JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Automated Article Summarization using Artificial Intelligence Using React JS and Generative AI

[1]Kunal Roy, [2]Subhash Mukherjee, [3]Sujata Dawn
[1]Assistant Professor, [2]Assistant Professor, [3]Assistant Professor
[1]Department of Computer Science and Engineering,
[1]Durgapur Institute of Advanced Technology and Management, Durgapur, India

Abstract : Automated Article Summarization using Artificial Intelligence (AI) has gained significant attention in recent years due to the increasing volume of online content. This research paper presents a novel approach that combines React JS, a popular JavaScript library for building user interfaces, with Generative AI techniques to automate the process of article summarization. The integration of React JS provides an intuitive and interactive platform for users to input articles and receive concise summaries in real-time. Generative AI models, such as neural networks and deep learning architectures, are employed to extract salient information and generate coherent summaries that capture the essence of the original articles. The paper discusses the technical implementation of the system, including data preprocessing, model training, and real-time summarization generation. Additionally, the evaluation metrics used to assess the quality and accuracy of the generated summaries are presented. The proposed system offers several advantages, including improved user experience, faster summarization process, and enhanced accessibility to information. The integration of React JS with Generative AI enables seamless interaction between users and the summarization system, allowing for efficient extraction of key content from articles across various domains. However, the paper also acknowledges the challenges associated with this approach, such as the potential limitations of Generative AI models in capturing nuanced contextual information and the need for continuous model training to adapt to evolving text patterns. Future research directions are discussed, including exploring hybrid approaches that combine extractive and generative techniques and investigating the integration of multi-modal inputs for more comprehensive summarization. In conclusion, this research paper presents an innovative system that combines React JS and Generative AI for automated article summarization, demonstrating its potential to revolutionize information processing and accessibility in the digital era.
***Index Terms* - React JS, Generative AI, Neural Networks, Deep Learning.**

I. Introduction

The users now have access to an excessive amount of information due to the rapid rise of digital material. Automated article summarizing using Artificial Intelligence (AI) has emerged as a viable method to extract essential insights and give succinct summaries of articles in this era of information overload. In order to automate the process of article summarization, this research paper offers a novel method that blends React JS, a well-known JavaScript toolkit for creating user interfaces, with Generative AI approaches. It has typically taken a lot of time and effort to manually summarize publications because it involves having human readers carefully consider and summarize the key ideas of large pieces. However, automated summarizing systems have gained popularity as a result of developments in AI and natural language processing, allowing users to easily find the information they need from massive amounts of text.

React JS's incorporation into the process of summarizing articles has a number of advantages. Users can submit articles and get real-time summaries on a simple and engaging platform made possible by React JS. The summarizing process is rapid and fluid thanks to its component-based architecture, which also ensures a smooth user experience. In parallel, Generative AI techniques play a crucial role in distilling essential information from articles and generating coherent summaries. These techniques leverage neural networks and deep learning architectures to learn from large datasets and capture semantic and contextual information. By utilizing Generative AI models, the system can extract salient information, condense it, and maintain the original context while generating summaries.

This research paper aims to explore the technical implementation of the proposed system, including data preprocessing, model training, and real-time summarization generation. It also discusses the evaluation metrics used to assess the quality and accuracy of the generated summaries, ensuring that the system produces reliable and meaningful results. While the integration of React JS and Generative AI offers numerous advantages, it is essential to acknowledge the challenges associated with this approach. Generative AI models may struggle with capturing nuanced contextual information and require continuous training to adapt to evolving text patterns. Additionally, maintaining the balance between brevity and preserving crucial details poses a significant challenge in generating coherent summaries.

Future research directions in the field of automated article summarization using React JS and Generative AI are also highlighted. These include investigating hybrid approaches that combine extractive and generative techniques to leverage their respective strengths and exploring the integration of multi-modal inputs, such as images or audio, to enhance the comprehensiveness of the summarization process.

II. Background

• **React JS:** React JS is a popular JavaScript library for building user interfaces. It was developed by Facebook and is widely used for creating interactive and dynamic web applications. React JS follows a component-based architecture, allowing developers to build reusable UI components and efficiently manage state changes. It uses a virtual DOM (Document Object Model) for rendering UI updates, resulting in better performance and user experience.

• **Vite:** Vite is a modern build tool for web applications. It focuses on fast development and fast builds, making it an ideal choice for React JS projects. Vite leverages native ES modules to deliver quick development server startup times and optimized production builds. It also provides features like hot module replacement (HMR) for faster code updates during development.

• **Tailwind CSS:** Rapid UI creation is made possible with Tailwind CSS, a utility-first CSS framework. The Tailwind CSS framework offers a set of utility classes that may be used to produce unique designs, in contrast to conventional frameworks that use pre-designed elements. There are numerous pre-built utility classes available for styling components, responsive layouts, and other things. Tailwind CSS encourages a mobile-first strategy and makes it easier to create UI/UX that is both aesthetically pleasing and responsive.

• **RTK Query:** RTK Query is a state management library built on top of Redux Toolkit. It simplifies data fetching and caching in React applications by providing a powerful and opinionated API for managing remote data. RTK Query helps in managing API requests, caching responses, and handling loading and error states. It integrates well with React JS and enables efficient handling of complex API interactions.

• **Local Storage:** Local storage is a web browser feature that allows data to be stored locally on the user's device. It provides a simple key-value storage mechanism that can persist data across browser sessions. In the context of the project, local storage can be utilized to save user history, such as previously summarized articles or user preferences. It enables a seamless user experience by preserving data even after page refresh or browser closure.

III. Motivation

In today's digital age, the vast amount of information available online can be overwhelming. Users often struggle to keep up with the constant influx of articles, blog posts, research papers, and news updates. Automated article summarization provides a solution by condensing lengthy articles into concise summaries, allowing users to quickly grasp the main points and decide which articles are worth further exploration. It can take a while to manually summarize and analyze an article, especially if there is a lot of language involved. Users can significantly reduce the amount of time and effort spent retrieving pertinent information by automating the summary process. The goal of this project is to create an effective and user-friendly system that can provide summaries in real-time, giving users rapid and easy access to essential data. the motivation for this project lies in addressing the challenges posed by information overload, improving time efficiency, enhancing accessibility, leveraging advancements in AI, and delivering a user-friendly interface. By combining the power of Generative AI, React JS, and other technologies, the project strives to contribute to the development of an effective and efficient automated article summarization system

IV. System Architecture

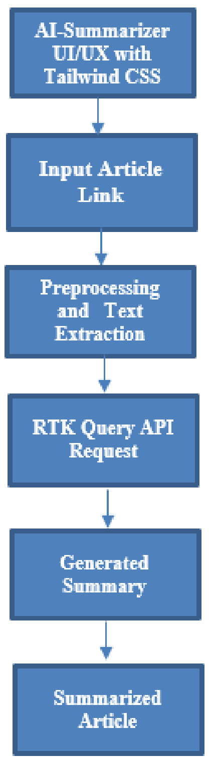


Figure 1: Conceptual Representation

In this figure 1, the AI summarizer takes an input article link as its source of information. The article goes through preprocessing and text extraction to prepare it for analysis. The model leverages advanced language understanding capabilities to generate a summary based on the extracted information. The generated summary is then presented as the output, providing a concise representation of the original article.

4.1. Generative AI Algorithm

Generative AI refers to a subset of artificial intelligence that focuses on generating new data instances that resemble a given dataset. Unlike other AI approaches that are focused on classification or prediction tasks, generative AI models aim to learn and recreate the underlying patterns and structures of the training data to produce new and unique data instances. Generative AI algorithms utilize probabilistic models and statistical techniques to understand and capture the data distribution. These models learn the patterns, correlations, and dependencies present in the training data, enabling them to generate new data samples that exhibit similar characteristics.

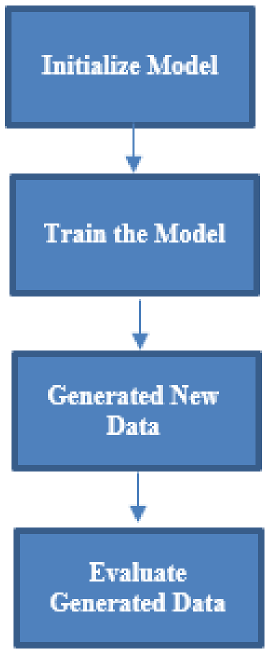


Figure 2: Generative AI Architecture

In the figure 2 below we made a simplified flowchart to explain the basic steps involved in a generative AI algorithm:

• **Initialize Model:** In this stage, the generative AI model, which could be a Generative Adversarial Network (GAN), a Variational Autoencoder (VAE), or any other generative model, is initialized. At this point, the parameters and architecture of the model are configured.

• **Training the Model:** A dataset of real data samples is used to train the model. In order to understand the underlying structures and patterns seen in the training data, the model's parameters must be optimized during the training phase. This aids the model's ability to reproduce the training data's data distribution and produce fresh data instances that are similar to it.

• **Generate New Data:** Once the model is trained, it can generate new data instances by sampling from the learned distribution. These generated instances may exhibit similar characteristics to the training data but are not exact replicas. The model uses random noise or latent variables as input to produce diverse outputs.

• **Evaluate Generated Data:** The quality and fidelity of the generated data can be assessed through various evaluation metrics or human judgment. This step helps assess how well the generative model has learned and whether the generated data aligns with the desired criteria.

V. UI/UX DESIGN

The design of a user-friendly and aesthetically pleasing user interface (UI) and user experience (UX) are essential components of making digital products` While UX design concentrates on improving user pleasure and usefulness by guaranteeing a seamless and intuitive user experience, UI design concentrates on the appearance and presentation of the user interface.

A number of principles are used in UI/UX design to create efficient and interesting interfaces. These guidelines consist of:

• **Consistency:** Maintaining consistency in the interface's design elements—such as colors, typography, and layouts—will give users a more seamless experience.
• **Simplicity:** Aiming for minimalism and simplicity in design to ease user understanding, increase usability, and lessen cognitive burden.
• **Visual Hierarchy:** Grouping and ranking interface components to direct user focus and enable simple navigation.
• **User feedback:** Giving users interactive and visual cues, including hover effects or button animations, to recognize their actions and make interactions more clear.
• **Accessibility:** Ensuring that the interface complies with accessibility standards and is usable by all users, including those with

disabilities, by offering suitable contrast, text alternatives, and keyboard accessibility.

5.1. Exploring the Concept of Glass Morphism in UI Design:

Glass morphism is a relatively recent trend in UI design that aims to create a modern and sleek appearance by simulating a glass-like effect on interface elements. It involves using translucent or semi-transparent elements with blurred backgrounds and subtle shadows to mimic the appearance of frosted glass or acrylic materials.

Glass morphism adds depth and visual interest to the interface, creating a sense of elegance and sophistication. It can be achieved using CSS properties like background-blur, box-shadow, and color overlays to create the desired glass-like effect.

Step-by-Step Guide to Designing a Visually Appealing and Responsive UI using Tailwind CSS:

1. **Set up the Project:** Start by setting up a new project and installing Tailwind CSS, a popular utility-first CSS framework.

2. **Define Layout and Components:** Plan the overall layout and structure of the UI, breaking it down into components. Identify the key UI elements, such as headers, navigation bars, content sections, forms, buttons, and cards.

3. **Utilize Tailwind CSS:** Leverage Tailwind CSS classes and utility-first approach to style the UI components. Utilize the extensive utility classes provided by Tailwind CSS for responsive layouts, typography, colors, spacing, and more.

4. **Implement Glass Morphism:** Apply the glass morphism effect to specific UI elements using CSS properties like background-blur, box-shadow, and color overlays. Experiment with different transparency levels, blur amounts, and shadow styles to achieve the desired glass-like appearance.

5. **Ensure Responsiveness:** Make sure the UI is responsive by designing it to adjust to various screen sizes and devices. To develop a responsive layout and successfully manage breakpoints, use Tailwind CSS's responsive utility classes.

6. **Test and iterate:** To make sure the UI is aesthetically pleasing, functional, and user-friendly, test it on various devices and screen sizes. On the basis of user testing and feedback, collect comments and iterate the design.

VI. RTK QUERY API INTEGRATION:

6.1 RTK Query Overview and Architecture:

Redux Toolkit (RTK) offers a potent data retrieval and caching package called RTK Query. By creating API request hooks and automatically controlling data caching, invalidation, and re-fetching, it makes API integration and state management simpler.

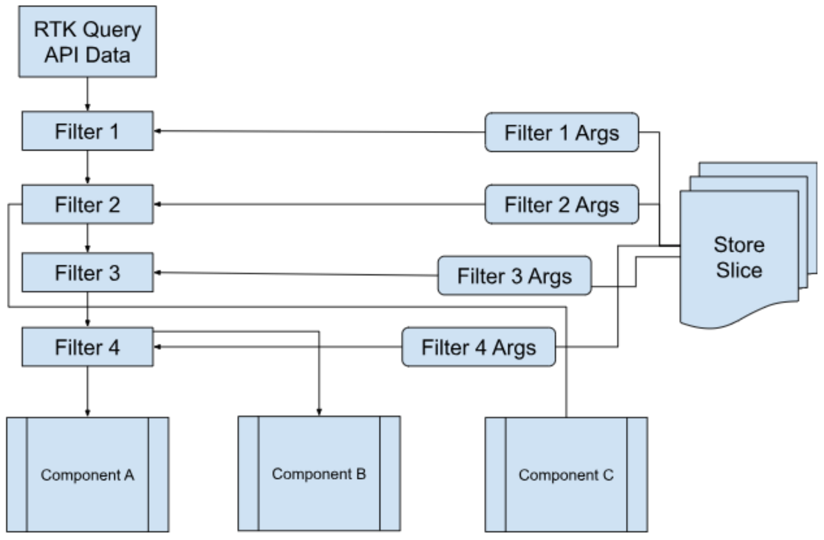


Figure 3: RTK Query Architecture

• We utilized query in each component and applied the necessary criteria in useEffect and useMemo. This is undesirable because it implies that at least two filters are repeated n times.

• We run the filter process after using the createSlice, extraReducers option and listening for query completion. This is great because I can use the filter arguments in the reducer, however once the filter arguments change but the query data doesn't, I don't think there is a way to repeat the operation with fresh arguments.

• After publishing data to a slice following each related filter stage, we subscribed one component, then subscribe each component to the associated data. This is how I presently have it set up, however it is not ideal because it couples components together that I want to avoid doing, bloats one component that was sort of chosen at random, and results in a lot of huge state activities that slow down my application.

• We passed data as props after raising the query subscription to the common ancestor component. The fact that these components are at different depths in relation to their common progenitor makes this less than ideal, and I assume that prop drilling would be necessary for at least some components.

• We also used react context to share results of first 2 filter operations with corresponding components. Haven't looked into this much yet; would it work with a query subscription?

• Intuitively i would think some callback that operates as some middleware between the API result and the component's subscribed data would be ideal. I am aware of the transform. Response option definable in the API slice, but I believe it is not appropriate or possible for this situation.

6.2 Implementing Advanced API Requests using RTK Query:

With RTK Query, we can implement advanced API requests, such as pagination, filtering, sorting, and custom queries, with ease. RTK Query provides a flexible syntax to define these advanced features:

• **Pagination:** It defines pagination parameters like page size and current page number in the endpoint configuration. RTK Query automatically handles pagination and provides pagination-related hooks and methods for navigating through the paginated data.

• **Filtering and Sorting:** It append query parameters for filtering and sorting to the endpoint configuration. RTK Query allows you to define query parameters and update them dynamically to fetch filtered and sorted data.

• **Custom Queries:** RTK Query supports custom queries where you can define specific parameters and structures required by your API. You can create custom endpoints and hooks to fetch data using your own query configurations.

VII. HISTORY AND LOCAL STORAGE MANAGEMENT

7.1 Storing and managing user history using local storage:

A web browser feature called local storage enables web apps to save data locally on a user's device. When a person navigates away from a web page or shuts their browser, it offers a straightforward key-value storage method that may be used to save and retrieve data.

To store and manage user history using local storage, you can follow these steps:

1. **Capture User History:** Whenever a significant event occurs in your application that you want to track as part of the user's history (e.g., page navigation, action completion), capture the relevant data or event details.

2. **Serialize and Store Data:** Convert the captured data into a serialized format, such as JSON, to ensure compatibility with local storage. Use the localStorage API provided by the web browser to store the serialized data. You can use a unique key to identify the stored history data.

3. **Retrieve User History:** To retrieve the user history, access the data from local storage using the key you assigned. Deserialize the data from its serialized form back into its original format for further processing or display.

4. **Update and Manage History:** As the user interacts with your application, continue capturing and updating the user's history data. You can append new events to the existing history or overwrite the entire history if needed.

7.2 Storing and managing user history using local storage:

For a seamless user experience and to guarantee mistake-free and seamless user interactions, form events and error handling must be implemented. Here are some actions to think about:

• We implement client-side form validation to make sure that user inputs adhere to the necessary standards (such as proper format, needed fields, and character limitations). Before submitting the data, use JavaScript to validate the form inputs.

• **Event Handling:** Attach event listeners to form components and log pertinent occurrences like form submissions, button clicks, and input changes. Based on the user's activities, employ event handling routines to carry out the proper actions.

• We implement error-handling tools to give the user feedback and assistance when mistakes happen. Display clear error messages next to the appropriate form fields or in a separate error area. Point out the faulty fields and offer instructions on how to fix the mistakes.

• **Real-Time Feedback:** As the user interacts with the form, provide them real-time feedback. As the user types, for instance, check certain input fields and give immediate feedback indicating whether the input is legitimate or invalid.

• **Clear and Reset:** We include a feature that allows users to opt to start over or to clear or reset form fields after submission. As a result, the interface will be clear and user-friendly for further interactions.

VIII. RESULTS AND EVALUATION

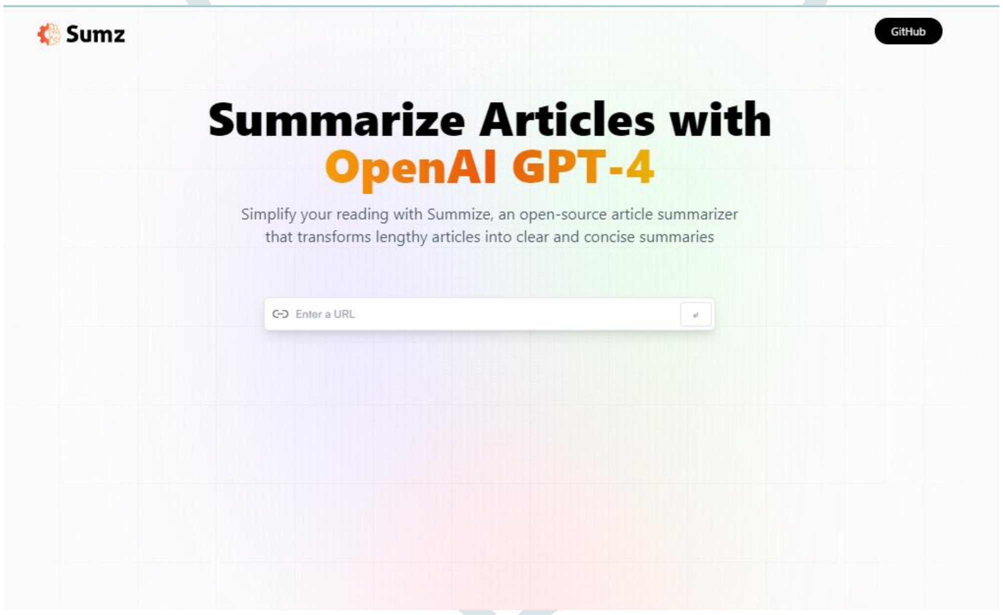
8.1 Testing and Evaluating the Application's Performance and Functionality of AI Summarizer:

1. **Functional Testing:** It conducts thorough testing of the AI Summarizer to ensure that it performs its core functionality correctly. Test various input articles of different lengths and complexities to verify the accuracy and quality of the generated summaries.

2. **Performance Testing:** It evaluates the performance of the AI Summarizer by measuring its response time and resource usage. Test it with different workloads and input sizes to assess its efficiency and scalability. Identify any bottlenecks or performance issues and optimize the algorithm or infrastructure as needed.

3. **Comparison with Human Summaries:** It compares the summaries generated by the AI Summarizer with human-generated summaries or existing benchmark datasets. Evaluate the similarity, coherence, and overall quality of the AI-generated summaries against human-written ones to assess its effectiveness.

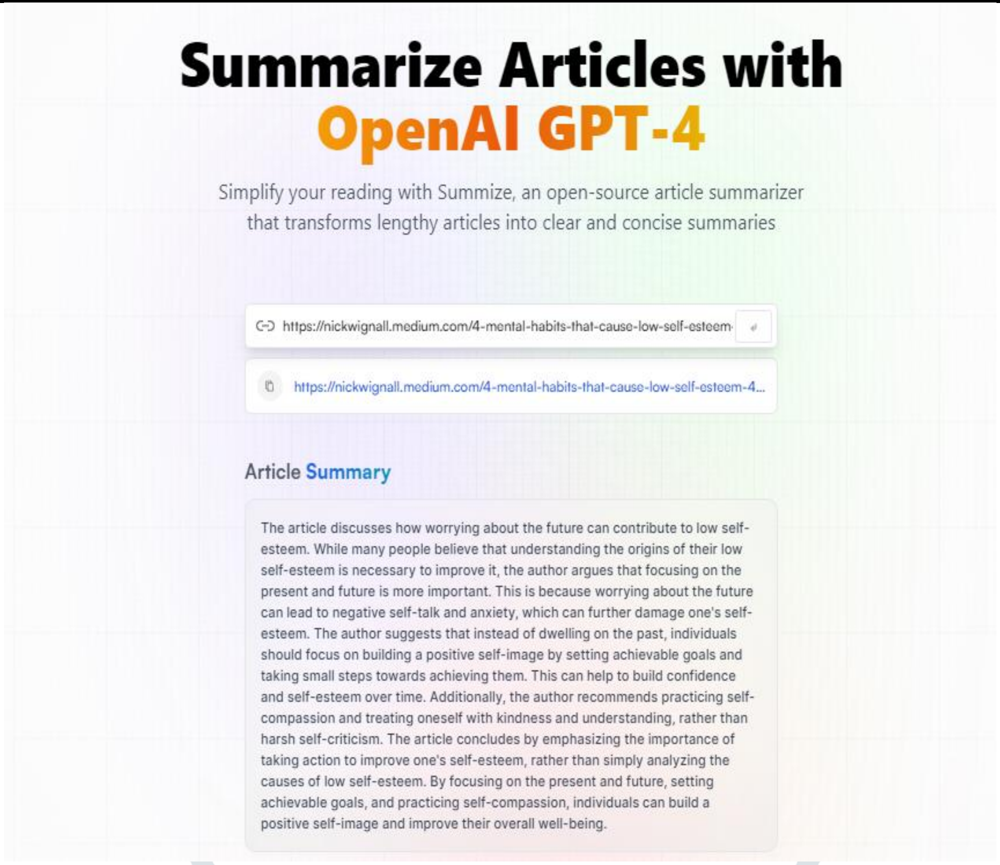
4. **Robustness Testing:** It performs robustness testing by introducing various edge cases, such as ambiguous or poorly structured articles, to evaluate how the AI Summarizer handles such scenarios. We also test its ability to gracefully handle unexpected inputs and produce meaningful summaries.



Summarize Articles with OpenAI GPT-4

Simplify your reading with Summize, an open-source article summarizer that transforms lengthy articles into clear and concise summaries

Figure 4: UI/UX of AI Article Summarizer


Summarize Articles with
OpenAI GPT-4
Simplify your reading with Summize, an open-source article summarizer that transforms lengthy articles into clear and concise summaries
https://nickwignall.medium.com/4-mental-habits-that-cause-low-self-esteem-
https://nickwignall.medium.com/4-mental-habits-that-cause-low-self-esteem-4...
Article Summary

The article discusses how worrying about the future can contribute to low self-esteem. While many people believe that understanding the origins of their low self-esteem is necessary to improve it, the author argues that focusing on the present and future is more important. This is because worrying about the future can lead to negative self-talk and anxiety, which can further damage one's self-esteem. The author suggests that instead of dwelling on the past, individuals should focus on building a positive self-image by setting achievable goals and taking small steps towards achieving them. This can help to build confidence and self-esteem over time. Additionally, the author recommends practicing self-compassion and treating oneself with kindness and understanding, rather than harsh self-criticism. The article concludes by emphasizing the importance of taking action to improve one's self-esteem, rather than simply analyzing the causes of low self-esteem. By focusing on the present and future, setting achievable goals, and practicing self-compassion, individuals can build a positive self-image and improve their overall well-being.

Figure 5: Summary of an Article

8.2 User Feedback and Potential Improvements:

• **User Testing and Feedback:** We gathered feedback from users who have used the AI Summarizer. Conduct surveys, interviews, or usability tests to understand their experience, satisfaction level, and any issues they encountered. Collect qualitative and quantitative feedback to identify strengths, weaknesses, and areas for improvement.

• **Feedback Analysis**: It analyzes the user feedback to identify recurring patterns, common pain points, or areas of confusion. Prioritize user feedback based on the frequency and severity of reported issues.

• **Enhancing Summary Quality:** It analyzes the generated summaries to identify any areas where the quality can be improved. Explore techniques such as fine-tuning the model on domain-specific data or incorporating external linguistic resources to enhance the coherence, readability, and relevance of the summaries.

• **Addressing Limitations:** It identifies limitations or shortcomings of the AI Summarizer, such as difficulty in handling long articles or bias in the generated summaries. Developing strategies to address these limitations, such as breaking down long articles into smaller chunks or implementing bias mitigation techniques.

• **Iterative Development:** We continuously iterate on the AI Summarizer based on user feedback and evaluation results. Implement regular updates and improvements to address identified issues and enhance the overall user experience.

By testing, evaluating, and incorporating user feedback, we can improve the performance, functionality, and user satisfaction of the AI Summarizer. This iterative process ensures that the summarization application evolves and adapts to meet the needs of its users.

CONCLUSION

In conclusion, starting this paper is a great chance to learn a lot about a variety of topics while also creating a thorough application. You will gain knowledge about establishing a ReactJS project utilizing Vite, a cutting-edge and effective tooling system, by working on this project.

We will also get a taste of UI/UX design through making responsive and visually appealing interfaces and discover how to provide our application a contemporary and slick appearance by using the well-liked glass morphism design trend using Tailwind CSS. The paper also focuses on enhancing functionality by implementing advanced RTK query API requests. You will gain experience in creating queries that trigger based on specific conditions, allowing for dynamic data retrieval and customization.

Additionally, this work emphasizes the importance of data management and user experience. You will learn how to leverage local storage to store and manage user history, ensuring a seamless and personalized experience. Furthermore, you will implement form event handling and error management, enabling smooth user interactions and providing helpful feedback.

Lastly, the paper emphasizes writing organized code and even explores implementing features like copy to clipboard. By following best practices and maintaining clean and structured code, you will develop not only a functional application but also ensure its maintainability and scalability. Overall, by mixing several facets of ReactJS development, UI/UX design, API integration, data management, and code organization, this project provides a comprehensive learning experience. By working on this project, you will not only create a useful application but also learn useful skills that you may use on other projects in the future, which will advance your development career.

REFERENCES

[1] Agarwal, H. 2022. ReactJS Components. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/reactjs-components/. Accessed 17 December 2022.

[2] Anthony, A., Nathaniel, M., & Lerner, A. 2017. Fullstack React: The Complete Guide to ReactJS and Friends. Fullstack.IO.

[3] Borusiuk, Y. 2022. Top 10 Benefits of React.js for Application Development. Online. NCube. Available at: https://ncube.com/blog/top-10-benefits-of-react-js-for-application-development. Accessed 03 March 2022.

[4] Chi, C. 2021. "A Beginner's Guide to Data Flow Diagrams". Blog. Hubspot. Available at: https://blog.hubspot.com/marketing/data-flow-diagram. Accessed June 10, 2023.

[5] Didacus O. 2018. System Design in Software Development. Online. Medium. Available at: https://medium.com/the-andela-way/system-design-in-software-development-f360ce6fcbb9. Accessed 21 March 2023.

[6] Document Object Model (ODM), 2023. Tips and Tricks. Meklit. Available at: http://www.meklit.com/index.php/document-object-model-odm. Accessed 05 December 2022.

[7] Eygi, C. 2020. React.js for Beginners — Props and State Explained. Online. freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/react-js-for-beginners-props-state-explained/. Accessed 29 January 2023.

[8] Hammad, M. 2020. "Use Case Diagram for Library Management System". Preprint. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/use-case-diagram-for-library-management-system/. Accessed June 13, 2023.

[9] Herbert, D. 2022. React provides state-of-the-art user interfaces. Blog post. HubSpot. Available at: https://blog.hubspot.com/website/react-js#:~:text=React%20provides%20state%2Dof%2Dthe,time%20with%20JavaScript%2Ddriven%20pages. Accessed 22 November 2022.

[10] Javatpoint. 2021. ReactJS State Vs Props. Online. Javatpoint. Available at: https://www.javatpoint.com/react-state-vs-props. Accessed 10 February 2023.

[11] Javatpoint. 2021. ReactJS State - Javatpoint. Online. www.javatpoint.com. Available at: https://www.javatpoint.com/react-state. Accessed 03 February 2023.

[12] Levai, D. 2022. Front-end-vs-back-end-development-the-ultimate-2022-guide. Online. Screamingbox. Available at: https://screamingbox.com/front-end-vs-back-end-development-the-ultimate-2022-guide/. Accessed June 13, 2023. 37

[13] MyEdu. 2019. "Importance of Library Management System for Education Institutes,". Preprint. MyEdu. Available at: https://www.myeducomm.com/blog/importance-of-library-management-system/. Accessed June 13, 2023.

[14] Norton, S. 2021. Getting Started with Bootstrap 5, React, and Sass. Online. Designmodo. Available at: https://designmodo.com/bootstrap-react-sass/. Accessed 27 March 2023.

[15] Oracleappshelp. 2021. React JS programming tutorial for Beginners. Available at: http://oracleappshelp.com/react-js-programming-tutorial-for-beginners/. Accessed: 25 November 2022.

[16] Osarome, O. 2011. "1. Technical feasibility 2. Operational feasibility 3. Economic feasibility". Blog. Osarome. Available at: https://osarome.blogspot.com/2011/10/1-technical-feasibility-2-operational.html. Accessed: June 13, 2023.

[17] Pandit, N. 2021. What and Why ReactJS. Online. C# Corner. Available at: https://www.c-sharpcorner.com/article/what-and-why-reactjs/. Accessed: 29 November 2022.

[18] After Academy. (n.d.). Mastering Mongoose for MongoDB and Node.js. Accessed 15.4.2023. https://afteracademy.com/blog/mastering-mongoose-for-mongodb-and-nodejs/.

[19] Luukainen. (2023). Deploying app to internet. Accessed 18.5.2023. https://fullstackopen.com/en/part3/deploying_app_to_internet. Material UI. (2023).

[20] Material UI Documentation. Accessed 14.4.2023. https://mui.com/material-ui/react-button/.

[21] Mongoose. (n.d.). Mongoose Documentation. Accessed 15.4.2023. https://mongoosejs.com/.

[22] Netlify. (n.d.). Netlify Documentation. Accessed 18.5.2023. https://docs.netlify.com/.

[23] Scaler 2023. (n.d.). Scaler 2023. Prop Drilling in React. Accessed 20.4.2023. https://www.scaler.com/topics/react/prop-drilling-in-react/.

[24] Staton. (2022). Students battle to stay afloat in cost of living crisis. Accessed 10.4.2023. https://www.ft.com/content/910c4c1a-df97-49a8-8422-2849d0179da0.

[25] Vailshery. (2022). Most popular web frameworks among developers worldwide 2022. Accessed 14.4.2023. https://www.statista.com/statistics/1124699/worldwidedeveloper-survey-most-used-frameworks-web/.

[26] Wikipedia, Database. (n.d.). Database. Accessed 14.4.2023. https://en.wikipedia.org/wiki/Database.

[27] Wikipedia, npm. (n.d.). npm (software). Accessed 14.4.2023. https://en.wikipedia.org/wiki/Npm_(software).

[28] Workfall. (2022). How to build and deploy a MERN Stack Application on AWS?. Accessed 14.4.2023. https://www.workfall.com/learning/blog/how-to-build-and-deploy-amern-stack-application-on-aws/.

[29] Ahmad, A., Waseem, M., Liang, P., Fehmideh, M., Aktar, M.S., Mikko-nen, T.: Towards human-bot collaborative software architecting with chatgpt. arXiv preprint arXiv:2302.14600 (2023)

[30] Amin, M.M., Cambria, E., Schuller, B.W.: Will a ective computing emerge from foundation models and general ai? a rst evaluation on chatgpt. arXiv preprint arXiv:2303.03186 (2023)

[31] Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., Lovenia, H., Ji, Z., Yu, T., Chung, W., et al.: A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. arXiv preprint arXiv:2302.04023 (2023)

[32] Basic, Z., Banovac, A., Kruzic, I., Jerkovic, I.: Better by you, better than me, chatgpt3 as writing assistance in students essays. arXiv preprint arXiv:2302.04536 (2023)

[33] Belouadi, J., Eger, S.: Bygpt5: End-to-end style-conditioned poetry gen-eration with token-free language models. arXiv preprint arXiv:2212.10474 (2022)

[34] Blanco-Gonzalez, A., Cabezon, A., Seco-Gonzalez, A., Conde-Torres, D., Antelo-Riveiro, P., Pineiro, A., Garcia-Fandino, R.: The role of ai in drug discovery: Challenges, opportunities, and strategies. arXiv preprint arXiv:2212.08104 (2022)

[35] Borji, A.: A categorical archive of chatgpt failures. arXiv preprint arXiv:2302.03494 (2023)

[36] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems 33, 1877{1901 (2020)

[37] Chen, N., Wang, Y., Jiang, H., Cai, D., Chen, Z., Li, J.: What would harry say? building dialogue agents for characters in a story. arXiv preprint arXiv:2211.06869 (2022)

[38] Chen, Y., Eger, S.: Transformers go for the lols: Generating (humourous) titles from scienti c abstracts end-to-end. arXiv preprint arXiv:2212.10522 (2022)

[39] Christiano, P.F., Leike, J., Brown, T., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. Advances in neural information processing systems 30 (2017)

[40] Dai, H., Liu, Z., Liao, W., Huang, X., Wu, Z., Zhao, L., Liu, W., Liu, N., Li, S., Zhu, D., et al.: Chataug: Leveraging chatgpt for text data augmentation. arXiv preprint arXiv:2302.13007 (2023)

[41] Du, X., Cardie, C.: Event extraction by answering (almost) natural ques-tions. arXiv preprint arXiv:2004.13625 (2020)

[42] Freitag, M., Rei, R., Mathur, N., Lo, C.k., Stewart, C., Avramidis, E., Kocmi, T., Foster, G., Lavie, A., Martins, A.F.: Results of wmt22 met-rics shared task: Stop using bleu{neural metrics are better and more ro-bust. In: Proceedings of the Seventh Conference on Machine Translation (WMT). pp. 46{68 (2022)

[43] Freitag, M., Rei, R., Mathur, N., Lo, C.k., Stewart, C., Avramidis, E., Kocmi, T., Foster, G., Lavie, A., Martins, A.F.: Results of wmt22 met-rics shared task: Stop using bleu{neural metrics are better and more ro-bust. In: Proceedings of the Seventh Conference on Machine Translation (WMT). pp. 46{68 (2022)

[44] Frieder, S., Pinchetti, L., Gri ths, R.R., Salvatori, T., Lukasiewicz, T., Petersen, P.C., Chevalier, A., Berner, J.: Mathematical capabilities of chatgpt. arXiv preprint arXiv:2301.13867 (2023)

[45] Fu, Q., Teng, Z., Georgaklis, M., White, J., Schmidt, D.C.: Nl2cmd: An updated work ow for natural language to bash commands translation. arXiv preprint arXiv:2302.07845 (2023)

[46] Gao, J., Zhao, H., Yu, C., Xu, R.: Exploring the feasibility of chatgpt for event extraction. arXiv preprint arXiv:2303.03836 (2023)

[47] Glymour, C., Zhang, K., Spirtes, P.: Review of causal discovery methods based on graphical models. Frontiers in Genetics (2019)

[48] Gormley, M.R., Yu, M., Dredze, M.: Improved relation extrac-tion with feature-rich compositional embedding models. arXiv preprint arXiv:1505.02419 (2015)

JETIR