Project Documentation: Code Visualizer AI

1. Introduction

Code Visualizer AI is an interactive Streamlit-based application designed to provide step-by-step visual explanations of how source code executes.
The system dynamically highlights each line of code, tracks variable updates, visualizes control flow, and provides insights into program behavior.
It also integrates debugging and complexity analysis functionalities, powered by **Groq's Llama 3.1 models** via **LangChain**.

---

2. Objectives

- To enable developers and learners to understand code execution flow visually.
- To support multiple programming languages for broader accessibility.
- To detect and suggest fixes for logical and syntax issues automatically.
- To provide time and space complexity analysis for functions and loops.
- To store, view, and export past analyses for continuous learning.

---

3. Key Features

- **Interactive Code Visualization**
  Step-by-step explanation of code execution with highlighted lines, variable updates, and control flow visualization.
- **Debugger Integration**
  Automated detection of issues, categorized by type (syntax, logic, runtime), with suggested fixes and corrected code output.
- **Complexity Analysis**
  Provides Big-O time and space complexity for each function, along with loops, recursion, and memory usage insights.
- **History Management**
  Maintains a persistent session history, allowing users to revisit, export, or delete past analyses.
- **Multi-Language Support**
  Supports code written in Python, JavaScript, C++, Java, C#, Go, Rust, and Ruby.
- **User Controls**
  Playback controls to auto-play or manually step through each execution phase.

---

4. System Architecture

4.1 Repository Structure

- **streamlit_app.py** — Main UI component managing visualization, navigation, and user interaction.
- **analyzer.py** — Contains LLM interaction logic, prompt generation, error handling, and data normalization.
- **requirements.txt** — Python dependency list.
- **pic.jpg** — Background image for the user interface.
- **test.py** — Sample demo code for initial testing.

4.2 Core Modules

- **get_groq_api_key()** — Retrieves the `GROQ_API_KEY` from environment variables or Streamlit secrets.
- **analyze_code_with_llm()** — Sends structured prompts to the LLM and returns a step-by-step execution plan.
- **analyze_errors_with_llm()** — Scans code for issues and generates corrected versions.
- **analyze_complexity_with_llm()** — Estimates algorithmic complexity for each function in the input code.

4.3 User Interface Flow

- **Navigation:** Sidebar with Home, History, Debugger, and Complexity options.
- **Session State:** Maintains variables for current code, language, steps, play state, and analysis results.
- **Views:**
    - *Home:* Visualization and analysis execution.
    - *History:* Manage and review previous analyses.
    - *Debugger:* Detect and fix code issues.
    - *Complexity:* Display algorithmic complexity insights.

---

5. Workflow Overview

1. **Input Phase:**
   The user pastes source code and selects a programming language.
2. **Processing Phase:**
   The system constructs a structured JSON prompt and sends it to the Groq LLM through LangChain.
3. **Response Handling:**
   The returned JSON is parsed and repaired if necessary using the `json-repair` library.

4. **Visualization Phase:**
   The app displays highlighted code lines, variable values, stack information, control flow, and memory states.
5. **Navigation Phase:**
   Users can manually explore or auto-play through the execution steps.

---

6. Installation

6.1 Prerequisites

- Python 3.9 or higher
- Internet access for API connectivity

6.2 Installation Steps

1. Clone or download the project repository.
2. Install dependencies using the command:
   ```
   pip install -r requirements.txt
   ```

---

7. Configuration

7.1 Environment Variable Setup

The application requires a valid **Groq API Key**.

Option 1: Using a `.env` file

```
GROQ_API_KEY=your_actual_api_key
```

Option 2: Using Streamlit Secrets

```
GROQ_API_KEY = "your_actual_api_key"
```

If the API key is not provided, the analyzer functions will raise a clear error message indicating the missing configuration.

---

8. Running the Application

To start the application, run the following command:

```
streamlit run streamlit_app.py
```

After successful execution, open the local URL provided in the terminal to access the web interface.

---

9. User Guide

9.1 Home (Visualizer)

- Select the desired programming language.
- Paste the source code and click **Analyze Code**.
- View the summary and detailed execution flow.
- Use the navigation buttons to step through or auto-play execution.

9.2 History

- Access and manage previously analyzed sessions.
- Reload old analyses or export them as JSON files.
- Delete unnecessary records.

9.3 Debugger

- Paste your code and click **Scan for Issues**.
- View identified problems with their line numbers and explanations.
- Retrieve an automatically generated corrected version of the code.

9.4 Complexity Analysis

- Paste code and click **Analyze Complexity**.
- Review per-function complexity results with Big-O estimates.
- View visual growth charts for algorithmic scaling.

---

10. Data Model

Example structure for saved analysis history:

```
{
  "id": "<type>-<timestamp>-<n>",
  "timestamp": 1710000000,
  "language": "python",
  "code": "...",
  "summary": "...",
```

```
  "num_steps": 12,
  "kind": "home|debugger|complexity",
  "issues": [
    {"type": "logic", "line": 10, "explanation": "...", "suggestion": "..."}
  ],
  "corrected_code": "...",
  "complexity": {
    "functions": [{"name": "f", "time_complexity": "O(n)",
"space_complexity": "O(1)"}]
  }
}
```

---

11. Dependencies

- Streamlit
- Python-dotenv
- LangChain
- LangChain-Groq
- JSON-Repair

---

12. Troubleshooting

Issue	Possible Cause	Solution
Missing API Key	GROQ_API_KEY not set	Add it to `.env` or Streamlit secrets
Empty Analysis	Large input or wrong language	Try smaller code or correct language selection
JSON Parsing Error	Response too large	Reduce code size or retry
UI Background Missing	`pic.jpg` not found	Ensure the image exists or use default background

---

13. Security and Privacy

- The app sends user code to Groq's API through LangChain for analysis.
- Sensitive or proprietary code should not be submitted.
- API keys are securely stored in local environment or Streamlit secrets, and are never logged.

---

14. Limitations

- Execution traces are simulated using model predictions, not an actual interpreter.
- Complex runtime behaviors might not be perfectly replicated.
- Large code files may exceed the model's context window.

---

15. Future Enhancements

- Support for multi-file and uploaded project analysis.
- Advanced visualization for data structures and memory representation.
- More robust JSON validation and error recovery.
- Integration of static analysis tools for accuracy improvement.

---

16. Conclusion

The **Code Visualizer AI** provides an innovative and educational platform for understanding program execution.
By combining visualization, debugging, and complexity analysis in a unified interface, it bridges the gap between code learning and intelligent interpretation.
The project lays a foundation for future expansion into real-time code simulation and collaborative educational tools.