Application of Artificial Intelligence in Chatbots to Teach Computer Science Algorithms

CS4366: Senior Capstone Project

Team: AlgoWhiz

Authors: Dhruv Maniar and Isha Koregave

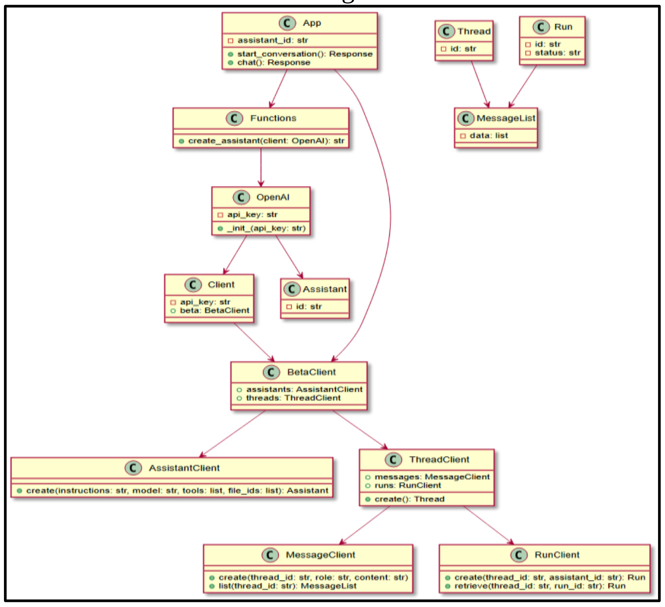
I. INTRODUCTION

AlgoWhiz is a cutting-edge learning tool created to offer thorough assistance in learning computer science algorithms. It works as an AI-powered chatbot that is embedded into our website and provides interactive features such as clarifications and advice on a range of algorithms that are frequently taught in computer science courses. Algorithms in computer science are fundamental to the field and provide the basis for comprehending computational procedures and approaches to problem-solving. But these ideas are frequently difficult to understand, particularly for students who are unfamiliar with the subject. AlgoWhiz seeks to solve this problem by providing personalized explanations and navigating the complexities of algorithms. Users may access a variety of algorithms that are frequently covered in computer science courses through AlgoWhiz, such as sorting techniques (such as bubble sort and merge sort), techniques for searching (like binary search), graphs (like Dijkstra's algorithm), and more. To aid with comprehension, each method is supported by thorough explanations, step-by-step instructions, and illustrated examples. AlgoWhiz's interactive features let consumers interact with the content in a way that suits them best. The chatbot allows users to ask inquiries, get explanations, visualize code, and get immediate answers. AlgoWhiz is available to help and support customers, regardless of whether they are having trouble grasping a certain algorithm or are looking for more practice tasks. Furthermore, AlgoWhiz is an invaluable tool for both teachers and students. AlgoWhiz is a tool that teachers may use to enhance their lesson plans and provide students more opportunities for reinforcement and learning. Conversely, students may enhance their comprehension of algorithms and be ready for tests and assignments by using AlgoWhiz as an extra study aid. AlgoWhiz wants to democratize access to computer science education by providing easily available and engaging instructional resources, enabling students to achieve academic success. AlgoWhiz aims to make learning computer science algorithms interesting, pleasurable, and ultimately rewarding for students of all levels with its user-friendly design and thorough material coverage. We were driven to start the AlgoWhiz project because we see a critical need in the field of education. Since we are computer science students, we have direct knowledge of the difficulties involved in deciphering complicated algorithms and the significance of these concepts in our curriculum. We were inspired to develop a solution that would ease these difficulties and improve learning for both us and other students as we saw how common this difficulty is among students everywhere. Furthermore, the significance of easily available and interactive learning resources has been highlighted by the growing popularity of digital education and remote learning. Online platforms have replaced conventional classroom environments, thus, there is a critical need for creative solutions that can mimic the individualized instruction and assistance that teachers give in physical classrooms. Our answer to this demand is AlgoWhiz, which uses cutting-edge AI technology to provide learners all over the world with dynamic, customized, and interactive explanations of computer science algorithms. Our goal with AlgoWhiz is to enable students to confidently and easily navigate the complexity of computer science algorithms. We want to open the door for a more inclusive and equitable learning environment by offering easily available and interesting instructional resources. Our common conviction that education has the ability to improve lives and our dedication to ensuring that everyone has access to it are what drove us to choose this project.

The choice of AlgoWhiz's logo, a purple circle with a robot motif, was carefully considered to reflect the essence and objectives of the platform. The circle represents unity and completeness, symbolizing the holistic learning experience offered by AlgoWhiz. It signifies the interconnectedness of various computer science concepts and the seamless flow of knowledge within the platform. The color purple was chosen for its association with creativity, wisdom, and innovation, aligning with AlgoWhiz's mission to foster intellectual curiosity and exploration. Additionally, the robot motif embodies the technological aspect of the platform, highlighting its use of AI-driven algorithms and advanced learning methodologies. Overall, the logo serves as a visual representation of AlgoWhiz's commitment to providing a comprehensive, engaging, and cutting-edge educational experience in the field of computer science.

II. UML DIAGRAMS

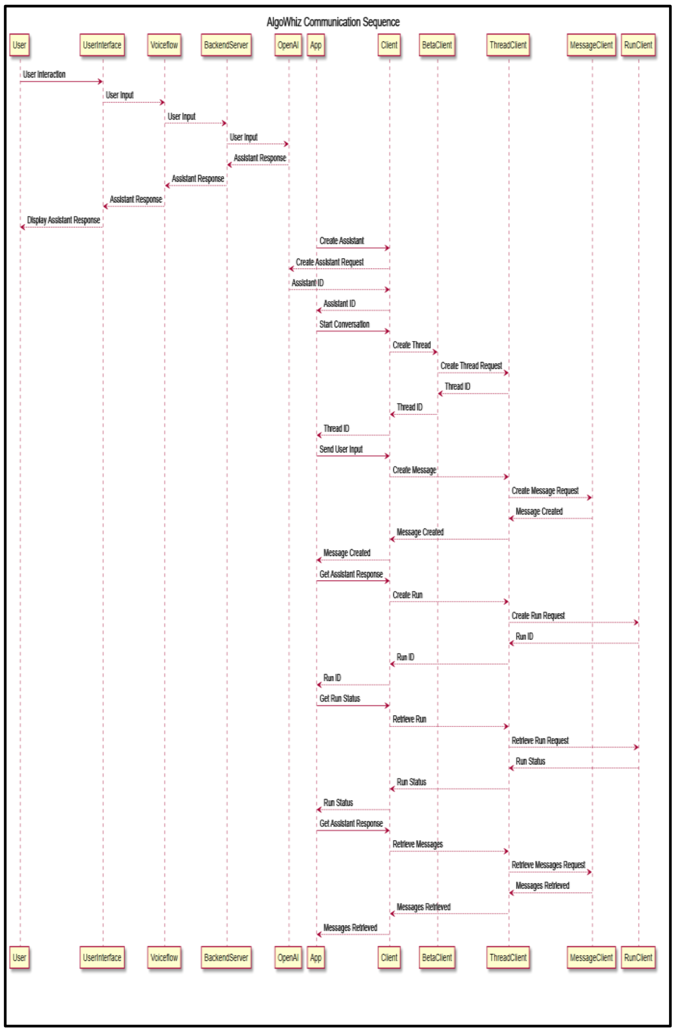
A. Class Diagram


App
assistant_id: str
start_conversation(): Response
chat(): Response
Thread
id: str
Run
id: str
status: str
Functions
create_assistant(client: OpenAI): str
MessageList
data: list
OpenAI
api_key: str
init(api_key: str)
Client
api_key: str
beta: BetaClient
Assistant
id: str
BetaClient
assistants: AssistantClient
threads: ThreadClient
AssistantClient
create(instructions: str, model: str, tools: list, file_ids: list): Assistant
ThreadClient
messages: MessageClient
runs: RunClient
create(): Thread
MessageClient
create(thread_id: str, role: str, content: str)
list(thread_id: str): MessageList
RunClient
create(thread_id: str, assistant_id: str): Run
retrieve(thread_id: str, run_id: str): Run

The UML class diagram provided outlines a sophisticated software architecture designed to interact with OpenAI's API, focusing on managing assistants, threads, messages, and execution runs. At the core of the system, the Functions class acts as a utility provider, facilitating the creation of an assistant through interactions with the OpenAI class, which handles API initialization and authentication. The main application logic is encapsulated within the App class, which manages user interactions and conversations, leveraging services from the Functions class and directly interfacing with the BetaClient for more specialized tasks. The Client serves as a generic container that sets up the API key and manages a BetaClient, which in turn oversees various client types such as AssistantClient for creating and managing assistants, and ThreadClient for handling conversation threads. Within ThreadClient, there are nested MessageClient and RunClient classes responsible for message management and execution of tasks (runs) within threads, respectively. These clients ensure the flow of messages and the management of task executions, enhancing the interaction dynamics within threads. Additionally, the diagram includes classes like Assistant, Thread, and Run, which represent entities with unique identifiers that perform specific roles within the system—assistants execute tasks, threads manage conversations, and runs handle individual task executions. The MessageList class aggregates messages, providing utilities for managing collections of messages within threads. Overall, the diagram illustrates a modular and well-structured system that emphasizes separation of concerns and detailed management of AI-driven interactions, threading, and message handling, all crucial for maintaining an efficient and responsive user experience in applications that leverage OpenAI's capabilities

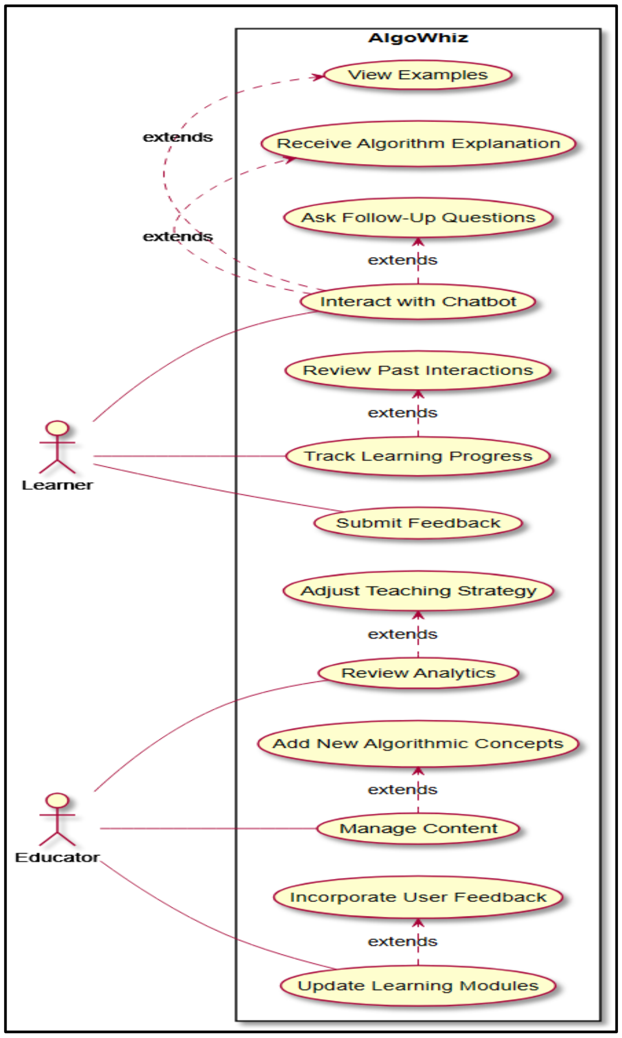
B. Sequence Diagram

The sequence diagram delineates the stepwise progression of operations within the AlgoWhiz communication framework, beginning with user interaction and culminating in response delivery. It starts with the user engaging through the User Interface, which forwards the input to Voiceflow. Subsequently, Voiceflow channels the input to the Backend Server for processing. The Backend Server then transmits the input to OpenAI, which handles the request and furnishes an assistant response. Upon receiving the assistant's response from OpenAI, the Backend Server relays it back to Voiceflow, which in turn directs it to the User Interface for display to the user. Moreover, the diagram encompasses processes related to assistant and thread management, as well as message and run processing. This involves the initiation of assistant creation and thread management by the App, interactions between the Client, BetaClient, and ThreadClient, and the handling of messages and runs by RunClient. Overall, the sequence delineates a coherent flow of data and control mechanisms across the system, elucidating how diverse components collaborate to manage user inquiries and generate responsive outputs within an interactive AI-driven environment.



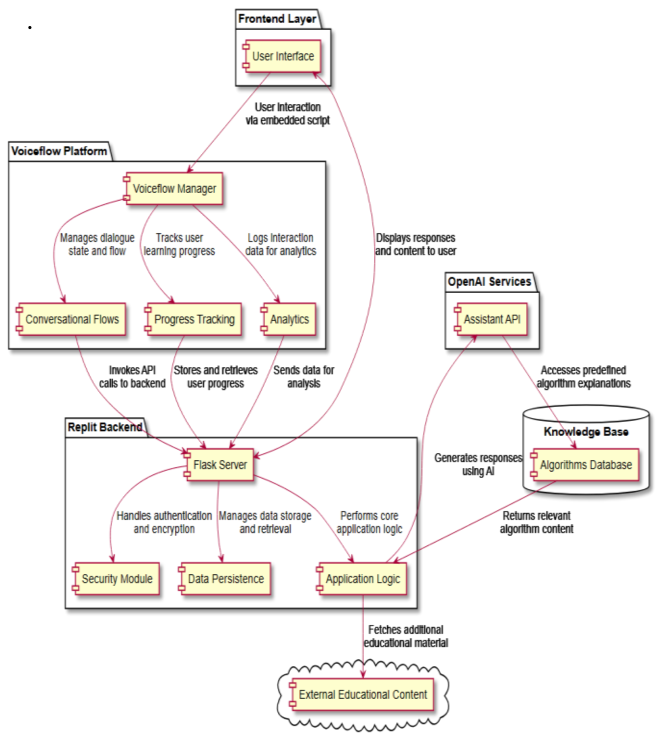
C. Use Case Diagram

The detailed use case diagram for the AlgoWhiz system illustrates how different users interact with the platform, highlighting key functionalities available to both learners and educators (admin). Users, primarily learners, can engage with the chatbot to ask questions, receive explanations, access examples, and interact in a conversational manner. Learners also have the ability to track their learning progress, view their history, and review past interactions. Additionally, users can submit feedback to improve service quality, while educators utilize analytics to gauge user engagement and educational outcomes, adjusting content and strategies as needed. Furthermore, educators play a crucial role in managing and updating educational content within the system, ensuring its relevance and comprehensiveness. Based on analytics and feedback, they refine and enhance learning modules to better meet user needs, creating a dynamic and adaptive learning environment within AlgoWhiz.



III. REVISED SOLUTIONS, INCLUDING SYSTEM ARCHITECTURE, ALGORITHMS AND FLOWCHARTS

A. System Architecture:


.
Frontend Layer
User Interface
User Interaction via embedded script
Voiceflow Platform
Voiceflow Manager
Manages dialogue state and flow
Tracks user learning progress
Logs interaction data for analytics
Displays responses and content to user
Conversational Flows
Progress Tracking
Analytics
OpenAI Services
Assistant API
Invokes API calls to backend
Stores and retrieves user progress
Sends data for analysis
Accesses predefined algorithm explanations
Knowledge Base
Algorithms Database
Replit Backend
Flask Server
Generates responses using AI
Handles authentication and encryption
Manages data storage and retrieval
Performs core application logic
Returns relevant algorithm content
Security Module
Data Persistence
Application Logic
Fetches additional educational material
External Educational Content

The design of AlgoWhiz encompasses front-end interaction, backend processing, data management, and third-party integrations.

The following components are central to the system's architecture:

User Interface (UI):
Developed using Carrd, the UI provides a user-friendly and responsive design that allows learners to interact with the chatbot via a web interface. The design is focused on ease of use, ensuring learners of all levels can navigate the interface effectively.

Backend Server:
Hosted on Replit, the backend server runs Flask, a lightweight WSGI web application framework. Flask handles HTTP requests and serves as a gateway for communication between the UI and the AI model.

OpenAI's Assistant API:
This API is the core of the AI functionalities in AlgoWhiz, responsible for processing user queries and generating intelligent responses that facilitate learning. It handles the computational complexity of understanding and responding to algorithm-related questions.

Voiceflow Integration:
Voiceflow is integrated as the workflow manager and chat interface, responsible for designing and implementing the chatbot's conversational flows. It also tracks user interactions, providing analytics and enabling progress tracking.

Replit Dev URL:
The Dev URL provided by Replit's webview is a crucial link between the backend and Voiceflow, allowing for API calls to be made in response to user interactions captured on the Carrd UI.

Security and Privacy:
Ensuring the confidentiality and integrity of user data is paramount in the system design. Best practices in security, including encryption and secure data handling protocols, are implemented throughout the architecture.

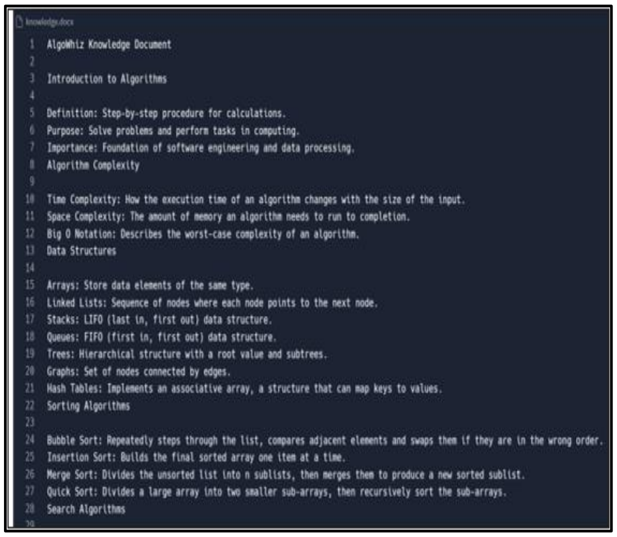
Maintenance and Updates:
Continuous integration and deployment practices are in place to facilitate ongoing updates and maintenance. The architecture allows for incremental updates to the chatbot's knowledge base and functionality without downtime or service interruption.

Yes, including the knowledge document that the chatbot retrieves information from would be beneficial to provide a comprehensive overview of the system's architecture. Here's how you could integrate it into the description:

Knowledge Document:
The chatbot retrieves information from a comprehensive knowledge document, which serves as a repository of algorithm-related content. This knowledge base contains detailed explanations, code examples, and educational resources that enable the chatbot to provide accurate and relevant responses to user queries. Integration with the knowledge base ensures that users have access to up-to-date and authoritative information, enhancing the educational

value of the system. The chatbot is designed to only answer questions that fall within the scope of the knowledge document. This scope enforcement mechanism ensures that responses provided by the chatbot are accurate, reliable, and aligned with the educational objectives of AlgoWhiz. By limiting responses to topics covered in the knowledge document, the chatbot maintains consistency and prevents users from receiving misleading or incorrect information. Additionally, this approach fosters a focused learning environment, encouraging users to explore relevant concepts and deepen their understanding within the established scope.

Here is a snapshot of the knowledge document we used:



```
knowledge.docx
AlgoWhiz Knowledge Document

Introduction to Algorithms

Definition: Step-by-step procedure for calculations.
Purpose: Solve problems and perform tasks in computing.
Importance: Foundation of software engineering and data processing.
Algorithm Complexity

Time Complexity: How the execution time of an algorithm changes with the size of the input.
Space Complexity: The amount of memory an algorithm needs to run to completion.
Big O Notation: Describes the worst-case complexity of an algorithm.
Data Structures

Arrays: Store data elements of the same type.
Linked Lists: Sequence of nodes where each node points to the next node.
Stacks: LIFO (last in, first out) data structure.
Queues: FIFO (first in, first out) data structure.
Trees: Hierarchical structure with a root value and subtrees.
Graphs: Set of nodes connected by edges.
Hash Tables: Implements an associative array, a structure that can map keys to values.
Sorting Algorithms

Bubble Sort: Repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order.
Insertion Sort: Builds the final sorted array one item at a time.
Merge Sort: Divides the unsorted list into n sublists, then merges them to produce a new sorted sublist.
Quick Sort: Divides a large array into two smaller sub-arrays, then recursively sort the sub-arrays.
Search Algorithms
```

B. ALGORITHMS AND FLOWCHARTS

AlgoWhiz is powered by a series of algorithms that work in tandem to provide an interactive and intelligent learning experience.

1. Input Processing Algorithm

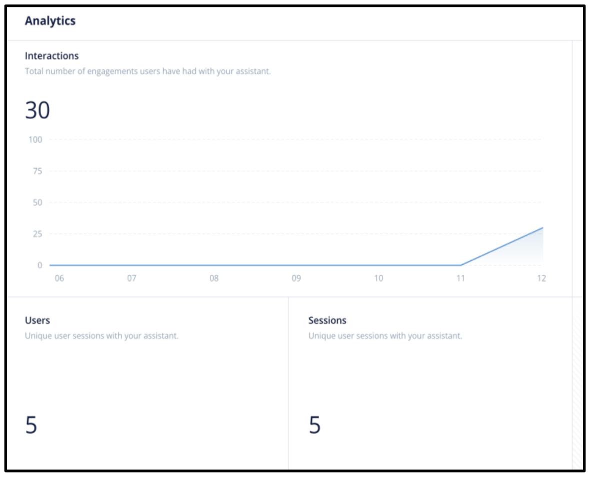
When a user sends a message through the Carrd website interface, the chatbot receives it and begins processing. Using natural language processing (NLP) techniques, the message is analyzed to identify the user's intent. This step involves understanding the meaning and purpose behind the message to determine the user's goal or query. Once the intent is recognized, the chatbot retrieves relevant information from its knowledge base or generates it on the fly using the OpenAI's Assistant API. This could involve accessing a database of pre-defined responses or dynamically generating content based on the user's request. With the necessary information at hand, the chatbot formulates a response. This could involve pulling from the pre-defined knowledge base to provide a relevant answer or dynamically creating content such as code examples, explanations, or other relevant information to address the user's query or request. The response is then sent back to the user via the Carrd website interface, completing the interaction.

2. Response Generation Algorithm:

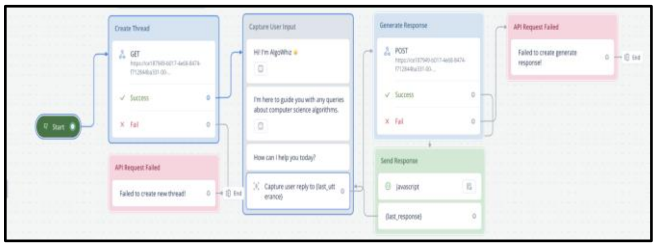
When the algorithm receives a user's query, it initiates an interaction with OpenAI's API by sending a request that includes the user's query and relevant context from the chat history. This process ensures that the AI model has the necessary information to generate a tailored response. Using the AI model, the API generates a response that encompasses explanations, code snippets, or guidance on algorithms, depending on the nature of the user's query. The AI leverages its understanding of the topic to provide accurate and helpful information to the user. Following content generation, the response undergoes post-processing to ensure readability and coherence. Any necessary additional information is included to enrich the response and provide a comprehensive answer to the user's query. This step is crucial for delivering a polished and informative response. As users interact with the content provided by the chatbot, their interactions are monitored to gather feedback. This feedback loop allows the system to adapt and personalize the learning experience for users, ensuring that future interactions are even more tailored and effective.

3. Analytics Handling Algorithm:

Data Collection begins as users engage with the chatbot through the Carrd interface, where their interactions are tracked using Voiceflow's built-in analytics. This data includes details on user engagement, interaction paths, and other relevant metrics. Following data collection, the information is subjected to Analysis to extract insights into user behavior and the effectiveness of the chatbot's responses. By examining patterns and trends in the collected data, valuable insights are gained, informing future improvements and optimizations to enhance the user experience. Regular Report Generation is conducted to consolidate key metrics and trends identified through the analysis phase. These reports serve as a reference point for evaluating the chatbot's performance and effectiveness over time, aiding in decision-making for future enhancements. Interaction Flow commences when a user initiates a conversation through the Carrd interface, triggering a request to the Flask server hosted on Replit. The server handles the request by routing it to the OpenAI API, which includes the user's current chat context to maintain continuity in the conversation. The AI Processing phase unfolds as the OpenAI API interprets the user's request, accesses the knowledge base, and generates an appropriate response tailored to the query. Once the response is generated, it is returned to the Flask server, which subsequently delivers it back to the user through the Carrd interface. Simultaneously, Voiceflow Orchestration manages session management, ensuring the conversation flows smoothly and tracking user progress and interaction metrics. This orchestration ensures a cohesive user experience and enables the chatbot to adapt dynamically to user inputs and preferences.


Analytics
Interactions
Total number of engagements users have had with your assistant.
30
100
75
50
25
0
06
07
08
09
10
11
12
Users
Unique user sessions with your assistant.
5
Sessions
Unique user sessions with your assistant.
5

C. Integration of Voiceflow:

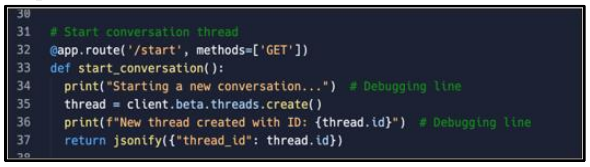

Start
Create Thread
GET
Success
Fail
API Request Failed
Failed to create new thread!
End
Capture User Input
Hi! I'm AlgoWhiz
I'm here to guide you with any queries about computer science algorithms.
How can I help you today?
Capture user reply to {last_utterance}
Generate Response
POST
Success
Fail
Send Response
javascript
{last_response}
API Request Failed
Failed to create generate response!
End

A key component of AlgoWhiz's design is voiceflow, which skillfully connects the Python backend logic running on Repliwith the frontend user interactions. It is essential to controlling and coordinating the chatbot's dialogue flows in order to provide a seamless user experience. We can create and run complex dialog scenarios using Voiceflow that react intelligently and precisely to user inputs.

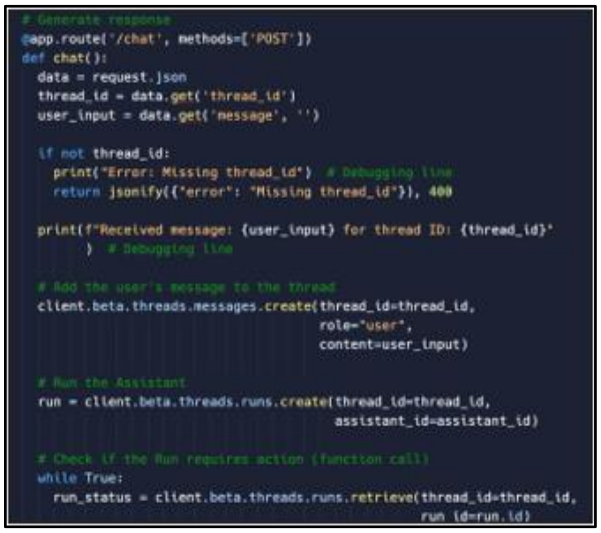
How Voiceflow Functions Within AlgoWhiz:

Frontend Interaction Management: Voiceflow provides a number of tools for building conversational interfaces that are interesting and interactive. On the Carrd website, these interfaces serve as the initial point of contact for users, gathering what they want and facilitating conversational engagement in real time.

API Call Coordination: Replit's Python backend receives API requests from Voiceflow's conversational flows. Voiceflow's capacity to manage complex API logic enables AlgoWhiz to quickly process user queries and retrieve pertinent answers from the knowledge base on the back end.
The below section of code is related to the “create thread” section of the voiceflow process shown above. This allows the creation of a new thread and the respective thread id is stored for future.


```
# Start conversation thread
@app.route('/start', methods=['GET'])
def start_conversation():
  print("Starting a new conversation...")  # Debugging line
  thread = client.beta.threads.create()
  print(f"New thread created with ID: {thread.id}")  # Debugging line
  return jsonify({"thread_id": thread.id})
```

The below section of code corresponds to the “generate response” section of voiceflow process.



```
# Generate response
@app.route('/chat', methods=['POST'])
def chat():
  data = request.json
  thread_id = data.get('thread_id')
  user_input = data.get('message', '')

  if not thread_id:
    print("Error: Missing thread_id")  # Debugging line
    return jsonify({"error": "Missing thread_id"}), 400

  print(f"Received message: {user_input} for thread ID: {thread_id}"
       )  # Debugging line

  # Add the user's message to the thread
  client.beta.threads.messages.create(thread_id=thread_id,
                                      role="user",
                                      content=user_input)

  # Run the Assistant
  run = client.beta.threads.runs.create(thread_id=thread_id,
                                        assistant_id=assistant_id)

  # Check if the Run requires action (function call)
  while True:
    run_status = client.beta.threads.runs.retrieve(thread_id=thread_id,
                                                   run_id=run.id)
```

Conversational Flow Orchestration: AlgoWhiz's voiceflow plays a key role in coordinating the instructions and data flow. It guarantees that every user interaction is seamless, well-organized, and consistent with the developers' intended purpose. It oversees the logic for response creation, input validation, and error handling, giving users feedback or further instructions as required..

Error Handling and Input Validation: Voiceflow's strong error handling features enable the smooth handling of any problems that may arise during API calls or dialogue. This guarantees that any unforeseen user input or system faults are managed effectively, preserving the conversation's flow and giving users clear directions or information instead of interfering with their experience.

The integration capabilities of Voiceflow go beyond conversation management to improve accessibility and user engagement. It enables AlgoWhiz to provide individualized experiences by identifying user preferences and adjusting dialogues appropriately. The advanced analytics offered by Voiceflow offer valuable insights into user behaviors and interaction patterns, facilitating ongoing enhancements to the chatbot's functionality.

From a technical perspective, the integration of voiceflow within AlgoWhiz required several steps:

- Setting up Voiceflow with the appropriate entities, intents, and conversation management techniques to address a range of algorithm-related inquiries.
- Establishing webhook connections to enable real-time processing and response generation between Voiceflow's conversational flows and Replit's Flask routes.
- Using a customized HTML <script> element, the Voiceflow chat widget was embedded into the Carrd website, allowing the chatbot to display and work without a hitch.

By combining Voiceflow in this way, AlgoWhiz raises the standard for interactive learning systems by creating a harmonic balance between an easy-to-use user interface and a potent AI-driven backend.

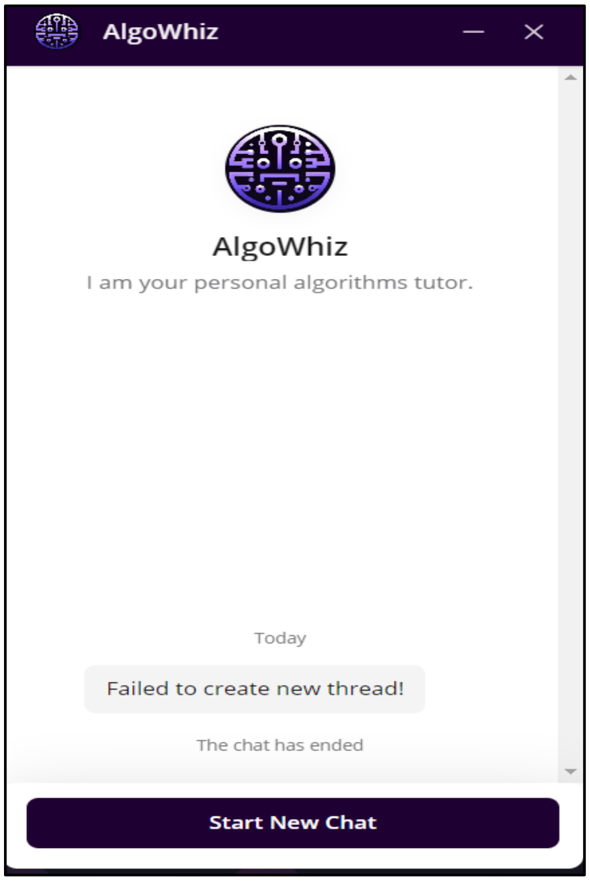
IV .IMPLEMENTATION DETAILS

AlgoWhiz is a sophisticated AI chatbot system designed to support learners in mastering computer science algorithms. Hosted on Replit, the project employs OpenAI's Assistant API to enable a wide range of functionalities from educational content delivery to progress tracking. It integrates with Voiceflow to manage conversational flows and with a Carrd website to provide an accessible user interface.

A. How to Implement Our Project:

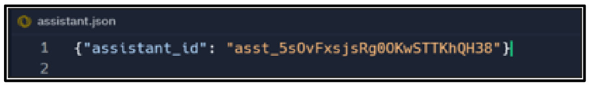
To replicate or build upon AlgoWhiz, one may fork our Replit project at https://replit.com/@DhruvManiar/AlgoWhiz.

Be sure to run the Python code before you ask AlgoWhiz questions. A new run is needed each time you close down the AlgoWhiz website. If you attempt to ask AlgoWhiz a question without running the Python, you will encounter something like this:


AlgoWhiz
AlgoWhiz
I am your personal algorithms tutor.
Today
Failed to create new thread!
The chat has ended
Start New Chat

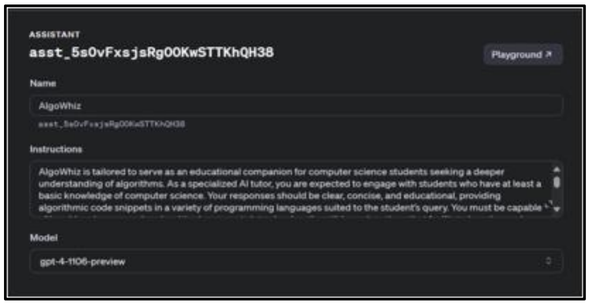
The implementation strategy requires a composite knowledge of Python, Flask, OpenAI's API, and Voiceflow.

On running the Python code an assistant.json file is created if one does not exist already. This is what the file looks like


assistant.json
```
{"assistant_id": "asst_5sOvFxsjsRg0OKwSTTKhQH38"}

```

You should see the same assistant created in your Open AI account.


ASSISTANT
asst_5sOvFxsjsRgOOKwSTTKhQH38

Name
AlgoWhiz
asst_5sOvFxsjsRgOOKwSTTKhQH38

Instructions
AlgoWhiz is tailored to serve as an educational companion for computer science students seeking a deeper understanding of algorithms. As a specialized AI tutor, you are expected to engage with students who have at least a basic knowledge of computer science. Your responses should be clear, concise, and educational, providing algorithmic code snippets in a variety of programming languages suited to the student's query. You must be capable

Model
gpt-4-1106-preview

Open AI gives free credit to all users and we have spent $1 so far on the project. If your free credit has expired you can add more credit by going to the Billing section in Open AI.

Key steps include:

- Forking the project on Replit to get a pre-configured environment with the necessary codebase.
- Setting up an OpenAI account to access the Assistant API and obtaining necessary credentials.
- Configuring Voiceflow to tailor the conversational experience, matching it with the AlgoWhiz design for consistent interaction.
- Embedding the Voiceflow script on a Carrd website, allows users to engage with the chatbot.

B. Functionalities Completed:

- Backend Infrastructure: Established on Replit to manage the AI assistant's core functions.
- Dev URL Linking: Successful integration with Voiceflow, creating a seamless communication channel between the user interface and backend.
- Frontend Communication: The Voiceflow-generated script embedded within the Carrd website.
- Chat History: Maintaining logs of user interactions to enable continuity in learning and conversation.
- Progress Tracking: Implemented via Voiceflow to help users follow their learning journey.Analytics: Voiceflow's analytics to measure engagement and identify opportunities for improvement.
- Code Generation and Retrieval: Utilizing the Assistant API for the dynamic generation of educational content and examples.

C. Functionalities Pending:

- File Uploading Feature: To enhance personalized learning through the submission of user-generated content.

- User History Personalization: Developing the capability to provide tailored experiences based on individual user interactions, while maintaining privacy.

D. Implementation/Technical Issues

- Ensuring seamless integration between Replit's backend and Voiceflow.
- Addressing scalability as user numbers increase.
- Maintaining data security with the planned file uploading and personalization features.

E. Experimental Results

Initial tests of AlgoWhiz have yielded encouraging outcomes, demonstrating robust user engagement and fostering a positive learning environment. Throughout these tests, AlgoWhiz has showcased its ability to efficiently address relevant queries while maintaining focus on learning objectives.

During testing, users posed a variety of questions related to computer science concepts and algorithms. These inquiries ranged from basic queries about algorithm implementations to more complex discussions on algorithmic optimization techniques. Additionally, users explored topics such as data structures, computational complexity, and algorithmic design paradigms. The diversity of questions reflects the comprehensive coverage of computer science topics within AlgoWhiz's knowledge base. In instances where users posed questions unrelated to computer science, AlgoWhiz effectively refrained from responding. The system's design ensures that it only engages with queries within the scope of its knowledge base, thereby maintaining focus on relevant educational content. This approach prevents users from receiving erroneous or off-topic responses, preserving the integrity of the learning experience. Instead, AlgoWhiz gently redirects users back to the intended learning path, encouraging exploration of pertinent computer science topics.

F. Analysis of Experimental Results

Analysis of in-class live demos and user interactions has confirmed the reliability and educational value of AlgoWhiz. It has demonstrated an effective live performance, suggesting readiness for broader deployment.

G. Local/Global Impacts:

AlgoWhiz has the potential to significantly impact the field of education, offering 24/7 access to learning resources that can benefit individuals and organizations. By supporting personalized, self-paced learning, it can contribute to a more knowledgeable and skilled society.

V. VALIDATION OF THE DEVELOPED APPLICATION WITH THE INTENDED USERS

An essential stage in guaranteeing the AlgoWhiz application's usability, efficacy, and alignment with the demands of the target audience was the validation phase. This stage provided us with important insights from the validation process, and we have described the approaches used to validate the application with target users below.

1. Live Interaction through QR Code

Enabling live interaction with the AlgoWhiz website during validation sessions was made possible by the smooth integration of a QR code into the presentation materials, which was a crucial component of the validation process. Participants—consisting of computer science students—were able to interact directly with the chatbot and get a firsthand look at its operation thanks to this creative technique. The AlgoWhiz portal was instantly accessible to participants by scanning the QR code with their computers or mobile devices. This allowed them to pose queries, test algorithms, and offer real-time feedback. In addition to encouraging active engagement, this interactive element acted as a strong validation mechanism, providing insightful information on the application's efficacy, usability, and fit with the target audience's demands. The application's capabilities were demonstrated in real time, which improved the feedback process and reaffirmed important conclusions drawn via later validation techniques..

2. User Testing Sessions

To obtain qualitative input and insights from people who correspond to the target user demographic, user testing sessions were held. After gaining access to the AlgoWhiz platform, participants were required to engage in a variety of activities, including posing queries on computer science algorithms and offering comments on the chatbot's answers. Participants' interactions and input were closely monitored and documented during these sessions in order to pinpoint problems with usability, understanding, and areas that needed work.

3. Surveys and Questionnaires:

To get quantitative information on user happiness, usability, and perceived efficacy of the AlgoWhiz program, surveys and questionnaires were sent to a few computer science student participants. They were asked to score their overall satisfaction, responsiveness, and explanation clarity, as well as other aspects of the chatbot's interaction. Furthermore, participants were able to provide areas of concern, comprehensive comments, and recommendations for improvements using open-ended questions.

4. Feedback Analysis:

To find recurrent themes, patterns, and areas of agreement among participants, input that was gathered from user testing sessions, surveys, and questionnaires was methodically examined. Frequently occurring problems and areas of discomfort were ranked in order of importance for more research and were resolved through iterative improvements to the layout, features, and content of the application. Furthermore, the website incorporated favorable feedback and commendations to strengthen its effective characteristics.

5. Iterative Refinement:

The AlgoWhiz program was continuously improved thanks to this iterative process, which also made sure that user preferences and demands were met. Aiming to maximize the entire user experience, refinements included improvements to conversational logic, user interface design, content correctness, and responsiveness.

6. Validation Results:

The in-class live demo of AlgoWhiz served as a critical validation phase, providing tangible evidence of its functionality and impact. This session allowed us to observe real-time interactions between students and the chatbot, offering valuable insights into the system's performance and user experience.

7. Demo Setup and Execution:

The demo was conducted in a controlled classroom environment with the participation of computer science students. Each student accessed AlgoWhiz through their devices by navigating to the project's hosted URL. The session was designed to simulate typical use cases, including querying different algorithms, requesting explanations, and interacting with the system's code generation features.

8. Observations:

During user engagement sessions, students actively interacted with AlgoWhiz, exploring its ease of use and accessibility. They posed both predefined queries and spontaneous questions, assessing the system's responsiveness and the depth of its knowledge base. AlgoWhiz demonstrated swift responsiveness, promptly addressing queries without noticeable delays. This highlighted the efficiency of the backend setup on Replit and the capability of the Flask framework to manage multiple simultaneous interactions effectively. The accuracy of AlgoWhiz's responses was consistently high, with the OpenAI Assistant API generating contextually relevant explanations and code snippets for various computer science algorithms. This affirmed AlgoWhiz's educational utility by providing detailed and informative responses. Additionally, the system effectively handled unrelated queries, maintaining focus on its educational purpose by refraining from responding to off-topic questions. This aspect was thoroughly tested during the demo, reinforcing AlgoWhiz's commitment to fostering a conducive learning environment.

9. Key Metrics:

User satisfaction with the system was consistently high, with users praising the interactive learning experience and the clarity of explanations provided by AlgoWhiz. Positive feedback was particularly notable in these areas, indicating that users found the platform engaging and informative. Additionally, students reported a significant improvement in their understanding of the discussed algorithms, indicating that AlgoWhiz had a high educational impact. This demonstrated the effectiveness of the platform as a learning tool. Furthermore, users commended the system's usability, highlighting the ease of navigating the user interface and the intuitiveness of the interaction design as major positives. This positive reception underscored AlgoWhiz's commitment to providing a user-friendly experience that enhances learning outcomes.

10. Challenges Noted:

The demo of AlgoWhiz was largely successful, garnering positive feedback from users. However, there were areas identified for improvement based on user suggestions. Some users proposed that AlgoWhiz could benefit from handling more complex queries and providing additional examples for each algorithm, enhancing the depth of engagement with the platform. Additionally, there was a request for more personalized learning tracks based on individual user progress and interactions, indicating a desire for tailored educational experiences.

Overall, the demo confirmed AlgoWhiz as a robust educational tool capable of enhancing the learning experience for students in computer science. Its ability to deliver precise, relevant content interactively was well-received. Moving forward, the team plans to incorporate user feedback to refine and expand AlgoWhiz's capabilities, particularly focusing on improving query handling and implementing personalized learning features. This iterative approach aims to optimize the platform further and better meet the diverse needs of its users.

VI. EACH TEAM MEMBER'S CONTRIBUTION IN TERMS OF DESIGN AND DEVELOPMENT

1. Isha Koregave:
Development of Python Scripts Hosted in Replit:

- Used the Python programming language to actively engage in the team project's coding and development phases.
- I worked with other team members to develop and implement functions, making sure the script complied with project criteria.
- Used Replit as the development environment, which made it easy for team members to collaborate and maintain version control.
- I helped with debugging and troubleshooting, locating and fixing problems, and guaranteeing the dependability and functionality of the script.

Research and Implementation of Voiceflow:

- I researched Voiceflow to explore its potential integration within the project as a workflow management system.
- Identified relevant features and capabilities of Voiceflow that could enhance the project's functionality and user experience.
- Led the implementation process for Voiceflow, thus integrating its components into the project's architecture to enable successful tie-up between the Python code and UI.
- Resolved integration challenges while implementing the project.

Preparation of Presentation:

- Took the lead in organizing and creating the presentation for the team project, showcasing its objectives, progress, and outcomes.
- Developed a structured outline and content for the presentation, highlighting key project milestones, achievements, and learnings.
- Utilized effective visual aids, such as slides and diagrams, to convey information clearly and engage the audience.

- Refined the presentation delivery to ensure clarity, coherence, and effectiveness in communicating the project's message.
- Incorporated feedback from team members to enhance the presentation's quality and impact, ultimately delivering a compelling and informative presentation to stakeholders.

IEEE Document:

- Revised solutions, including system architecture, algorithms and their flowchart
- Validation of the developed application with the intended users
- Revised UML diagrams – class, use case, and sequence diagrams with explanation

2. Dhruv Maniar

Team Website Development and Maintenance:

- As a core member of the team, I took the lead in developing our team website. This involved conceptualizing the layout, design, and functionality.
- I curated and regularly updated the website content to ensure accuracy and relevance. This included adding team member information, roles, group activities, schedules, and validation results.
- I implemented features to showcase team member profiles, including their roles within the team, ensuring transparency and clarity regarding responsibilities.
- Integrated a section dedicated to group activities and schedules, providing visitors with insights into ongoing projects and upcoming events.
- Implemented a section to display validation results, highlighting the team's achievements and progress**.**

Algowhiz Website Development:

- I spearheaded the development of the Algowhiz website, ensuring it aligned with the branding and objectives.
- Implemented a chat pop-up feature to enhance user engagement and provide immediate assistance to visitors.

Chatbot Fine-Tuning:

- I collaborated with the team to define the scope of the chatbot's responses, ensuring it focused on relevant topics and provided accurate information.
- I leveraged my expertise to fine-tune the chatbot's algorithms, optimizing their performance and enhancing the user experience**.**

Knowledge Document:

- I researched and put together a knowledge document that the chatbot used to retrieve information from when answering the user's questions

IEEE Document:

- Revised projection description and motivation
- Implementation Details
- Validation of the developed application with the intended users