API Dash GSoC Contributor Proposal

Full Name: Mrityunjay Kukreti

Email: mrityunjaykukreti@gmail.com

Phone Number: +91-8929633669

Github: https://github.com/weeebhu

Time zone: IST(GMT+05:30)

Mentor: Ashita Prasad, Ankit Mahat

Summary

The main goal of DashBot is to act as a smart, AI-based assistant in API Dash to automate mundane yet crucial API development operations. With natural language interaction, DashBot will make debugging, documentation, testing, and visualization easy, thus greatly improving the developer experience and streamlining the API development cycle.

Project Abstract

We are creating DashBot, a smart AI helper for API Dash, to assist developers in automating monotonous and often time-consuming API-related tasks like debugging, documentation, testing, visualization, and frontend integration. Developers currently spend a significant amount of time manually interpreting API errors, writing or modifying documentation, creating test cases, and integrating endpoints into frontend codebases. Not only are these tasks time-consuming, but they are also vulnerable to inconsistency and human error. DashBot addresses this with natural language and large language models (LLMs) to process the API context and support developers in automating these actions.

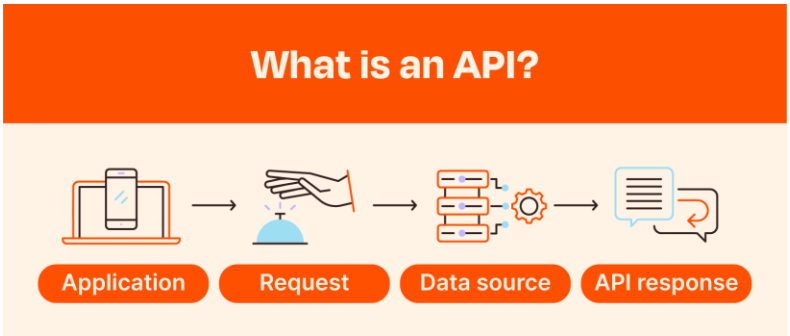
The mission is to narrow the gap between developer productivity and API complexity by providing human-like interaction capabilities with APIs. Developers can query DashBot questions such as "Why is the request not going through?" ", "Create a test for this endpoint," or "Make a chart for this response," and DashBot will provide actionable, context-sensitive responses. Our objectives are to create a modular, extensible assistant that can be used with various LLM backends, provide benchmark comparisons, and integrate well with the API Dash UI.". Non-goals are not creating a complete code editor or an API gateway—DashBot is an extension, not a substitute, that is meant to exist within the limits of the existing API Dash infrastructure.The aim is to enhance productivity, not change the fundamental workflow or infrastructure.

Background

In order to have a complete appreciation for the scope and motivation of this project, it is helpful to be knowledgeable in a few prominent areas:

- API Dash is a developer tool, open-source in nature, to test, debug, and document APIs. API Dash has an easy-to-use UI and has features such as environment management, request history, and data visualization. For details on how to use the tool, see the *API Dash Developer Docs*.

- APIs (Application Programming Interfaces) are structured interfaces through which software components can communicate. They are used by the majority of contemporary software development in the form of RESTful APIs or GraphQL APIs. A good introductory primer for APIs is available on *Wikipedia's API* page and *REST API Introduction on MDN*.



- Large Language Models (LLMs) such as OpenAI's GPT-4, Meta's LLaMA, and open-source models by Hugging Face are revolutionizing the way developers interact with systems and code. These models are able to understand natural language and produce code, explanations, and recommendations. For more information on LLMs, refer to *OpenAI's GPT-4 Technical Report* and *Meta's LLaMA paper*.

- LangChain and LlamaIndex are popular libraries for developing LLM-based apps. LangChain offers agents, tools, and memory components to build modularly, and LlamaIndex assists in indexing and querying data with LLMs. Visit the official *LangChain documentation* and *LlamaIndex docs* for more details.

- Flutter and React are two top frontend frameworks through which cross-platform applications are made. DashBot will create the integration code for these platforms. For reference, see the official *Flutter Docs* and *React Docs*.

Benefits to the Community

- **Enhanced Productivity:** Automates tedious API tasks, reducing development time.
- **Improved Debugging:** Provides AI-assisted debugging of API responses.
- **Better Documentation:** Auto-generates API documentation with explanations.
- **Automated Testing:** Generates test cases based on API usage patterns.
- **Data Visualization:** Creates interactive charts for API responses.
- **Seamless Integration:** Generates frontend code for React, Flutter, and other frameworks.
- **Benchmarking:** Helps users choose the best LLM for their use case.

Design Ideas

DashBot will be developed as a modular AI assistant integrated into API Dash, enabling developers to interact with APIs using natural language. It will be powered by LLMs (OpenAI, LLaMA, etc.) and structured using agentic frameworks like LangChain. The system will consist of multiple independent tools connected to a central orchestrator.

Key Components & Technologies

- **Core Agent Framework:**
  - **Tech:** LangChain, Python, FastAPI
  - **Purpose:** Orchestrate user input → tool selection → LLM output
  - Includes memory + prompt management
- **Modular Tools:**
  - **Debugger:** Parses status codes/errors → suggests fixes (Python + Regex parsing)
  - **Doc Generator:** Parses API requests → generates Markdown/OpenAPI-style docs
  - **Test Generator:** Creates unit/integration tests using LLM + example-based prompts
  - **Visualizer:** Uses Matplotlib, Plotly to generate graphs from JSON responses
  - **Frontend Code Generator:** Outputs Flutter/React code for API integration
- **LLM Benchmarking System:**
  - **Metrics:** Response accuracy, latency, cost per call
  - **UI/CLI:** Visual comparison of different backends (OpenAI, LLaMA, Gemini)

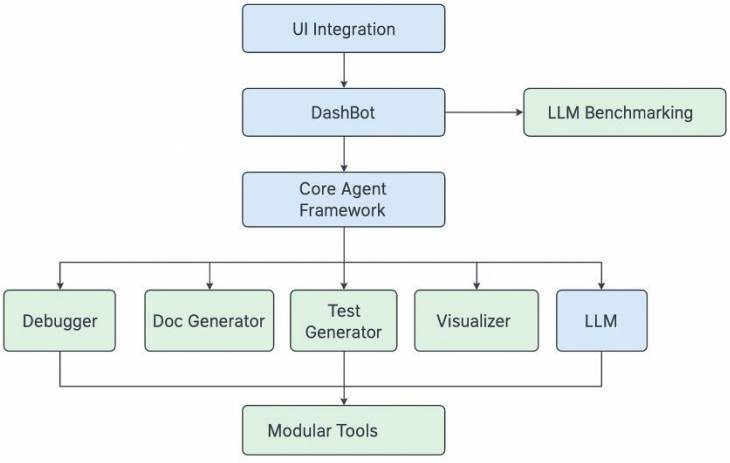
- **UI Integration:**
  - **Technologies:** Flutter, Dart
  - **Interface:** DashBot chat interface + action buttons for generated artifacts

- **Scaling Parameters:**
  - **Estimated API usage:** 1–5 QPS per user during development
  - **Data Size:** Small JSON payloads (~1KB–100KB)
  - **Max Users (Initial):** Targeting dozens to hundreds of concurrent developers

- **Rollout Strategy:**
  - Start as a CLI/side-panel tool
  - Integrate with API Dash UI via plugins/widgets
  - Gradual feature flagging for different modules
  - Collect usage logs (opt-in) for feedback loops

- **Security & Privacy Considerations:**
  - Avoid sending sensitive data to third-party LLMs without user consent
  - Use local LLMs when privacy is critical
  - Anonymize logs used for evaluation
  - Rate limit + API key encryption for secure usage

- **Ownership Strategy:**
  - Weekly milestones, progress reports, and PRs
  - Active engagement with mentors and community
  - Prior experience with agents, APIs, and LLM integration
  - Testing + benchmarking to ensure quality delivery



Pseudocode

```
# // Main entrypoint for DashBot
# function start_dashbot():
#     initialize_components()
#     while True:
#         user_input = get_user_input()
#         task_type = classify_task(user_input)
#         response = handle_task(task_type, user_input)
#         display_response(response)

# // Initialize core components
# function initialize_components():
#     load_llm_backend()
#     load_task_modules()
#     load_visualization_tools()
#     load_prompt_templates()

# // Classify user input into a task type
# function classify_task(user_input):
#     if contains_keywords(user_input, ["error", "status", "debug"]):
#         return "debug_api"
#     elif contains_keywords(user_input, ["doc", "explain", "description"]):
#         return "generate_docs"
#     elif contains_keywords(user_input, ["test", "unit test", "assert"]):
#         return "generate_tests"
#     elif contains_keywords(user_input, ["chart", "plot", "graph"]):
#         return "visualize_response"
#     elif contains_keywords(user_input, ["frontend", "integration", "code"]):
#         return "generate_frontend_code"
#     else:
#         return "generic_response"

# // Handle each task type using modular handler
# function handle_task(task_type, user_input):
#     switch task_type:
#         case "debug_api":
#             return debug_api_request(user_input)
#         case "generate_docs":
#             return generate_api_documentation(user_input)
#         case "generate_tests":
#             return generate_api_tests(user_input)
#         case "visualize_response":
#             return generate_visualization(user_input)
#         case "generate_frontend_code":
#             return generate_integration_code(user_input)
#         default:
#             return query_llm(user_input)

# // Debugging logic
# function debug_api_request(user_input):
#     error_info = extract_error_context(user_input)
#     prompt = build_prompt("debug_template", error_info)
#     return query_llm(prompt)

# // Generate documentation from API schema or response
# function generate_api_documentation(user_input):
#     api_info = parse_api_structure(user_input)
#     prompt = build_prompt("doc_template", api_info)
#     return query_llm(prompt)

# // Generate test cases for API usage
# function generate_api_tests(user_input):
#     api_response = get_api_sample_response(user_input)
#     prompt = build_prompt("test_template", api_response)
#     return query_llm(prompt)

# // Visualize API responses using charts
# function generate_visualization(user_input):
#     data = extract_data_from_response(user_input)
#     chart_config = interpret_visualization_intent(user_input)
#     return render_chart(data, chart_config)

# // Generate frontend integration code
# function generate_integration_code(user_input):
#     frontend_framework = detect_framework(user_input)
#     api_details = parse_api_info(user_input)
#     prompt = build_prompt("frontend_template", api_details, frontend_framework)
#     return query_llm(prompt)

# // Query the selected LLM backend
# function query_llm(prompt):
#     return call_selected_llm_api(prompt)

# // Benchmark different LLMs
# function benchmark_llms(test_set):
#     results = []
#     for model in available_models:
#         metrics = evaluate_model(model, test_set)
#         results.append({model: metrics})
#     return results
```

Code Affected

1. Main App UI (Flutter)

This is where DashBot's frontend interface will be integrated.

- lib/pages/home.dart:
  Likely to add DashBot UI trigger button and result display panels.
- lib/widgets:
  DashBot chat interface and response components will be added here.

2. Request/Response Handling Logic

Will be used by DashBot for debugging, generating docs/tests.

- lib/controllers/request_controller.dart:
  Used to get request-response metadata for DashBot tools.

3. Testing & Request Utilities

Useful for generating test cases via DashBot.

- test folder (all unit/integration tests): May be extended with auto-generated tests.

4. Backend LLM Agent (to be added)

This is a new module that you will create during GSoC. It will live in a separate Python repo or in a new backend/ directory within API Dash, depending on mentor preference.

- Proposed location: backend/agent/
  - agent.py – core orchestrator
  - tools/ – all feature-specific tools
  - llm_utils.py – LLM abstraction layer
  - server.py – FastAPI server for frontend connection

Alternatives Considered

While exploring the architecture and implementation of DashBot, several alternative designs were considered. These options were ultimately not selected based on scalability, modularity, or user experience trade-offs.

1. **Monolithic Backend Instead of Modular Agent**
   - **Idea:** Build a single large AI function that handles all tasks (debugging, documentation, test generation, etc.) through prompt chaining.
   - **Why Rejected:** This approach lacks modularity, makes it harder to maintain or add new tools, and limits flexibility in selecting optimal LLMs per task.

2. **Frontend-Only LLM Integration (No Backend Agent)**
   - **Idea:** Use JavaScript/Flutter packages to call OpenAI APIs directly from the frontend.

- **Why Rejected:** This exposes API keys to clients, increases latency due to poor caching/reuse, and makes it difficult to add logic-heavy tools or benchmarking systems.

3. **Model-Specific Hardcoding**
   - **Idea:** Write dedicated tool logic for a specific LLM provider (e.g., OpenAI or Gemini).
   - **Why Rejected:** Limits flexibility and defeats the purpose of benchmarking. Instead, an abstraction layer (llm_utils.py) will support pluggable backends.

4. **Client-Side Visualization via Embedded Charts**
   - **Idea:** Use only client-side JS libraries for all data visualizations.
   - **Why Rejected:** This approach limits customization and data preprocessing capabilities, which are easier to manage server-side using Python libraries like matplotlib, Plotly, or Pandas.

Related Work

DashBot follows on from current developments in developer tool and API assistant AI technology. While assistants such as Postman AI or Hoppscotch AI do offer AI-driven API exploration help, they do not have any extensibility, backend connectivity, or handling of advanced capabilities such as visualization or benchmarking multiple LLMs. And while all-around tools such as GitHub Copilot or Cursor AI are superb coding assistants, they are not specially designed for handling API workflows or debugging.

DashBot seeks to bridge this gap by providing a modular, backend-supported, API-specific AI assistant that not only assists developers in automating tasks through natural language but also enables extensible tools, visualized insight, and comparison of various language models. It is intended to be an open, community-oriented solution that is capable of adapting to the future needs of the API Dash community.

Tool / Feature	Postman AI	Hoppscotch AI	GitHub Copilot	DashBot (Proposed)
Natural Language Input	✓	✓	✓	✓
API Request Debugging	✗	✓(basic)	✗	✓(deep with LLMs)
Auto API Documentation	✓	✓	✗	✓
Test Case Generation	✗	✗	✗	✓
API Response Visualization	✗	✗	✗	✓(customizable)
Frontend Integration Code Gen	✗	✗	✓(generic)	✓(React, Flutter)
Pluggable LLM Support	✗	✗	✗	✓
LLM Benchmarking Suit	✗	✗	✗	✓
Open Source & Extensible	✗	✓	✗	✓

Pre proposal Work

Starter Issues Explored

As part of my pre-proposal preparation, I have actively explored the API Dash codebase and contributed to discussions around Issue #621, which focuses on the development of DashBot, the AI assistant for API Dash. This issue outlines the vision of creating a modular, natural language-driven tool that automates API-related workflows, and it helped me align my approach with the project's goals.

In addition, I reviewed and interacted with other relevant starter issues and features to understand how API requests, testing, and visualization are currently handled in the project. I am also in the process of contributing by raising PRs related to codebase improvements and testing to become more familiar with the architecture and get feedback from maintainers.

Understanding of the Project

DashBot is envisioned as a developer-focused AI agent integrated into API Dash that allows users to interact with their APIs using natural language. From explaining responses and debugging errors to auto-generating documentation, test cases, and frontend code, the assistant will bring intelligence and automation directly into the API workflow. Its modular design will support plugging in multiple LLMs, along with a benchmarking framework to evaluate their performance on different tasks. The goal is to empower developers with fast, context-aware suggestions and improve their overall experience when building, testing, and integrating APIs.

Potenrial Risks & Mitigation Strategies

Risk	Impact	Mitigation Strategy
LLM Integration Challenges	Medium	Use modular design to allow easy swapping of models (e.g., OpenAI, LLaMA). Start with proven APIs to ensure baseline functionality.
API Response Variability	Medium	Use a robust schema parsing strategy and fallback handling for unexpected API structures. Validate responses before processing.
Performance Bottlenecks	Medium	Optimize with async handling, caching intermediate results, and quantized LLMs where applicable.

Visualization Complexity	Low	Start with basic plotting (e.g., matplotlib, Plotly) and extend features iteratively.
Frontend Code Generation Accuracy	High	Fine-tune prompts, implement user-editable templates, and validate outputs via tests.
LLM Benchmarking Inconsistency	Medium	Define standard test cases and metrics upfront. Automate benchmarks and use reproducible environments.
Time Management	Medium	Use milestone-based planning, weekly goals, and continuous feedback loops with mentors to stay on track.

Schedule of Deliverables (timeline)

May 8 - June 1 (Community Bonding Period)

- Engage with mentors on Discord.
- Study API Dash's codebase and architecture.
- Finalize feature set and development milestones.
- Set up local development and testing environment.

June 2 - July 18 (Phase I)

- Develop DashBot core with modular architecture.
- Implement debugging and response discrepancy detection.
- Start work on API documentation generation.
- Initial benchmarking setup for LLM evaluation.
- Mid-term evaluation and feedback.

July 19 - Sept 1 (Phase II)

- Complete API documentation generator.
- Implement automated API testing feature.
- Develop API response visualization tool.
- Benchmark and test different LLMs.

Sept 1 - November 9 (For Extended Timelines)

- Implement frontend code generator.
- Fine-tune performance and optimize API calls.
- Finalize LLM benchmarking results.
- Complete documentation and prepare final report.
- Submit PR for review and integrate DashBot into API Dash.

Expected Outcome

By the end of the GSoC period, DashBot will be an AI-powered assistant seamlessly integrated into API Dash, automating API-related tasks and offering extensive benchmarking capabilities for LLMs. The project will be well-documented, tested, and optimized for future improvements.

Future Work

- Extend support for more frontend frameworks.
- Improve benchmarking with additional evaluation metrics.
- Enhance LLM adaptability for specific API use cases.
- Explore integration with OpenAPI/Swagger.

Communications

Clear and consistent communication is key to a successful GSoC project. I am fully committed to dedicating 40 hours per week to this project and will be available throughout the program duration. My preferred mode of communication is via the official API Dash Discord server, where I will stay active in the #gsoc-foss-apidash channel. I'll provide regular updates on progress, blockers, and decisions through GitHub Issues, PRs, and weekly check-ins with my mentor.
In case of any unforeseen delays or challenges, I will proactively communicate the situation, adjust my timelines, and re-prioritize tasks accordingly to stay on track. I also plan to maintain a shared progress tracker (via Notion or GitHub Projects) for full transparency. My goal is to stay responsive, receptive to feedback, and maintain a smooth and productive collaboration throughout the program.

- **Timezone / Working Hours:** IST (UTC+5:30), available daily from 10:00 AM to 6:00 PM.
- **Email:** mrityunjaykukreti@gmail.com
- **Phone:** +91-8929633669
- **Discord ID:** Weeebhu#5618

About Me

I'm Mrityunjay, a Computer Science graduate from Maharshi Dayanand University (MDU). I'm passionate about building intelligent systems that solve real-world problems, and over the past year, I've focused deeply on Artificial Intelligence, Large Language Models (LLMs), and agent-based systems.

I have hands-on experience with technologies like Python, PyTorch, TensorFlow, LangChain, and FastAPI. I've worked on various AI projects, including an AI-powered health assistant and LLM-based tools for cold emailing and research assistance. I'm also skilled in full-stack development (MERN stack), API integration, and MLOps. Open-source has played a huge role in my learning journey, and contributing to the API Dash ecosystem through DashBot excites me because it combines AI, automation, and developer tooling—three things I'm truly passionate about. Apart from technology, I like to mentor colleagues, experiment with developer tools, and stay current on developments in generative AI. I look forward to developing through this GSoC opportunity while contributing meaningfully to the community.

Here's my resume: Mrityunjay's Resume

Prior Experience with open source

Whereas the majority of my work has been on private repositories and proprietary AI projects, I have indeed worked with teams to develop end-to-end applications using LLMs, APIs, and AI automation. All these experiences exposed me to very good working environments with modular codebases, ensuring clean and testable code, feature branch management, and engagement in code review.

One of my publicly available projects is:

- **AI-Powered Health Assistant**
  GitHub Repository
  A smart chatbot that delivers context-aware health suggestions by interpreting medical data from pdf files using LLMs and prompt engineering.

Although my contributions to open source have been limited in public visibility, I'm eager to deepen my involvement through GSoC by making impactful and transparent contributions to API Dash.

Why API Dash?

I chose API Dash for Google Summer of Code because it aligns perfectly with my passion for developer tools, AI integration, and automation. API Dash offers a unique opportunity to work at the intersection of natural language processing, developer experience, and API management—all areas where I've been actively building and experimenting.

What I'm most excited about is the vision of DashBot: to enable developers to work more effectively with the help of AI-powered assistants. I envision this as an applied use case of LLMs, where their potential can dramatically cut

down the time and mental effort involved in handling APIs. This mission deeply aligns with my interests and previous work in creating AI agents and API utilities.

Working with API Dash will give me invaluable mentorship and a collaborative environment to polish my software engineering practices, contribute to open source in a meaningful way, and grow as a builder of developer-focused AI tools. I'm confident that this project will not only allow me to learn from experienced contributors but also enable me to make a tangible, lasting impact.

Feedback from API Dash

If you read this document please provide your short general feedback in the section below. Please also feel free to make comments above.

Username	Date	Comment