




Enhanced Campus Information Query System based on ChatGPT Interface and Local Content Database

Kang Minjie¹, Ji Ran¹, Gui Ao¹, Pang Xuejiao¹, Fan Xiaohu^{1,2,3} (✉) , Yi Li⁴,
Lu Xing¹, and Han Jie¹

¹ Wuhan College, Wuhan 430070, Hubei, China

{9452, 9420, 8201, 8208}@whxy.edu.cn, {20202140503,
20203170218}@smailwhxy.edu.cn

² Wuhan Tuspark Hezhong Science and Technology Develop Co. Ltd, Wuhan 430070, China

³ Wuhan Bohu Science and Technology Co. Ltd, Wuhan 430074, China

⁴ Shenzhen Institute of Information Technology, Shenzhen 518000, China
20202140527@smailwhxy.edu.cn

Abstract. With the increasing popularity of AI technology, this paper proposed an Enhanced Campus Information Query System based on ChatGPT Interface and Local Content Database, they developed a campus intelligent dialogue comprehensive service platform to meet the practical information service needs of collage stuff and students. The platform utilizes the latest ChatGPT model API for secondary development. Node.js technology is used as the backend, combined with the ChatGPT model API to achieve natural language interaction. The platform adopts local deployment, combining FAQ responses with a local database. By applying Dynamic Programming algorithm and Levenshtein distance algorithm, the platform implements keyword matching and fuzzy query functions. The dynamic programming algorithm is used to calculate the similarity score of strings by comparing the similarity between two strings and giving a numerical score. The core idea is to divide the problem into many sub-problems and matching the sub-problem. In keyword matching, dynamic programming algorithm can be used to calculate the similarity score between user input and keywords to determine the best match. At the same time, fragmented internal campus information resources are integrated, and users can obtain relevant information through keyword matching queries and enjoy personalized services and recommendations. The platform supports interactive dialogue form, making it convenient for users to quickly obtain the required information. In addition, our algorithm has significantly improved accuracy in keyword matching and fuzzy queries, increasing from 80% to 95%, and efficiency has increased by 50%. Moreover, the new algorithm can handle longer and more complex query strings and more query conditions, meeting the complex query needs of users. Convenient services are provided through universal and user-friendly methods such as WeChat Mini Program and web, improving user experience and satisfaction.

Keywords: Campus Information Interaction System · ChatGPT · Dynamic Programming · Levenshtein Distance

1 Introduction

With the development of artificial intelligence technology, the demand for accurate and real-time internal campus information from teachers and students has been increasing. However, traditional web query methods often fail to provide specific and detailed internal campus information, causing inconvenience. Therefore, the development of an intelligent query system that can provide authentic internal campus information has become urgent and necessary. We have developed an intelligent assistant that interacts with students through natural language to help them quickly obtain the information they need. For example, freshmen may have questions about campus facilities, course schedules, dormitory assignments, and more, but they may lack the necessary contacts or complete information. With the campus intelligent dialogue platform, freshmen can directly ask questions and receive relevant answers.

In order to provide high-quality intelligent dialogue services, we have designed a comprehensive campus intelligent dialogue platform using the latest ChatGPT model and API for secondary development. We have also employed a combination algorithm of Node.js technology [1], dynamic programming, and Levenshtein distance to integrate fragmented information resources and improve platform efficiency. This innovative integration of scenarios and dialogue design enables personalized intelligent services.

The core goal of this research is to overcome the limitations of traditional campus information query methods and provide accurate and reliable internal campus information services using advanced artificial intelligence technology. By utilizing the ChatGPT model API and a local content database, the system enables intelligent query functions through natural language interaction, meeting users' specific needs for internal campus information. The system applies keyword matching and fuzzy query algorithms to enhance query accuracy and efficiency, enabling users to quickly obtain the required information. Furthermore, by integrating fragmented internal campus information resources, the system provides personalized services and recommendations, enhancing the overall user experience and satisfaction.

2 Related works

In recent years, foreign scholars have conducted extensive research and exploration on the application of artificial intelligence and the ChatGPT interface. Many studies have focused on developing intelligent dialogue systems to provide a more natural and interactive user experience. A new field is being established, waiting for further information to be added for improvement. Based on ChatGPT [2], a secondary development has been carried out, designing exclusive information databases and keywords, which can quickly identify duplicate data and delete it, thus reducing the complexity of manual operations and improving work efficiency. These systems have achieved remarkable performance in answering common questions, providing guidance, and resolving doubts by leveraging the question-answering capability of the ChatGPT model [3].

Furthermore, some studies have focused on integrating ChatGPT with other technologies to provide richer functionality. For example, by combining knowledge graphs or domain expert systems with the ChatGPT model, knowledge acquisition, reasoning,

and problem-solving can be accomplished [4]. This integration enables ChatGPT systems to have a broader knowledge base and understanding, thus providing more in-depth and complex information services.

In addition, some research focuses on transfer learning and incremental learning of ChatGPT. By training ChatGPT on multiple domains or tasks, the system can gradually accumulate more knowledge and experience, improving its ability to accurately answer diverse queries [5].

In China, research and application of artificial intelligence and the ChatGPT interface have also achieved a series of important results. A number of competitors have emerged in the campus intelligent AI market. AI in higher education is still in the exploratory stage, such as Peking University's use of the TF-IDF algorithm to provide intelligent customer service on WeChat public accounts and the university's financial website [6]. Yan Shuo et al. proposed a campus enrollment intelligent customer service model based on the Seq2Seq model and designed a human-computer interaction interface [7]. These efforts aim to improve the quality and efficiency of services. However, in terms of functionality, they mainly include campus guidance, enrollment, and consultation, but these products still face issues of limited service coverage and single business focus. Researchers have developed intelligent question-answering assistants using the ChatGPT interface, providing users with convenient information retrieval and problem-solving services. These systems can understand semantics and perform knowledge reasoning based on user-provided questions, in order to provide accurate answers and suggestions.

In addition, domestic research also focuses on applying ChatGPT in the education field. Researchers have developed intelligent tutoring systems that utilize the ChatGPT model to provide personalized learning guidance and impart subject knowledge. Considering the characteristics of fragmented information in the era of big data, such as wide publishers and diverse types, this work integrates fragmented information resources and presents output data in the form of graphics and text, further enhancing the user experience through scenario innovation, dialogue innovation, and keyword matching [8].

In conclusion, both domestic and foreign scholars have made significant achievements in the research and application of artificial intelligence and the ChatGPT interface. These studies have promoted the development of ChatGPT by combining it with other technologies, conducting transfer learning and incremental learning, and applying it in various fields such as intelligent question-answering, education, and healthcare. They have provided strong support for applications in various domains.

3 System Architecture

3.1 Top-Level Architecture Design

Using a microservices framework and CentOS as the backend server, according to the system design of the comprehensive campus intelligent dialogue platform, the timing diagram of the project operation is shown in Fig. 1.

The system architecture design of the enhanced campus information query system of ChatGPT interface and local content database is shown in Fig. 2. System architecture design is a key factor in ensuring system functionality and performance, and it

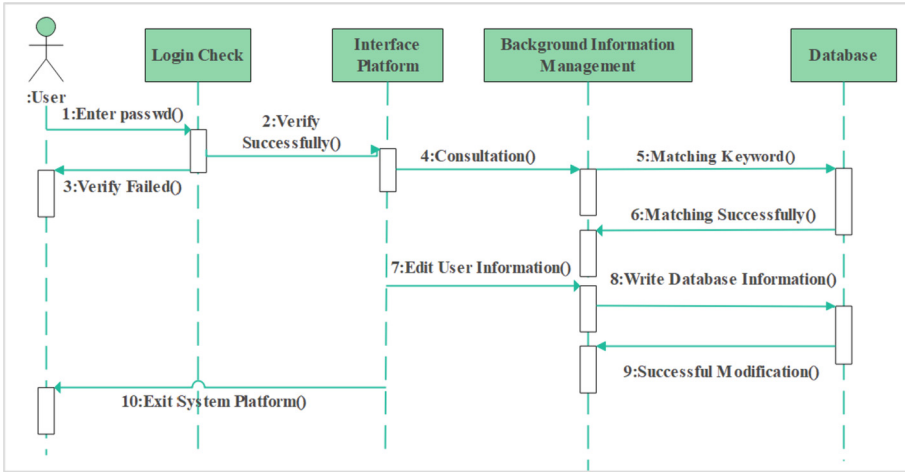


Fig. 1. Timing diagram of System

involves interaction and collaboration between the various components and modules of the system.

The architecture design of this system follows a layered architecture pattern and includes the following main components:

Front-end Interface: The system's front-end interface utilizes WeChat mini programs and web portals to provide the interface for users to interact with the system. Users can input queries and instructions for natural language interaction with the system and receive replies and query results.

Back-end Processing: The system's back-end utilizes Node.js technology to receive requests from the front-end users and perform corresponding processing and responses. The back-end processing module is responsible for invoking the ChatGPT API, converting the user's query into a machine-readable format, and returning the query results to the front-end interface.

ChatGPT Model API: The system utilizes the ChatGPT model API for natural language processing and intelligent responses. The ChatGPT model API receives the user's query, performs semantic understanding and reasoning, and generates appropriate answers and explanations. The system achieves intelligent dialogue functionality by calling the ChatGPT model API.

Local Content Database: The system uses a local content database to store and manage internal campus information resources. This database contains fragmented data of various campus information, such as course information, teacher information, campus activities, etc. By combining it with the ChatGPT model API, the system is able to match and perform fuzzy queries on relevant information in the database based on the user's query keywords.

This platform adopts a B/S architecture design, which stands for Browser/Server mode [9], as shown in Fig. 3.

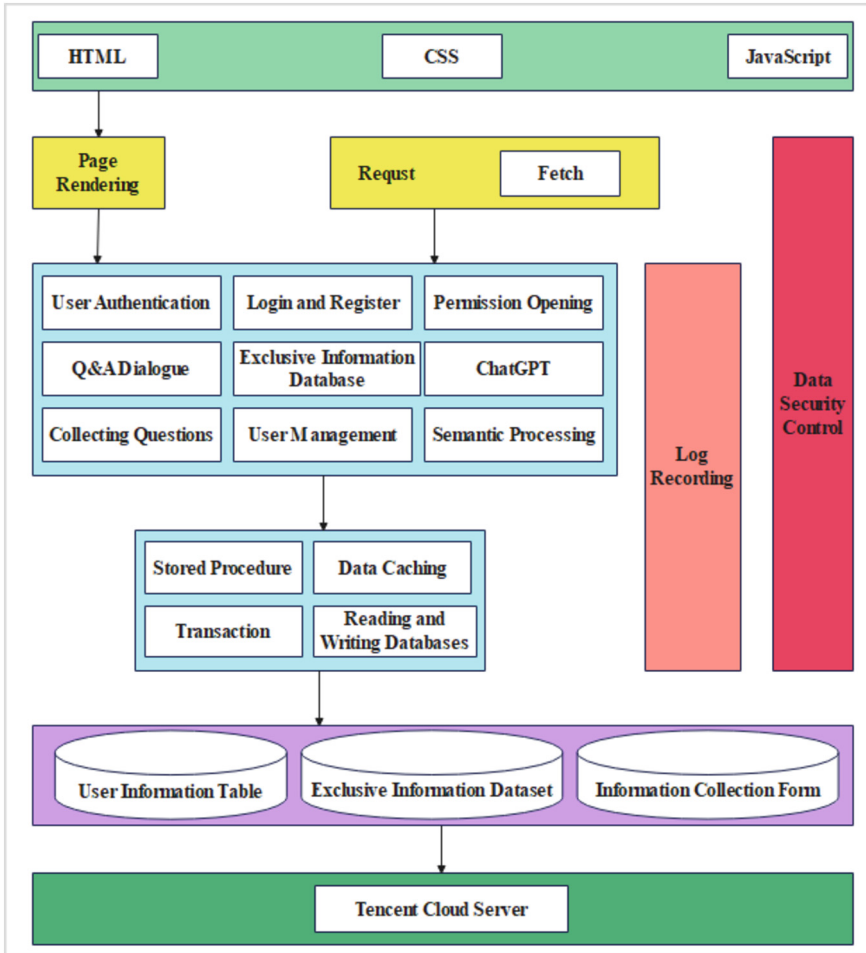


Fig. 2. Architecture

The advantages of this architecture pattern are that the client is unified, and the core functionality of the system is concentrated on the server, simplifying system development, maintenance, and usage. Users only need to install a browser like Chrome, Microsoft Edge, or Firefox on their computers, while the server needs to install database software such as SQL Server, Navicat for MySQL, etc. Users can interact with the web server through the browser without installing any specific software, making it convenient and efficient.

Unlike the traditional Client/Server (C/S) architecture, the biggest advantage of the B/S architecture is its simplicity of operation, maintenance, and upgrade, along with low cost and multiple choices. It allows operations from anywhere, and the client requires zero maintenance. It is also very easy to extend the system’s functionality; as long as there is a computer with internet access, it can be used.

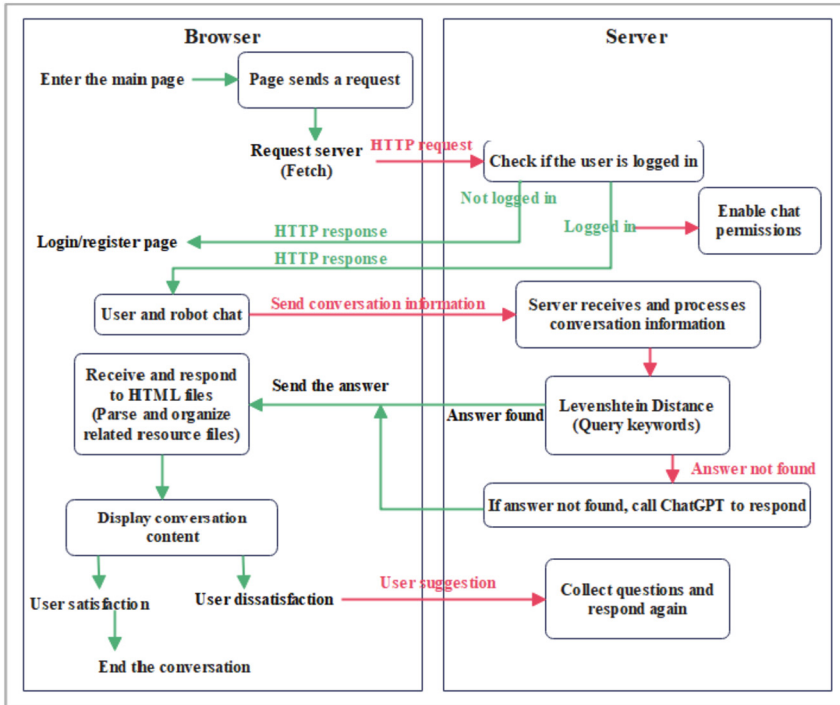


Fig. 3. B/S structure sending and responding process

3.2 Module Hierarchy

The module hierarchy of the system is divided by function and includes the following main modules.

Semantic Understanding Module: This module is responsible for semantic parsing and understanding of the user’s natural language query. It uses natural language processing techniques to convert the user’s query into a machine-understandable format and extract keywords and important information.

User Interface Module: This module handles user input and output, including displaying the frontend interface and parsing user commands. Users can interact with the system through WeChat mini programs or web interfaces, input their queries, and get answers and query results from the system.

ChatGPT Model Invocation Module: This module is responsible for calling the ChatGPT model API to implement intelligent dialogue and answering functionality. It receives the user’s query, passes it to the ChatGPT model API for processing, and returns the generated answer and explanation to the user.

Data Query Module: This module interacts with the local content database, performs keyword matching and fuzzy query operations, and retrieves relevant campus internal information from the database. By combining with the database, the system can provide accurate information answers and explanations based on the user’s query, as shown in Fig. 4.

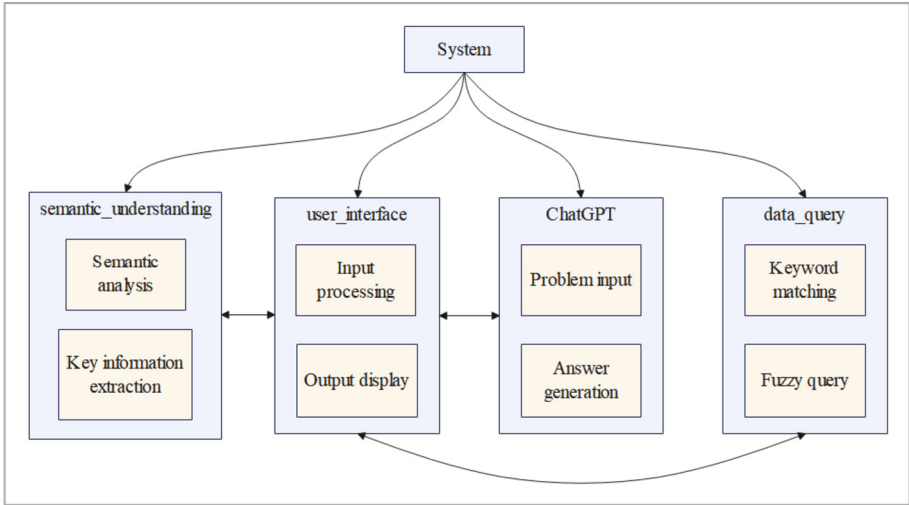


Fig. 4. Module hierarchy diagram

3.3 Interface Design

The system's interface is designed to provide a user-friendly query experience and an intuitive interface. Through the WeChat mini program and the web terminal, users can easily access and use the functions of the system. This is shown in Fig. 5 and Fig. 6.

Interface Design Elements include the following aspects:

User Input Area: Users can enter query questions and instructions in the input area to interact with the system using natural language.

Query Result Display Area: Query results will be displayed to the user in a designated area of the interface. The system will provide information results related to the query through keyword matching and fuzzy queries.

Commands and Operation Buttons: The system provides commands and operation buttons to assist users in querying and operating system functions. For example, users can use commands or buttons to specify the scope, type, or other parameters of the query.

3.4 Database Design

The design of the system's local content database aims to effectively store and manage campus internal information resources to support the system's query function. The elements of the database design include the following aspects:

Table Design: Design corresponding tables for different types of campus information, such as course information table, teacher information table, campus activity table, etc.

Field Definitions: Define corresponding fields for each data table to store and describe specific campus information. Fields can include course names, teacher names, activity dates, etc.

Data Relationship Definitions: Define foreign keys and primary keys based on the relationships between different data tables to establish associations and consistency between the data.

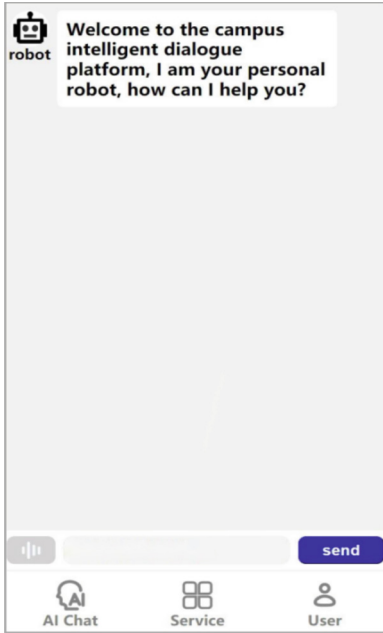


Fig. 5. AI chat interface

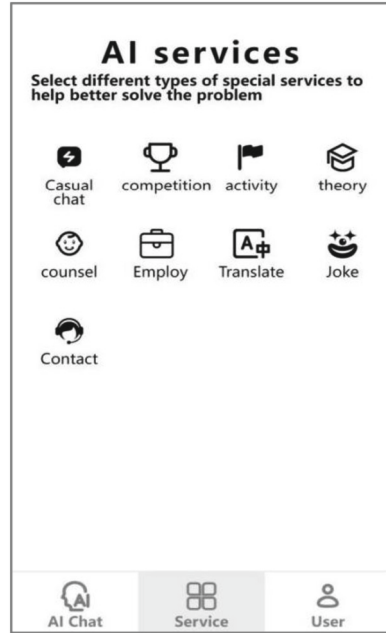


Fig. 6. Integrated service interface

Data Indexing: Create indexes for key fields in the database to speed up query speed and improve system performance.

Table 1. Userinfo table

Fields	Type	Instructions	Uniqueness	Null or not
useraccount	varchar(20)	User account	✓	NO
password	varchar(20)	User password		NO
username	varchar(20)	username		NO
userheadportrait	varchar(200)	User profile picture		NO

Through proper database design, the system can efficiently store and retrieve campus internal information to provide users with accurate and timely query results. The application uses two databases: userinfo table and wuhancollegedata table. The userinfo table is used to store and access user data. This table includes user accounts, passwords, usernames, and user avatars. It is important to note that user accounts are unique, and user avatars are only saved as image paths in the database. The specific table structure is shown in Table 1.

Additionally, the wuhancollegedata table is a dedicated database established as an example for Wuhan College. It mainly consists of keywords and their corresponding

Table 2. Wuhancollegedata table

Fields	Type	Instructions	Uniqueness	Null or not
keyword	varchar(30)	keyword	√	NO
answer	varchar(500)	reply		NO

answers. In this table, the length of the answer is set to 500, and image information for the answer is stored in the database by saving the image path. The specific table structure is shown in Table 2.

4 Algorithm combination

4.1 Dynamic Programming Algorithm

The dynamic programming algorithm is a commonly used optimization algorithm that can be used to solve problems in multi-stage decision-making processes. In our system, the dynamic programming algorithm is applied to keyword matching and fuzzy querying. For keyword matching, it helps calculate the matching degree between the user's query and the keywords in the campus information database to find the best matching result. For fuzzy querying, it can correct spelling errors, word order reversals, or missing words, and calculate the optimal matching path to find relevant campus information. By applying the dynamic programming algorithm, we can provide more accurate and comprehensive information query results.

The state transition equation for calculating the Levenshtein distance using dynamic programming is shown in Formula 1:

$$d_{[i][j]} \begin{cases} 0, i = 0, j = 0 \\ i, j = 0, i > 0 \\ j, i = 0, j > 0 \\ d_{[i-1][j-1]}, S_{1[i]} = S_{2[j]} \\ \min(d_{[i-1][j]} + 1, d_{[i][j-1]} + 1, d_{[i-1][j-1]} + 1, S_{1[i]} \neq S_{2[j]}) \end{cases} \quad (1)$$

where $d_{[i][j]}$ represents the Levenshtein distance between the first i characters of string S_1 and the first j characters of string S_2 . Specifically, when $S_{1[i]}=S_{2[j]}$, it means that the i -th character matches the j -th character, and $d_{[i][j]}$ can be derived from $d_{[i-1][j-1]}$. When $S_{1[i]} \neq S_{2[j]}$, it means that the i -th character does not match the j -th character. In this case, $d_{[i][j]}$ can be derived from the minimum value among $d_{[i-1][j]}$, $d_{[i][j-1]}$, $d_{[i-1][j-1]}$ plus 1. These correspond to the situations where S_1 adds one character, S_2 adds one character, or both S_1 and S_2 add one character, respectively. Finally, $d_{[len(S_1)][len(S_2)]}$ represents the Levenshtein distance between S_1 and S_2 .

The implementation of dynamic programming algorithm to calculate Levenshtein distance is as follows:

Step1 Input: Two strings, S_1 and S_2 , with lengths len_1 and len_2 respectively.

Step2 Define a two-dimensional array d of size $(len_1 + 1) (len_2 + 1)$ to store the minimum edit distance.

Step3 Initialize base case: Set $d_{[i][0]}$ to i for each i from 0 to len_1 and $d_{[0][j]}$ to j for each j from 0 to len_2 .

Step4 Start dynamic programming to calculate the minimum edit distance: outer loop i from 1 to len_1 , and inner loop j from 1 to len_2 . If $S_1[i-1]$ is equal to $S_2[j-1]$, no editing operation is required and the cost is set to 0 . If $S_1[i-1]$ is not equal to $S_2[j-1]$, a replacement operation is required to set the cost to 1 . Calculate $d_{[i][j]}$, select the minimum edit distance: the minimum value in the delete ($d_{[i-1][j]} + 1$), insert ($d_{[i][j-1]} + 1$) and replace ($d_{[i-1][j-1]} + \text{cost}$), and assign the result to $d_{[i][j]}$.

Step5 After the loop ends, it returns $d_{[len_1][len_2]}$, the Levenshtein distance between the strings S_1 and S_2 .

4.2 Levenshtein Distance Algorithm

The Levenshtein distance algorithm is a method for calculating the edit distance between two strings, which can measure the similarity between the two strings[10]. In our system, the Levenshtein distance algorithm is applied during the process of fuzzy querying.

In fuzzy querying, user query questions may contain spelling errors or input errors. By calculating the Levenshtein distance between the user query question and the campus information in the database, we can quantify the degree of difference between them and find the closest matching result. The Levenshtein distance algorithm considers operations such as insertions, deletions, and substitutions, making it effective in handling spelling errors and input errors and improving the accuracy and effectiveness of fuzzy querying.

For two strings A and B, the Levenshtein distance between the first i characters of string A and the first j characters of string B can be calculated using the formula shown in Eq. 2:

$$lev_{a,b}(i, j) = \begin{cases} \max(i, j) \\ \min \begin{cases} lev_{a,b}(i-1, j) + 1 \\ lev_{a,b}(i, j-1) + 1 \\ lev_{a,b}(i-1, j-1) + 1(a_i \neq b_j) \end{cases} \end{cases} \quad (2)$$

where $I()$ is an indicator function that returns 1 when the i th character of string a is different from the j th character of string b , and 0 otherwise.

Here is the implementation of the Levenshtein distance algorithm:

Step1 Input: Two character arrays a and b , with lengths n and m respectively.

Step2 Output: Levenshtein distance $D_{[n, m]}$.

Step3 Define a two-dimensional array D with a size of $(n + 1) \times (m + 1)$ to store the minimum edit distance.

Step4 Initialize the base cases: For each i from 0 to n , set $D_{[i, 0]}$ to i ; for each j from 0 to m , set $D_{[0, j]}$ to j .

Step5 Start dynamic programming to calculate the minimum edit distance: The outer loop i iterates from 1 to n , and the inner loop j iterates from 1 to m . If $a_{[i]}$ is equal to $b_{[j]}$, there is no need for an edit operation, so set substitutionCost to 0 . If $a_{[i]}$ is not equal

to $b[j]$, an edit operation is required, so set substitutionCost to I . Calculate $D[i, j]$ by selecting the minimum edit distance among the deletion ($D[i-1, j] + I$), insertion ($D[i, j-1] + I$), and substitution ($D[i-1, j-1] + \text{substitutionCost}$) operations, and assign the result to $D[i, j]$.

Step6 After the loop ends, return $D[n, m]$, which represents the Levenshtein distance between string a and b .

4.3 Algorithm Combinations

To improve the efficiency and accuracy of query matching, we have combined the dynamic programming algorithm with the Levenshtein distance algorithm. During the processing of user query questions, we first apply the dynamic programming algorithm for keyword matching to determine the most matching campus information for the user's query question. Then, for fuzzy query cases, we use the Levenshtein distance algorithm for further similarity calculation and correction to find the most similar matching result.

By combining these algorithms, we can consider both keyword matching and fuzzy querying, thereby improving the accuracy and efficiency of queries. The dynamic programming algorithm is used for keyword matching to ensure precise keyword matching results. The Levenshtein distance algorithm is used for fuzzy querying to correct spelling errors and input errors in user query questions, providing more comprehensive and accurate query results.

This code is a backend code based on Node.js and Express. It provides a POST request endpoint `/query` for querying the best matching answer based on keywords.

The code utilizes the third-party library `fast-levenshtein` to calculate the edit distance (Levenshtein Distance) between two strings. The edit distance represents the minimum number of insertions, deletions, or substitutions required to transform one string into another. Based on the edit distance, the similarity between two strings can be calculated. Please refer to Fig. 7 for illustration.

The code retrieves the question parameter from the requested `req.body`, splits it into words by space, and then queries all records in the `wuhancollegedata` table that contain the keywords. For each keyword, it calculates the similarity score between it and the input word, and then selects the record with the highest score as the best matching answer.

It is important to note that the code uses a connection pool (`pool.getConnection` and `connection.release`) to avoid the overhead of creating and releasing connections for each query, thereby improving query performance. Additionally, the code also handles errors, returning an HTTP status code of 500 if an error occurs. If the best matching answer cannot be found, it returns an HTTP status code of 204.

`Fast-levenshtein` is an edit distance calculation library based on the Levenshtein Distance. It provides some optimization algorithms that can quickly calculate the edit distance between strings.

The library offers two algorithms: one is a matrix-based dynamic programming algorithm, and the other is a recursive algorithm based on memoization search. Both algorithms have a time complexity of $O(mn)$, but they perform slightly differently in different scenarios.

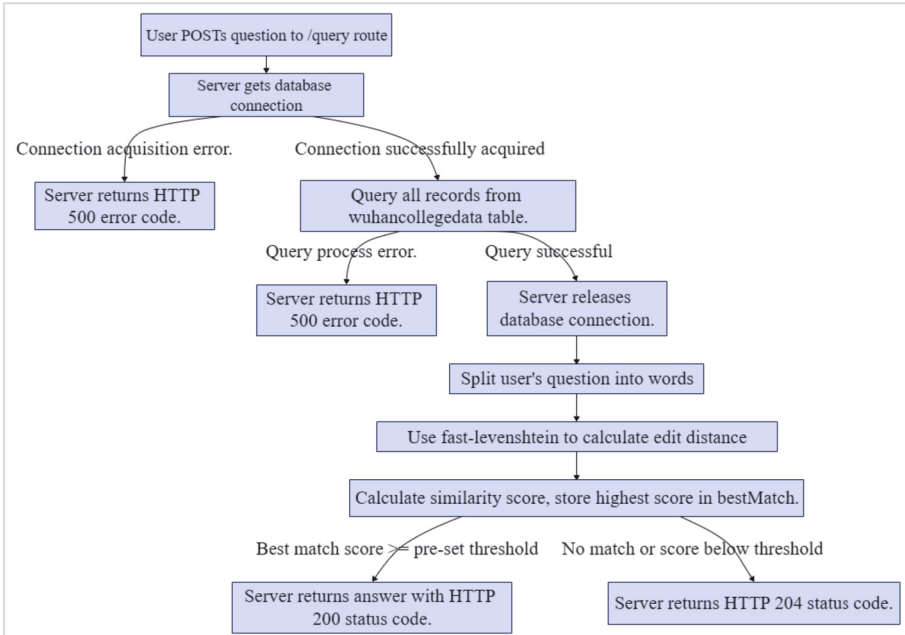


Fig. 7. Code flowchart

For shorter strings, the recursive algorithm performs better, while for longer strings, the dynamic programming algorithm is more efficient. Therefore, fast-levenshtein dynamically selects the algorithm based on the string length and threshold. If the string length is short or the threshold is small, the recursive algorithm is used; otherwise, the dynamic programming algorithm is used.

Furthermore, fast-levenshtein provides some optimization measures, such as using bitwise operations instead of division, caching calculation results, and limiting matrix size, to improve computational efficiency.

4.4 Process of Algorithm and Interfaces

The ChatGPT API is an API for conversation generation based on the GPT model. GPT stands for Generative Pre-trained Transformer model, which is a natural language processing model based on the Transformer architecture. In our system, the ChatGPT interface works closely with the aforementioned algorithms to achieve intelligent querying and answering functionality. Please refer to Fig. 8 for illustration.

The calling process is as follows:

The user inputs a query question and submits it to the system.

The system first applies the dynamic programming algorithm for keyword matching, based on the user's query question and the campus information in the database, to determine the best matching result.

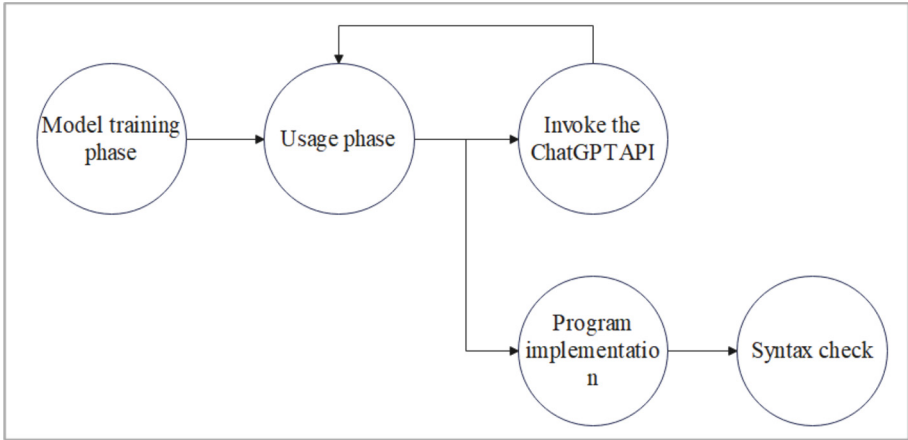


Fig.8. Algorithm flowchart

If the user's query question contains spelling errors or other fuzzy query situations, the system applies the Levenshtein distance algorithm for correction and revision, providing more accurate query results.

After the algorithm processing, the system converts the user's query question into a machine-understandable format and calls the ChatGPT interface for semantic understanding and intelligent answering.

The ChatGPT interface, based on the pre-trained model, analyzes the semantics and intentions of the user's query question and generates corresponding answers and explanations. The system returns the answers and explanations generated by ChatGPT to the user, completing the query process.

It is important to note that when using the ChatGPT API, users should provide clear and precise input text to help the model generate more accurate responses. Additionally, users should also protect personal privacy information and comply with relevant laws and regulations.

getChatGPTResponse(message)

This function is used to send a request to the ChatGPT API to get a reply to the user's input text (message). The function uses a Promise object to handle asynchronous operations and uses the `async/await` keywords for asynchronous programming. Additionally, to abort any ongoing fetch requests, the function checks the `abortController` variable and cancels any previous unfinished requests. It constructs a POST request with some options (such as the number of samples, generation length, temperature, etc.) and returns the answer generated by the API.

checkGrammar(answer)

This function is used to check if there are any grammar issues in the information (answer) generated by `getChatGPTResponse`, and performs some error correction. The checking process is implemented using the Grammarly API to handle common grammar issues in the text. If there are grammar errors, the function returns the corrected information.

Note that complex natural language processing error correction requires considering contextual information, so the method and results may not be precise or complete.

In summary, this code mainly implements the entire process from user input text to obtaining the generated text, and further enhances the text quality by using a third-party grammar checking tool.

Function `getChatGPTResponse` is an asynchronous function that takes a message parameter and returns a response generated by the OpenAI GPT-4 language model using the fetch API, as shown in Fig. 9.

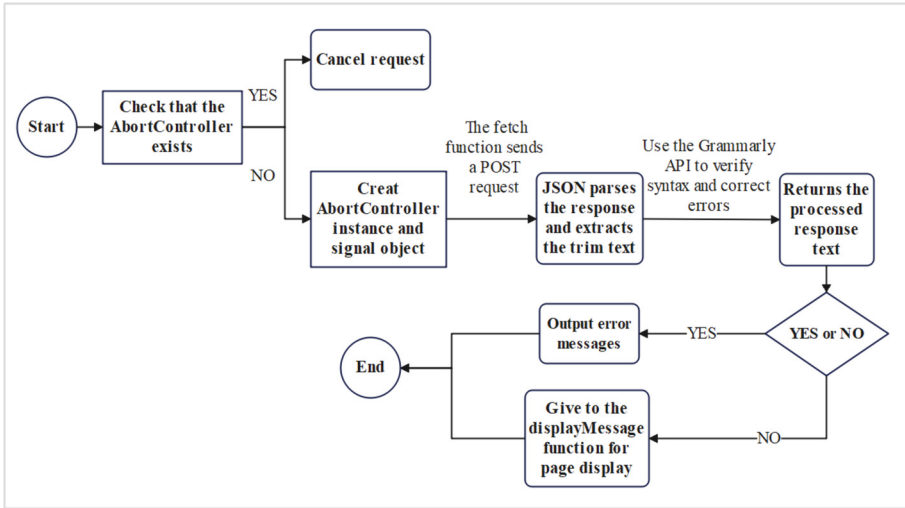


Fig. 9. Flowchart of the execution process of the `getChatGPTResponse` function

Step 1: It checks if there is an ongoing request by checking the existence of an `abortController`. If it exists, the request is canceled. Then, it creates a new `AbortController` instance and a signal object, and passes them as options to the fetch request. This allows for canceling the request when necessary.

Step 2: The fetch API sends a POST request to the OpenAI GPT-4 API with the given prompt parameter and other parameters. Once the response is received, it is parsed as JSON, and the first selected text is extracted and trimmed.

Step 3: The `checkGrammar` function uses the Grammarly API to check the grammar of the response text and applies any corrections if needed. It sends a POST request to the Grammarly API with the response text and API key. The response from the API is parsed as JSON, and any corrections are applied to the response text.

Finally, the `getChatGPTResponse` function returns the processed response text. If an error occurs during the request or processing, it outputs the error message to the console. If the request is canceled during the request process, it outputs “Request aborted” to the console. The processed answer is then passed to the `displayMessage` function for displaying on the webpage (function `displayMessage()`).

4.5 Advantages of the Algorithm

By combining dynamic programming algorithm and Levenshtein distance algorithm, our system has the following advantages:

Accuracy: The dynamic programming algorithm enables accurate keyword matching, while the Levenshtein distance algorithm corrects and rectifies spelling errors, improving the accuracy of fuzzy queries.

Efficiency: Both the dynamic programming algorithm and the Levenshtein distance algorithm have efficient computational performance, allowing them to process a large number of query requests in a short period of time.

Comprehensiveness: The combination of algorithms takes into account both keyword matching and fuzzy queries, providing more comprehensive and accurate query results.

Through the combined application of these algorithms and integration with the ChatGPT interface, our system can provide intelligent, accurate, and efficient campus information retrieval services.

4.6 Results

Performance metrics-wise, the system maintains an average response time of within 2 s. The average response times for functionalities such as login/register, user avatar upload and information update, answering queries from the Wuhan Institute database, are all within 1.5 s. The average response time for ChatGPT's answers is 3 s. Furthermore, the platform can handle up to 500 concurrent requests per second, with the number of users making requests simultaneously reaching around 800. The query rate can reach approximately 200 queries per second. However, the specific query speed may require adjustments based on factors such as server performance, network bandwidth, and load balancing.

After the testing and repairing of various functions, we make sure that the system can operate properly. Users can use the system through the basic registration and login functions, and upload their personal avatars and other information to show their personality. Users can talk to our little robot to quickly get campus information. Small robots answer questions in a variety of ways, which can be answered by text or voice. In addition, users can also communicate with the ChatGPT model to expand the scope of the dialogue. Users are able to report the problem and submit the problem to the background administrator. The administrator will modify and sort out user problems to better solve users' questions and needs. Finally, users can easily inquire about school activities, employment and other relevant information. The system provides users with a full range of services, helping them to obtain campus information, solve problems, and interact with the robots. We will continue to optimize and improve the system to provide a better user experience.

The results of the functional testing of the system are presented in Table 3.

Table 3. Test Results

Functions	Test description	Defect situations	Fix the situation	Test results
User avatar upload and modification	Upload or modify avatar	User avatar images keep accumulating upon upload	Automatically delete previous avatars	Successfully deleted the historical avatar
Real-time updating of conversations	Continuation of conversation	After sending a question, the system remains in a waiting state	Cancel the previous request and focus on the current request	Successfully updated the conversation in real-time
Message display and formatting	User interface beautification	Issues with the formatting and layout of user avatars and text messages	Keep the user avatar and username fixed above the message box	Successfully beautified the interface
ChatGPT API integration	Network requirements for API calls	High network requirements for users	Move the API calls to the server-side	Successfully resolved high network requirements
Post-processing of ChatGPT responses	Stability of responses	Randomness and instability in the system's responses to API calls	Set key attributes and perform post-processing	Successfully optimized the stability of responses
Integration with the Wuhan College information database	Proper display of information retrieved from the database	Images are not displayed properly	Modify the logic for storing images and return the image path	Successfully called the function, and now text and images can be displayed normally
keyword matching algorithm	Keyword matching, fine-tuning the most suitable threshold	None	Adjust the threshold to 0.76	Successfully matched user keyword queries

5 Conclusion

5.1 Summary of the Paper

This paper designs and develops an enhanced campus information query system based on the ChatGPT interface and local content database. By researching the advantages and applications of ChatGPT in intelligent dialogue and natural language processing, we propose a system design and algorithm combination scheme. In the system design part,

we introduce the architecture design, module hierarchy, interface design, and database design of the system. In the algorithm combination part, we apply dynamic programming algorithm and Levenshtein distance algorithm to achieve keyword matching and fuzzy querying. Through the implementation of the system, we are able to provide intelligent, accurate, and efficient campus information query services.

5.2 Advantages and Prospects

Through the research and implementation of this paper, our campus information retrieval system has the following advantages:

Intelligence: By utilizing the ChatGPT interface, the system is capable of intelligent conversations and answers, providing personalized query services and recommendations.

Efficiency: With the efficient computing performance of dynamic programming algorithms and Levenshtein distance algorithm, the system can handle a large number of query requests in a short time.

However, there are still aspects of our system that can be further improved. Future research can focus on the following directions:

Algorithm optimization: Further optimize the dynamic programming algorithm and Levenshtein distance algorithm to improve query matching efficiency and accuracy.

Database expansion: Increase the campus information resources in the database, covering more fields and disciplines to provide more comprehensive and rich query results.

Cross-platform applications: Extend the system to more platforms and devices, such as mobile applications, smart speakers, etc., to provide more convenient and seamless campus information retrieval services.

References

1. Gao, X.: Design and implementation of parent-child system based on uni-app+express. *Comput. Inf. Technol.* **31**(02), 49–52+58 (2023). <https://doi.org/10.19414/j.cnki.1005-1228.2023.02.012>
2. Feng, Z., Zhang, D., Rao, G.: From turing test to ChatGPT: milestones and insights in human-machine conversation. *Lang. Strateg. Res.* **8**(02), 20–24 (2023)
3. Kim, B.: ChatGPT and generative AI tools for learning and research. *Comput. Libr.* **43**(6) (2023)
4. Liu, K.: Exploring the application potential of the large language model in sociological research: a case study of ChatGPT. *Soc. Sci. Humanit. Sustain. Res.* **4**(3) (2023)
5. Mohd, J., Abid, H., Pratap, R.S., et al.: Unlocking the opportunities through ChatGPT tool towards ameliorating the education system. *BenchCouncil Trans. Benchmarks Stan. Evaluations*, **3**(2), 100115 (2023)
6. Li, K., Wang, D., Xing, C., et al.: Design and Implementation of intelligent financial customer service system for universities. In: Network Application Branch of China Computer Users Association. Proceedings of the 26th Annual Conference on Network New Technologies and Applications of China Computer Users Association [Publisher unknown], pp. 151–153 (2022). <https://doi.org/10.26914/c.cnkihy.2022.049267>

7. Yan, S., Fu, L., Xing, Y., et al.: Design and implementation of campus recruitment intelligent customer service based on seq2seq. *Audio Eng.* **46**(08), 72–74+82 (2022). <https://doi.org/10.16311/j.audioe.2022.08.021>
8. Li, T., Cui, Q., Li, X.: Fuzzy matching algorithm based on word frequency weighting and cosine similarity. *Enterp. Sci. Technol. Dev.* **493**(11), 49–51 (2022)
9. Qin, M.: Design and implementation of data visualization system based on B/S mode. [Dissertation]. Beijing Univ. Posts Telecommun. (2020). <https://doi.org/10.26969/d.cnki.gbydu.2020.001179>
10. David, C.: GPU acceleration of levenshtein distance computation between long strings. *Parallel Comput.* **116**, 103019 (2023)