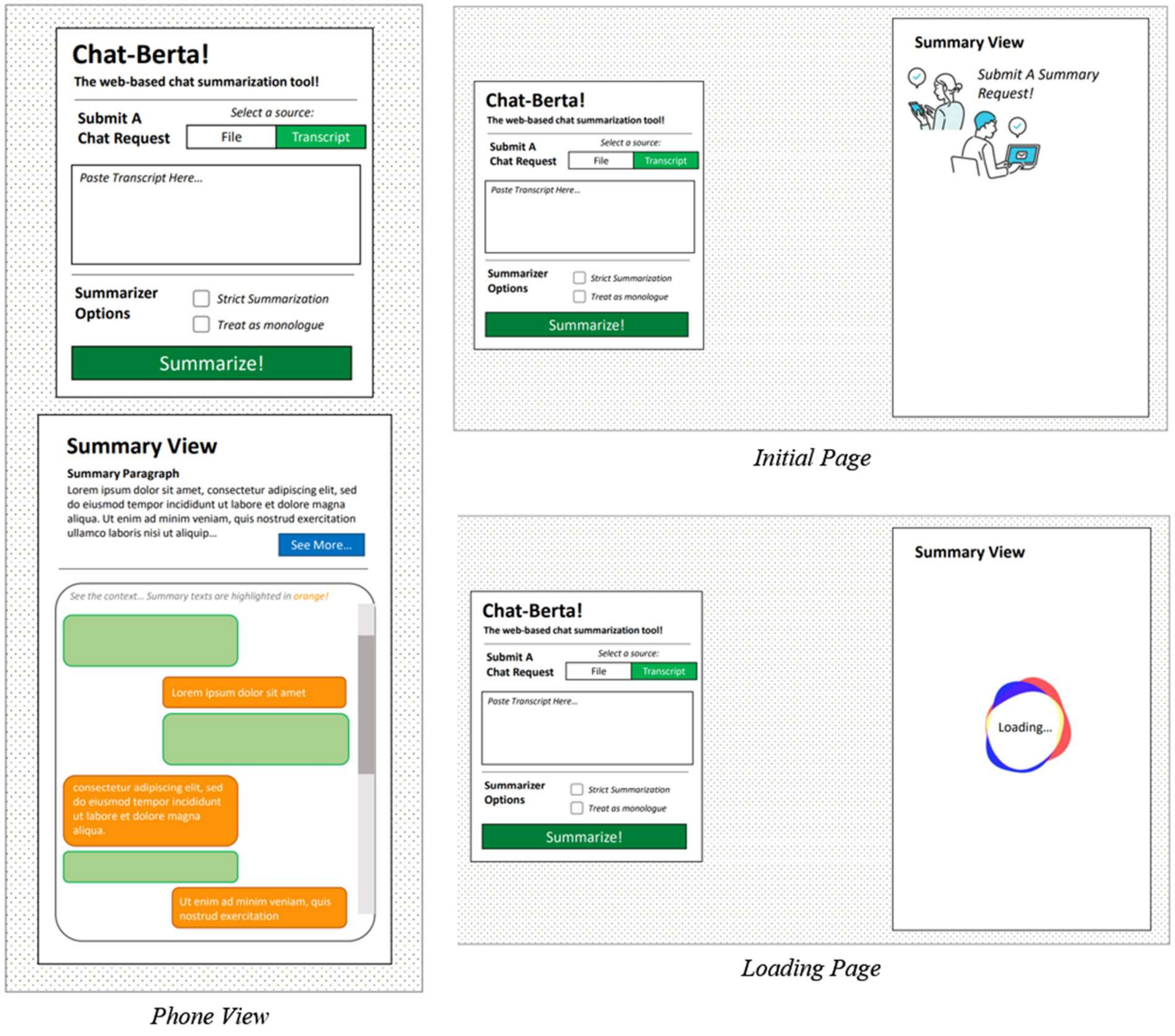
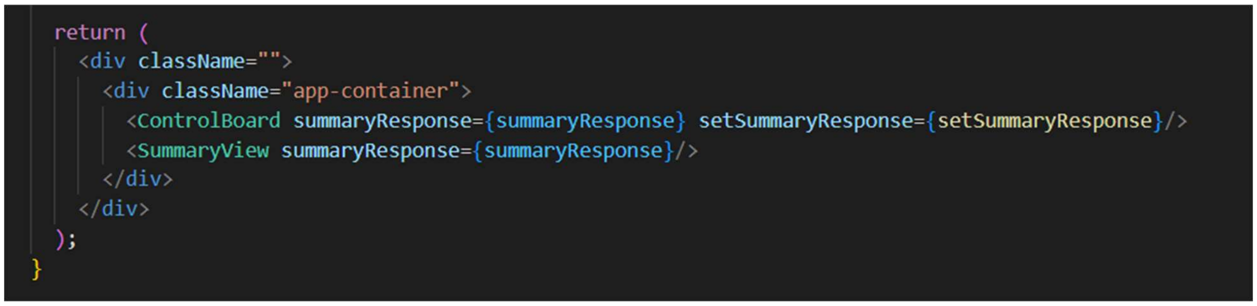


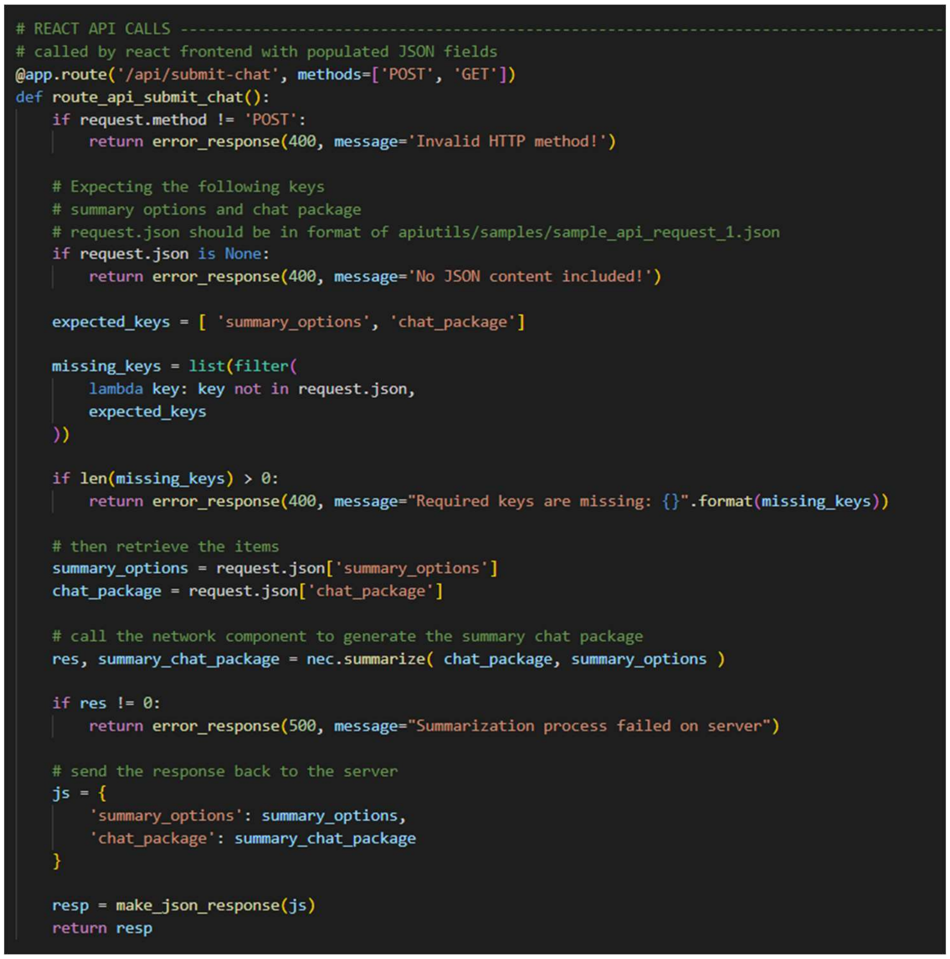
Figure 23 - Additional Mock-Ups for FEC GUI

10.3 Application Code



```
  return (
    <div className="">
      <div className="app-container">
        <ControlBoard summaryResponse={summaryResponse} setSummaryResponse={setSummaryResponse}/>
        <SummaryView summaryResponse={summaryResponse}/>
      </div>
    </div>
  );
}
```

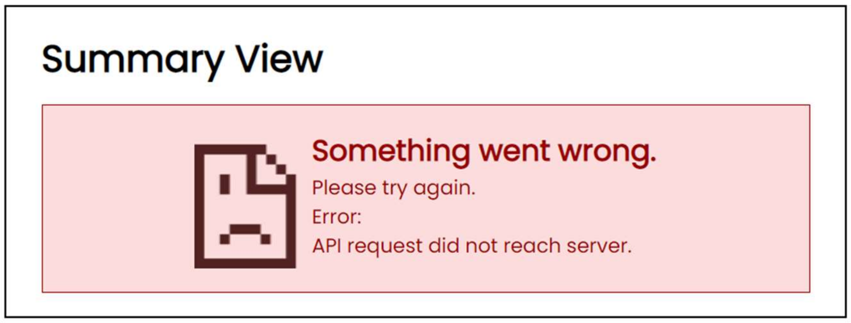
Figure 24 - Highest-Level Component Organization for Web Application



```python
# REACT API CALLS ------------------------------------------------------------------------------------
# called by react frontend with populated JSON fields
@app.route('/api/submit-chat', methods=['POST', 'GET'])
def route_api_submit_chat():
    if request.method != 'POST':
        return error_response(400, message='Invalid HTTP method!')

    # Expecting the following keys
    # summary options and chat package
    # request.json should be in format of apiutils/samples/sample_api_request_1.json
    if request.json is None:
        return error_response(400, message='No JSON content included!')

    expected_keys = [ 'summary_options', 'chat_package']

    missing_keys = list(filter(
        lambda key: key not in request.json,
        expected_keys
    ))

    if len(missing_keys) > 0:
        return error_response(400, message="Required keys are missing: {}".format(missing_keys))

    # then retrieve the items
    summary_options = request.json['summary_options']
    chat_package = request.json['chat_package']

    # call the network component to generate the summary chat package
    res, summary_chat_package = nec.summarize( chat_package, summary_options )

    if res != 0:
        return error_response(500, message="Summarization process failed on server")

    # send the response back to the server
    js = {
        'summary_options': summary_options,
        'chat_package': summary_chat_package
    }

    resp = make_json_response(js)
    return resp
```

Figure 25 - Code for submit-chat API Route

10.4 Application Components

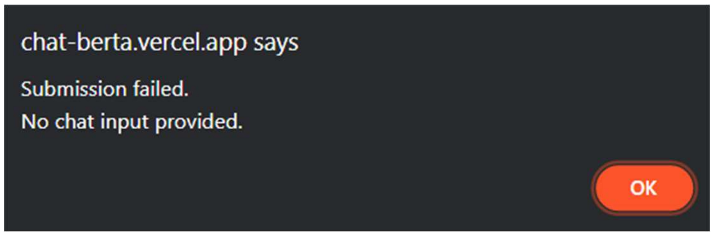


Summary View

Something went wrong.
Please try again.
Error:
API request did not reach server.

Figure 26 - Summary View Erroneous Response State

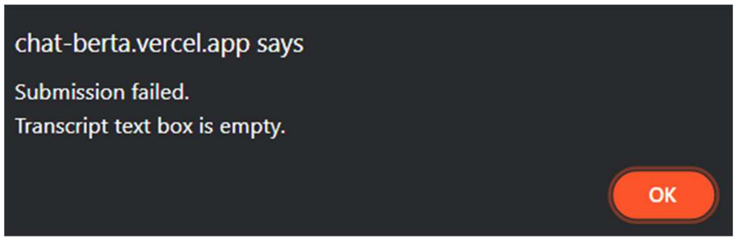
10.5 Web App Request Validation Failure Messages



chat-berta.vercel.app says

Submission failed.
No chat input provided.

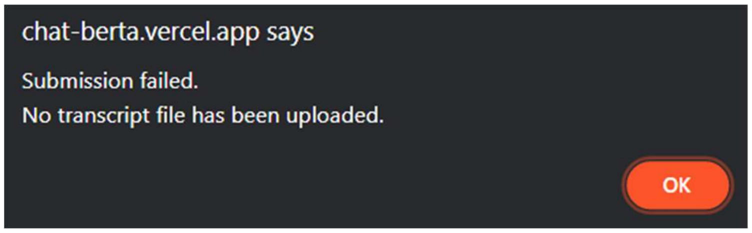
Figure 27 - Failure message when no input option is selected.



chat-berta.vercel.app says

Submission failed.
Transcript text box is empty.

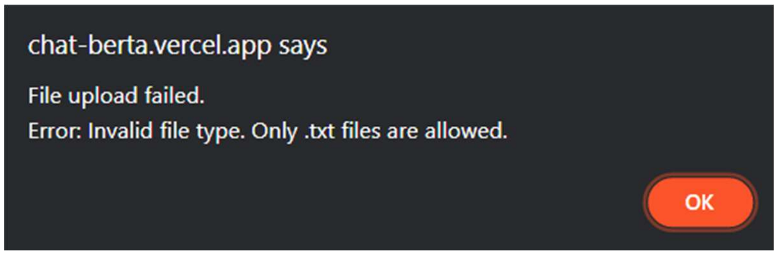
Figure 28 - Failure message when transcript is selected as source and text box has no content.



chat-berta.vercel.app says

Submission failed.
No transcript file has been uploaded.

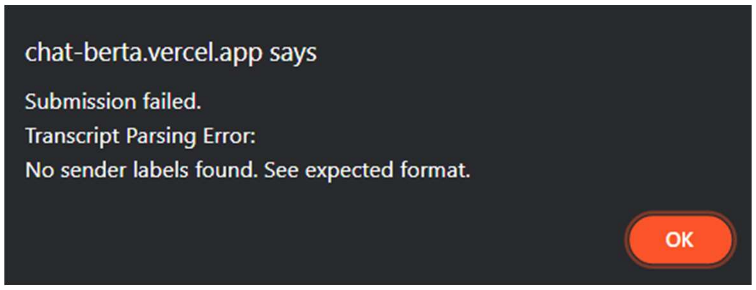
Figure 29 - Failure message when file is selected as source but no file has been uploaded.



chat-berta.vercel.app says

File upload failed.
Error: Invalid file type. Only .txt files are allowed.

Figure 30 - Failure message when invalid file type is uploaded.

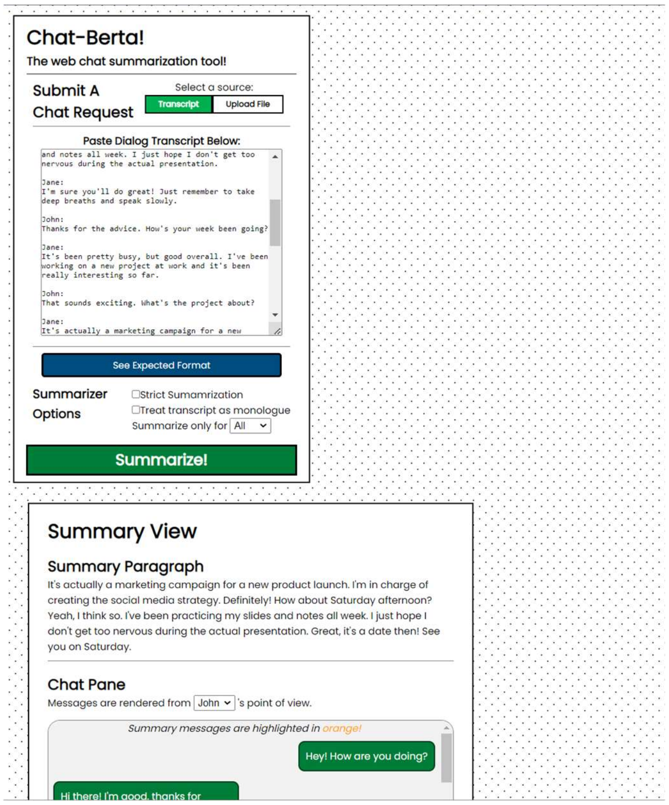


chat-berta.vercel.app says

Submission failed.
Transcript Parsing Error:
No sender labels found. See expected format.

Figure 31 - Failure when transcript parsing fails.

10.6 Application Layouts at Different Window Sizes



```
and notes all week. I just hope I don't get too
nervous during the actual presentation.

Jane:
I'm sure you'll do great! Just remember to take
deep breaths and speak slowly.

John:
Thanks for the advice. How's your week been going?

Jane:
It's been pretty busy, but good overall. I've been
working on a new project at work and it's been
really interesting so far.

John:
That sounds exciting. What's the project about?

Jane:
It's actually a marketing campaign for a new
```

It's actually a marketing campaign for a new product launch. I'm in charge of creating the social media strategy. Definitely! How about Saturday afternoon? Yeah, I think so. I've been practicing my slides and notes all week. I just hope I don't get too nervous during the actual presentation. Great, it's a date then! See you on Saturday.

Hey! How are you doing?

Hi there! I'm good, thanks for

Figure 32 - Layout on Reduced Width on Laptop/PC

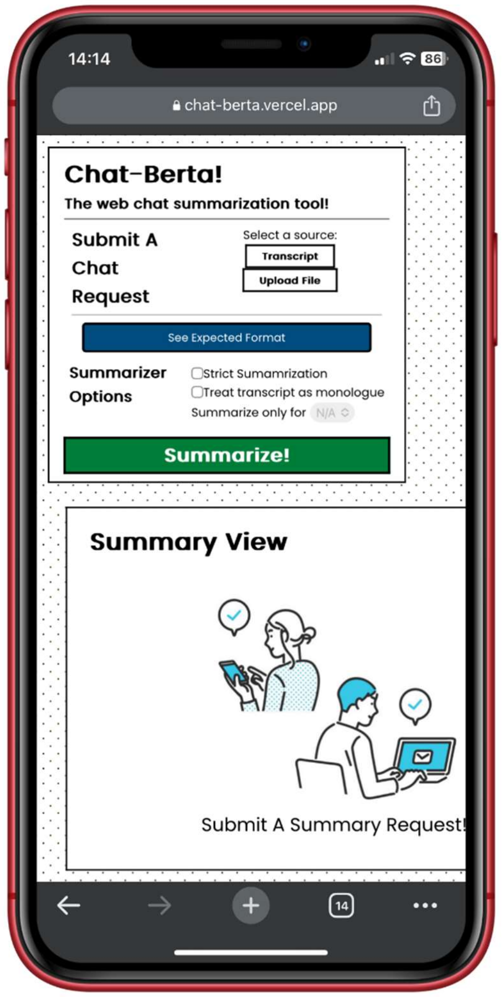


Figure 33 - Application Layout on Mobile Phone