COMP4801 Final Year Project

Interim Report

LLM-Enhanced Cross-Platform Web Search Application Combining AI and Traditional Search Techniques

Supervisor: Professor Heming Cui

Student: Sze Shing Fung (3035930060)

Date of submission: 2025-01-26

i. Abstract

The rapid advancement of technology is transforming online information retrieval, particularly through the emergence of chatbots powered by large language models (LLMs) that challenge traditional search engines, which often produce less concise results and can be influenced by search engine optimization techniques. This report proposes an innovative cross-platform web search application that utilizes LLMs to enhance the quality of search results retrieved from search engine APIs. The project aims not only to generate concise summaries of search results retrieved from search engines but also to reorder them based on content relevance, while possibly mitigating the limitations associated with LLM-powered chatbots, including their knowledge cutoff dates and the hallucination problem where LLMs sometimes present false information as the truth to users. The preliminary prototype, which utilizes the Bing Search API alongside DeepSeek-V3, generates LLM-based summaries of search results and demonstrates the proposed method's potential to address user needs more effectively than traditional search engines and LLM-powered chatbots. This approach could pave the way for more reliable and user-centric information retrieval systems that can meet the evolving demands of users.

ii. Acknowledgement

I would like to express my heartfelt gratitude to Professor Heming Cui for his invaluable support throughout this project. His guidance was instrumental in helping me navigate the challenges of my research.

I am also grateful to Mr. Guichao Zhu for his insightful suggestions on the architecture of the and new features of the project.

Additionally, I wish to extend my deepest thanks to my parents for their unwavering love and support. Their belief in my abilities has continually motivated me to strive for excellence in my studies. This accomplishment would not have been possible without them.

iii. Table of Contents

iv. List of Figures

v. List of Tables

vi. Abbreviations

Abbreviation	Definition
API	Application Programming Interface
AWS	Amazon Web Services
AWS SAM	AWS Serverless Application Model
CSS	Cascading Style Sheets
Fig.	Figure
HTML	Hypertext Markup Language
LLM	Large Language Model
MRR	Mean Reciprocal Rank
Sec.	Section
SEO	Search Engine Optimization
UI	User Interface
URL	Uniform Resource Locator

1. Introduction

The rapid evolution of technology is transforming how users seek information online. According to Gartner [4], AI-powered chatbots are expected to increasingly replace traditional search engines like Google in the coming years. These chatbots often utilize large language models (LLMs) such as ChatGPT and Claude, which are pre-trained language models that use the probability of word sequences in their training data to generate outputs [5]. Unlike conventional search engines that display a list of links, chatbots offer concise and direct answers to their queries, thus enhancing the efficiency of users' search experience.

While LLM-powered chatbots offer advantages in information retrieval, they also present significant challenges. One primary concern is that chatbots rely on LLMs, which typically have a knowledge cutoff date, rendering them unable to provide the most up-to-date information. For instance, the knowledge cutoff for GPT-4o-Mini is October 2023 [6]. This limitation can hinder users seeking current information, as illustrated in Fig. 1.1 where GPT-4o-Mini failed to provide information about the Prime Minister of the United Kingdom in 2024 due to the cutoff date.

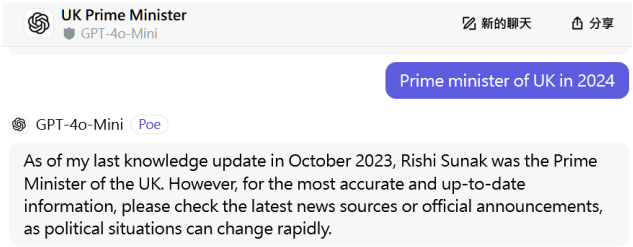

Fig. 1.1 Screenshot from Poe with GPT-4o-Mini [1], indicating it can't answer questions about current events beyond the knowledge cutoff.

Moreover, while some chatbots like Perplexity leverage current web sources to mitigate the knowledge cutoff issue, all LLM-powered chatbots ultimately face a critical problem: hallucinations. Hallucinations occur when LLMs generate inaccurate or misleading information, which can mislead users. Research by Yao et al. [7] highlights the vulnerability of LLMs to adversarial attacks, achieving a success rate of 53.85% against LLaMA2-7B-chat. They further suggest that hallucinations may be an inherent flaw in LLMs, making their eradication a complex challenge.

Additionally, the user interface (UI) of many chatbots, such as Poe, often lacks source attribution for the information provided and presents their answers as the absolute truth. Some

chatbots, like Perplexity, display sources alongside their answers but still predominantly portray their summaries as definitive by not providing any alternative result (see Fig. 1.2). This can lead users to accept chatbot responses without critical evaluation or verification of the information, which may be inaccurate or false due to the hallucination problem.

Fig. 1.2 Screenshot from Perplexity [2] showing the query, presenting the current answer and sources, but doing so in a way that may suggest it as the sole truth.

Conversely, traditional search engines like Google and Bing present results as a list of links and brief descriptions. They face ongoing challenges related to search engine optimization (SEO) tactics and spam content. A comprehensive review of major search engines conducted between 2022 and 2023 [8] revealed that top-ranking results often prioritize SEO strategies that may compromise content quality. As of October 1, 2024, Google has implemented two spam updates and two core updates in an effort to enhance search quality [9]. However, findings [8] indicate that these updates only temporarily alleviate spam issues, leaving users with a subpar search experience between updates.

Furthermore, traditional search engines can struggle with complex queries, often requiring users to sift through multiple results to find relevant information. For example, when querying “generate multiple HTML and CSS files in Vite,” GPT-4o mini provided detailed, tailored responses, while Google’s top results included links to forums like Stack Overflow, which may not address the users’ specific needs (see Fig. 1.3).

Given the strengths and weaknesses of both chatbots and traditional search engines, this project aims to develop an innovative cross-platform web search application. The application will retrieve search results from search engine APIs such as the Bing Web Search API.

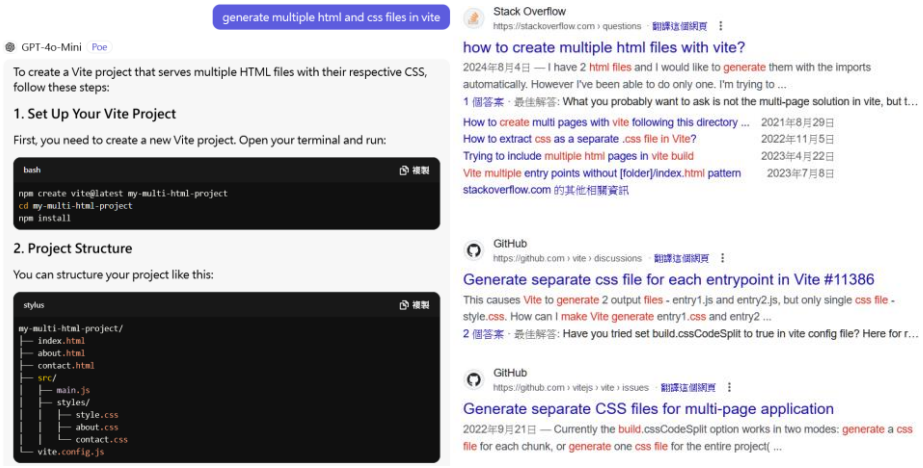

Fig. 1.3 (Left) Prompt result from Poe with GPT-4o-Mini [1] with prompt “generate multiple html and css files in vite”. (Right) Search result from Google with a search query same as the GPT-4o-Mini prompt.

results. Then, the system will generate concise summaries for each individual result alongside a comprehensive overview of the search query.

The deliverables for this project will include:

1. a responsive web application and a mobile application in the form of a .apk file that will serve as the primary interfaces for users,
2. a backend system to support data storage and processing, and
3. evaluations that determine the best prompting strategies and LLMs for the project, as well as the effectiveness of the application.

The next section of this report outlines the methodology (Sec. 2) that will guide the project from conception to deployment. It covers design considerations and implementation strategies for both the frontend and backend, followed by the evaluation metrics. Subsequently, the report presents the current results (Sec. 3), detailing the frontend implementation and the backend implementation, as well as the challenges encountered thus far. Finally, the report concludes with the future project schedule (Sec. 4) and an overall summary (Sec. 5).

2. Methodology

To support the application’s use cases (Sec 2.1), the application requires the implementation of a frontend (Sec. 2.2) and a backend (Sec. 2.3). Additionally, evaluations (Sec. 2.4) will be performed during development to guide the selection of LLMs and prompts used in the backend. The use cases, implementation strategies, the evaluation metrics and the source

control tools used in the project (Sec 2.5) are detailed below.

2.1. Use Cases

2.1.1. Search Query

The application's main use case is handling user search queries. For example, as a user enters a query such as "current weather" in the frontend, the backend retrieves a list of URLs from traditional search engines with the user query, parse the content of each webpage, and generate a summary for each URL. A general summary consolidating all retrieved information is then generated. Finally, the general summary, the webpage URLs and the webpage summaries are presented to the user in the format of a traditional search engine. This use case has been implemented, with details provided in Section 3.

2.1.2. Follow-Up Queries to LLM

In addition to the primary use case, the application will allow users to ask follow-up queries based on the search results. For example, after searching for "the current weather," the user will be able to ask a related question like "what should I wear?" directly under the general summary. The planned implementation is illustrated in Fig. 2.1: The frontend will capture the user's query under the general summary and retrieves the compressed partial summaries of webpages stored in the user's browser local storage during the initial search. These summaries, along with the query, will be sent to the backend via an HTTP request. The backend will then use LLMs to generate a relevant answer, which is then returned to the frontend as an HTTP response and displayed to the user.

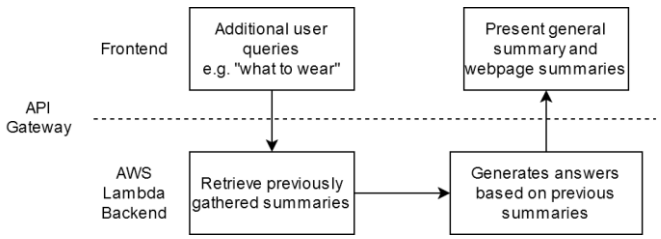

Fig. 2.1 Overview of the planned implementation of the follow-up query use case.

2.1.3. Recursive Search

When users provide vague or suboptimal search queries, traditional search engine APIs may return insufficient results. For example, a query like "most popular Linux distros info" might yield many articles listing Linux distributions without detailed information about the distributions themselves. To address this, the backend will be able to prompt the user to refine

their query or perform additional related searches when deemed necessary by LLMs. Additionally, the backend will use LLMs to automatically detect when the initial search results are inadequate. In such cases, the backend should generate refined search queries using LLMs, perform searches via the traditional search engine API, and feed the results back to the LLM for further processing. Finally, the user will be presented with an enhanced general summary, along with the search results of each recursive search, ensuring the search results to be comprehensive and relevant.

2.2. Frontend Implementation

2.2.1. Framework

This project will use Expo, built on React Native, to develop the frontend. Expo simplifies cross-platform development by allowing a single codebase for Android, iOS, and web applications, reducing development time. Unlike React Native, which natively supports only Android and iOS, Expo extends this capability to include web output, ensuring a unified experience across all platforms. Additionally, Expo projects include the Expo Router, which enables the implementation of navigation patterns such as tabs and stacks with a single definition of React Native component.

2.2.2. Programming Language

The frontend will be developed using TypeScript. It provides type safety, which helps prevent logic errors in codes. Additionally, TypeScript's type definitions for components, objects and functions can enhance the readability and maintainability of the project's code.

2.2.3. UI and Mobile-First Design

As of 2016, the majority of searches on Google are performed through mobile devices [10]. Furthermore, Google switched to prioritizing mobile versions of websites in indexing in 2020 [11]. This demonstrates the importance of providing a great mobile searching experience alongside a web frontend when building a web searching platform.

As most users are expected to be using mobile devices, the project will take a mobile-first approach to UI design. Responsive design, where the webpage layouts change with dimensions of the display [12], will be used to provide a seamless experience across devices with different sizes and aspect ratios.

2.3. Backend Implementation

2.3.1. Backend Framework and Technologies

This project will implement its backend using managed cloud services. Compared to setting up local servers, this reduces the time required for maintenance and enhances the scalability of the service. It also reduces security concerns through ways such as providing DDoS protection.

This project will use Amazon Web Services (AWS) as the cloud computing platform due to its leading market position. As illustrated in Fig. 2.2, Amazon holds 32% of the market as of Q2 2024 [3], nearly matching the combined market share of all other providers outside the top three. The extensive adoption of AWS ensures a wealth of online documentation and support, while minimizing the risk of service discontinuation.

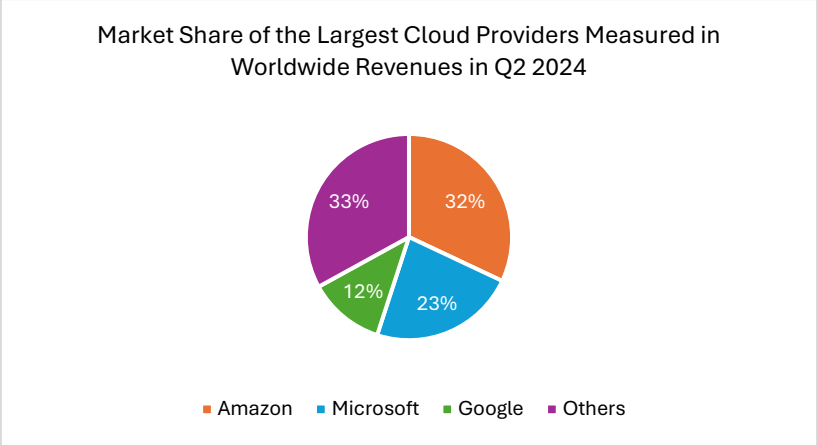

Fig. 2.2 Market share of the largest cloud service providers measured in worldwide revenues in Q2 2024 [3]. The chart shows that Amazon holds a dominant position in the market.

AWS Lambda will be used to host the backend. AWS Lambda primarily charges compute costs with little to no ongoing running costs [13]. This minimizes idle costs and is particularly well-suited for this project, as the user base is expected to be initially small. As the user base grows, AWS Lambda's ability to scale seamlessly ensures the backend services can continue to run without disruption.

Amazon API Gateway will be used to create REST API endpoints for frontend communication. AWS Lambda provides libraries that allow access to API Gateway requests and responses. This allows the integration of the backend codes with the API endpoints, enhancing code readability.

To streamline the development and deployment of the backend, this project will utilize the AWS Serverless Application Model (AWS SAM). AWS SAM is a framework that allows

developers to define cloud infrastructures, including AWS Lambda functions, APIs, and other resources, using a single YAML template file. Additionally, AWS SAM enables local testing of Lambda functions without deploying to the cloud, significantly speeding up the development cycle.

2.3.2. Programming Language

The backend will be developed using TypeScript. In addition to the advantages mentioned before, sharing the language in both the frontend and the backend enables the reuse of codes and type definitions, which helps to reduce development time.

2.3.3. Backend System Architecture

The project currently includes the following systems in the backend to support its main use case:

1. **Backend REST APIs**

REST APIs will be developed to facilitate communication between the frontend and the backend systems. The APIs are developed with AWS SAM and will be deployed with Amazon API Gateway.

2. **Web search and parsing system**

This system fetches relevant website links and extracts text content from the pages. While search engine APIs may provide snippets of search results, they often lack sufficient detail, especially for single-page applications where content may not load without JavaScript. To address this, an effective web parsing system is essential for retrieving the text data needed for LLM processing. Libraries like Puppeteer are used to dynamically load JavaScript content. Then, HTML parser such as Cheerio is used to parse the returned HTML and extract the relevant body text for further processing by the summary generation systems.

3. **Partial summary generation system**

Generating a general summary directly from the HTML contents of all search results would require an impractically large context length for the LLM. To optimize this, the system first generates a concise partial summary for each webpage, extracting key information. These partial summaries serve as inputs for both the individual webpage summaries and the general summary, enabling efficient and focused processing of search results while minimizing the context length required for the LLM.

4. **Webpage summary generation system**

This system generates a summary for each webpage by querying the LLM with the parsed content obtained from the web search and parsing system. It leverages the partial summaries to produce concise and relevant webpage summaries.

5. **General summary generation system**

The system uses LLMs to generate a cohesive and concise general summary by consolidating the partial summaries of all search results. This delivers a chatbot-like experience to the user, providing clear, to-the-point answers to their search query.

2.4. Evaluation Metrics

An essential aspect of the project is generating a general summary of the search results. To evaluate its accuracy, automatic evaluation tools will be used, reducing the need for human evaluators. Shen et al. [14] demonstrated that using LLMs to evaluate LLM-generated results outperforms other common automatic evaluation methods. Therefore, a head-to-head comparison approach, as described in [14], will be employed: both the application and an LLM will generate responses, and an LLM will rank the two to assess their quality.

However, Shen et al. also found that LLMs may not align well with human evaluators when both compared summaries are of high quality. In addition, these methods cannot be directly used to evaluate the webpage summaries as well as the platform's user experience. To address these issues, a user test will be conducted at the project's conclusion to further evaluate the application's effectiveness. Participants will be given a set of search queries and asked to rate the search results from the application, a search engine (e.g., Google), and an LLM-powered chatbot (e.g., GPT-4) on a scale from 1 (worst) to 5 (best) based on how quickly they retrieve relevant information.

2.5. Source Control

Git is used as the source control tool. A public repository has been hosted on GitHub for this project (URL: https://github.com/hoverGecko/LLMSearch).

3. Results

This section outlines the current outcomes of the project implementation of the use cases (Sec 3.1), detailing the frontend implementation (Sec 3.2) and the backend implementation (Sec 3.3), along with the challenges encountered during the project (Sec 3.4).

3.1. Use Cases

The primary use case of the application, handling user search queries, has been implemented and tested locally, with deployment on AWS planned for the following week. Preliminary results demonstrate the application's potential to be more effective and efficient in data retrieval compared to traditional search engines. For example, as shown in Fig. 3.1, when queried with "compare climate of Hong Kong with Singapore," Bing returns filler texts that provide little meaningful information for each webpage. In contrast, the prototype generates a meaningful comparison of the climates between the two cities, both in the general summary and in the summaries of individual webpages.

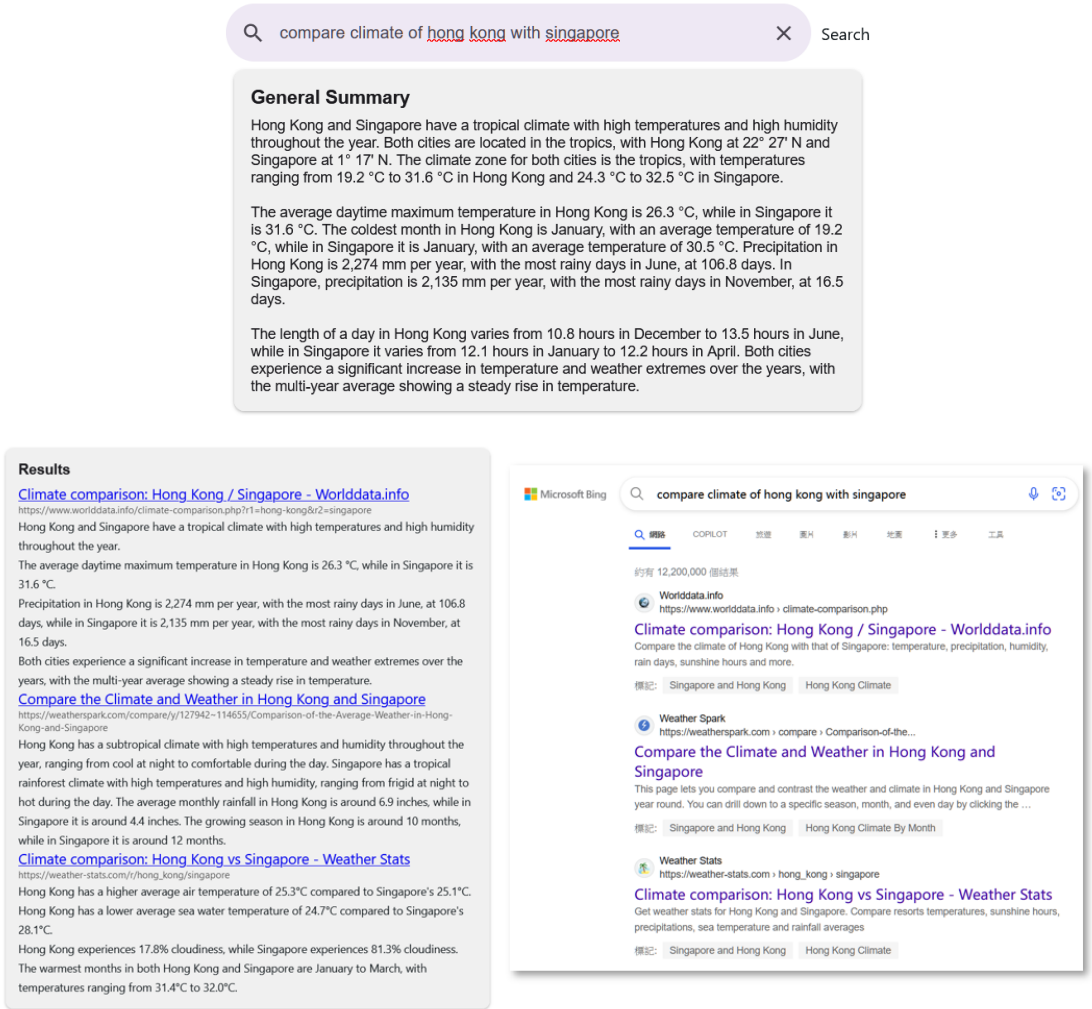

Fig. 3.1 Search results from the prototype (top, left) and Bing (right) with the query "compare climate of hong kong with singapore".

Additional use cases, such as handling follow-up user queries and recursive search, will be implemented in the future.

3.2. Frontend Implementation

A frontend prototype has been successfully developed using Expo, enabling deployment on both mobile and web platforms. The project used the React Native Paper library, which provides various React Native UI components including buttons, containers and search bars based on Material Design. Responsive design was implemented to ensure the interface adapts seamlessly to various devices, as illustrated in Fig. 3.2, where search results are displayed in common aspect ratios for desktops (16:9) and mobile phones (1:2).

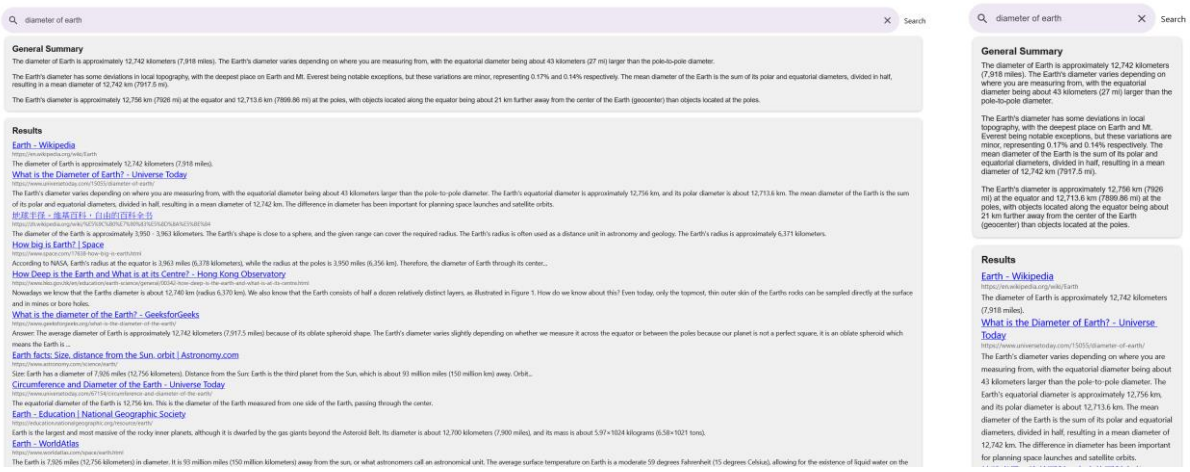

Fig. 3.2 Search results with query “diameter of earth” in the prototype in widescreen and in 1:2 (Right). The first result’s description is replaced by an LLM summary.

The prototype currently features a main page with a search bar (see Fig. 3.3) for user queries, presenting the general summary and the search results in a traditional format with URLs and a summary for each webpage, similar to conventional search engines. Future enhancements will include modification to facilitate other use cases, such as allowing the users to query the LLM after a search.

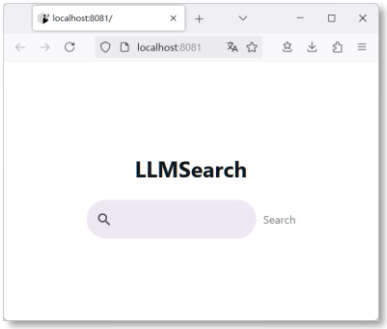

Fig. 3.3 Front page of the application.

3.3. Backend Implementation

Among the backend systems outlined in the methodology, backend REST APIs have implemented with AWS SAM and tested locally to support the application's primary use case, which is handling user search queries. Fig. 3.4 shows an overview of the implementation. First, the frontend captures a user's search query, such as "current weather", and sends it to the backend via an HTTP request.

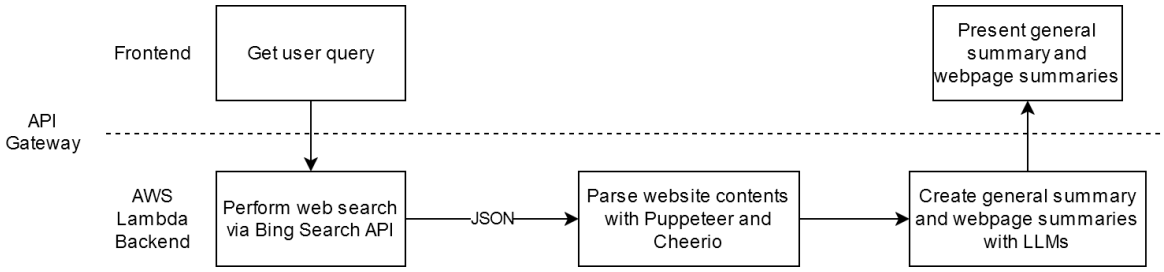

Fig. 3.4 Overview of the current implementation of the search query use case.

The backend retrieves search results from a traditional search engine API, returning a list of URLs. It then processes each webpage associated with these URLs. Directly passing the fetched HTML content to LLMs for summary generation is unfeasible for two reasons:

1. Some webpages, such as single-page applications, require JavaScript to dynamically load meaningful content.
2. The large size of the combined HTML content of multiple webpages would likely exceed the context length limit of most LLMs and incur a high running cost.

To address these challenges, the backend processes each URL as illustrated in Fig. 3.5: Puppeteer is used to handle JavaScript-dependent webpages by launching a headless Chromium browser and loading each webpage in a browser tab. Cheerio then parses the HTML plaintext returned by Puppeteer, removing unnecessary elements such as image tags and hyperlinks. Using this cleaned data, the backend uses DeepSeek-V3 to generate a partial summary for each webpage. These partial summaries are subsequently passed to DeepSeek-V3 to create a short summary for each webpage and a general summary.

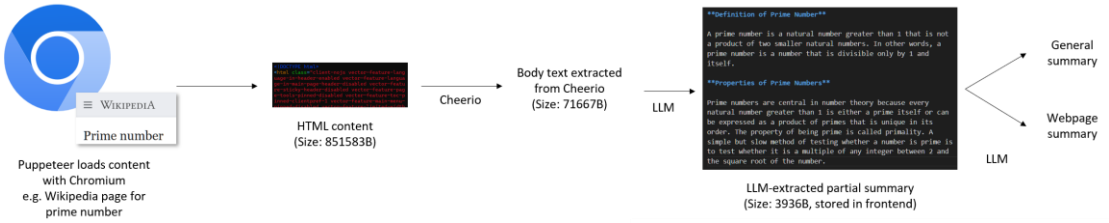

Fig. 3.5 Process of handling a single search result URL.

Finally, the backend sends the general summary, along with the list of URLs and their corresponding webpage summaries, back to the frontend as the response to the request. The frontend then presents them to the user.

3.4. Evaluations

No formal evaluations or user testing have been conducted at this time. They will be performed near the end of the project's implementation. This will ensure that the evaluation is meaningful and reflective of the system's performance in delivering relevant and concise information to users.

3.5. Difficulties Encountered

3.5.1. Use of LLM Service Provider

Azure was initially chosen as the LLM service provider due to its access to OpenAI's models, particularly since OpenAI's APIs are unavailable in Hong Kong. However, during backend integration, it was discovered that Azure's default rate limit was exceeded after processing the top three results twice within 10 seconds, which is insufficient for our scalability requirements. Additionally, Azure only provides access to older models like GPT-3.5 and GPT-4 by default, and upgrading to advanced models like GPT-4 or increasing rate limits requires additional approval from Microsoft.

To address these limitations, alternative LLM service providers such as Hyperbolic (with a rate limit of 600 requests per minute) were tested. Currently, the project uses DeepSeek as the LLM service provider due to its lack of rate limits. However, Hyperbolic offers smaller models like Llama-3.2-3B for faster processing, taking less than 5 seconds to generate partial, general and webpage summaries for 3 webpages with the search query "diameter of earth". On the other hand, DeepSeek primarily provides larger models such as DeepSeek-V3, which takes 19 seconds to process the same 3 webpages. As development progresses, other LLMs and providers will continue to be evaluated to balance performance and efficiency.

3.5.2. Responsiveness of the Application

Parsing and loading webpages with Puppeteer, combined with generating summaries using LLMs, can be time-consuming, with processing times reaching up to 30 seconds for all search results. The processing times can be further increased by slow responses from the parsed webpages. Currently, the backend handles user queries in a single HTTP request, forcing users to wait until all summaries are generated before seeing any results. This creates the impression of a slow or unresponsive service. To enhance the user experience, the single HTTP request will be split into multiple requests, allowing webpage summaries to be displayed as they are generated. Additionally, different methods to optimize the webpage parsing speed will be explored, such as attempting a simple HTTP request without Puppeteer first and only resorting to Puppeteer if there is JavaScript content that needs to be dynamically loaded.

4. Project Schedule

Table 4.1 presents the proposed future project schedule. Additional use cases listed in Section 2.1, such as allowing additional user queries and recursive searches, will be implemented. Near the end of the project, user testing and evaluations of the application's effectiveness will be performed.

Month	Schedule
2025-01	Deploying backend and frontend, UI refinements
2025-02	Implementing user query system, recursive searches
2025-03	Implementing recursive searches, user account system
2025-04	User testing and evaluations

Table 4.1 Proposed project schedule.

5. Conclusion

This report presented the development of a cross-platform web search application designed to combine the strengths of traditional search engines and AI-powered chatbots while addressing their limitations. By retrieving search results through APIs, reordering them for relevance, and generating concise summaries, the project aims to improve the efficiency and reliability of the search experience. A functional prototype has been developed, which fetched results from the Bing Search API and leveraged DeepSeek-V3 to generate summaries for search results, demonstrating the potential for better clarity and relevance compared to traditional search engines. This advancement could be a step forward in enhancing the user experience in online

information retrieval by providing accurate information in a streamlined and concise format.

Despite these promising results, the project encountered several challenges. First, limitations with the initial LLM service provider, Azure, such as restrictive rate limits and access to older models, necessitated a switch to DeepSeek, which lacks rate limits but relies on larger models that increase processing times. Additionally, the responsiveness of the application was impacted by the time-consuming nature of parsing webpages with Puppeteer and generating summaries with LLMs, with processing times reaching up to 30 seconds for all search results.

Continuous evaluation of different LLMs and providers will be essential for identifying potential improvements in the application's responsiveness and efficiency. Furthermore, planned features, such as enabling follow-up user queries and recursive searches, are yet to be implemented. These efforts are expected to enhance the overall effectiveness and user experience of the application, paving the way for a more intuitive and efficient search tool.

6. References

[1] "Poe." Accessed: Oct. 01, 2024. [Online]. Available: https://poe.com/

[2] "Perplexity," Perplexity AI. Accessed: Oct. 01, 2024. [Online]. Available: https://www.perplexity.ai/search/new?q=pending&newFrontendContextUUID=8200c74c-1369-4d77-b6af-36bfac36eab1&_rsc=1csqs

[3] "Cloud Market Growth Stays Strong in Q2 While Amazon, Google and Oracle Nudge Higher | Synergy Research Group." Accessed: Oct. 16, 2024. [Online]. Available: https://www.srgresearch.com/articles/cloud-market-growth-stays-strong-in-q2-while-amazon-google-and-oracle-nudge-higher

[4] "Gartner Predicts Search Engine Volume Will Drop 25% by 2026, Due to AI Chatbots and Other Virtual Agents," Gartner. Accessed: Oct. 01, 2024. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2024-02-19-gartner-predicts-search-engine-volume-will-drop-25-percent-by-2026-due-to-ai-chatbots-and-other-virtual-agents

[5] W. X. Zhao *et al.*, "A Survey of Large Language Models," Oct. 13, 2024, *arXiv*: arXiv:2303.18223. Accessed: Oct. 16, 2024. [Online]. Available: http://arxiv.org/abs/2303.18223

[6] "Introducing GPT-4o mini in the API - Announcements," OpenAI Developer Forum. Accessed: Oct. 01, 2024. [Online]. Available: https://community.openai.com/t/introducing-gpt-4o-mini-in-the-api/871594

[7] J.-Y. Yao, K.-P. Ning, Z.-H. Liu, M.-N. Ning, Y.-Y. Liu, and L. Yuan, "LLM Lies: Hallucinations are not Bugs, but Features as Adversarial Examples," Aug. 04, 2024, *arXiv*: arXiv:2310.01469. Accessed: Oct. 01, 2024. [Online]. Available: http://arxiv.org/abs/2310.01469

[8] J. Bevendorff, M. Wiegmann, M. Potthast, and B. Stein, "Is Google Getting Worse? A Longitudinal Investigation of SEO Spam in Search Engines," in *Advances in Information Retrieval*, vol. 14610, N. Goharian, N. Tonellotto, Y. He, A. Lipani, G. McDonald, C. Macdonald, and I. Ounis, Eds., in Lecture Notes in Computer Science, vol. 14610. , Cham: Springer Nature Switzerland, 2024, pp. 56–71. doi: 10.1007/978-3-031-56063-7_4.

[9] "Google Search Status Dashboard." Accessed: Oct. 01, 2024. [Online]. Available: https://status.search.google.com/products/rGHU1u87FJnkP6W2GwMi/history

[10] "Mobile-first indexing | Google Search Central Blog," Google for Developers. Accessed: Sep. 30, 2024. [Online]. Available: https://developers.google.com/search/blog/2016/11/mobile-first-indexing

[11] "Announcing mobile first indexing for the whole web | Google Search Central Blog," Google for Developers. Accessed: Sep. 30, 2024. [Online]. Available: https://developers.google.com/search/blog/2020/03/announcing-mobile-first-indexing-for

[12] F. Almeida and J. Monteiro, "The role of responsive design in web development," *Webology*, vol. 14, pp. 48–65, Dec. 2017.

[13] "Serverless Function, FaaS Serverless - AWS Lambda - AWS," Amazon Web Services, Inc. Accessed: Sep. 30, 2024. [Online]. Available: https://aws.amazon.com/lambda/

[14] C. Shen, L. Cheng, X.-P. Nguyen, Y. You, and L. Bing, "Large Language Models are Not Yet Human-Level Evaluators for Abstractive Summarization," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Feb. 2023, pp. 4215–4233. doi: 10.18653/v1/2023.findings-emnlp.278.