Vega: LLM-driven Intelligent Chatbot Platform for Internet of Things Control and Development

Harith Al-Safi *, Harith Ibrahim, Paul Steenson

Posted Date: 7 May 2025

doi: 10.20944/preprints202505.0380.v1

Keywords: embedded systems; internet of things; large language models; natural language processing; Raspberry Pi; user interaction; web applications

 doi:10.20944/preprints202505.0380.v1

Article

Vega: LLM-driven Intelligent Chatbot Platform for Internet of Things Control and Development

Harith Al-Safi *, Harith Ibrahim and Paul Steenson

School of Electronics and Electrical Engineering, University of Leeds, Leeds LS2 9JT, U.K
* Correspondence: harith.alsafi@gmail.com

Abstract: Large language models (LLMs) have revolutionized natural language processing (NLP), yet their potential in Internet of Things (IoT) and embedded systems (ESys) applications remains largely unexplored. Traditional IoT interfaces often require specialized knowledge, creating barriers for non-technical users. We present Vega a modular system that leverages LLMs to enable intuitive, natural language control and interrogation of IoT devices, specifically a Raspberry Pi (RPi) connected to various sensors, actuators and devices. Our solution comprises three key components: a physical circuit with input and output devices, an RPi integrating a control server, and a web application integrating LLM logic. Users interact with the system through natural language, which the LLM interprets to remotely call appropriate commands for the RPi. The RPi executes these instructions on the physically connected circuit, with outcomes communicated back to the user via LLM-generated responses. The system's performance is empirically evaluated using a range of task complexities and user scenarios, demonstrating its ability to handle complex and conditional logic without additional coding on the RPi reducing the need for extensive programming on IoT devices. We showcase the system's real-world applicability through physical circuit implementation, while providing insights into its limitations and potential scalability. Our findings reveal that LLM-driven IoT control can effectively bridge the gap between complex device functionality and user-friendly interaction, and also opens new avenues for creative and intelligent IoT applications. This research offers insights into the design and implementation of LLM-integrated IoT interfaces and is showcased in the following link https://youtu.be/CKV__A8G5Rk.

Keywords: embedded systems; internet of things; large language models; natural language processing; Raspberry Pi; user interaction; web applications

1. Introduction

Large language models (LLMs) have significantly revolutionized natural language processing (NLP), demonstrating unprecedented capabilities in understanding and generating human-like text [1]. However, their potential in Internet of Things (IoT) and embedded systems (ESys) applications remains largely untapped. IoT systems have become increasingly prevalent across various domains, from smart homes to industrial automation [2]. Despite their widespread adoption and security issues notwithstanding, developing and interacting with adaptive IoT systems often requires specialized knowledge and good programming skills, creating significant barriers for new or non-technical users [3].

Traditional IoT interfaces typically rely on graphical user interfaces (GUIs) or specific programming languages, which can be challenging for users to develop without technical expertise [3]. This limitation hinders the widespread adoption and utilization of IoT technologies, particularly in scenarios where rapid deployment and intuitive user interaction are crucial. While research has been conducted on natural language interfaces for IoT, the application of advanced language models to IoT control and interaction remains an unexplored area [4].

To address these challenges, we propose **Vega**, an intelligent chatbot platform that leverages LLMs to enable intuitive, natural language control of IoT devices. Our system focuses on a Raspberry Pi (RPi)

 doi:10.20944/preprints202505.0380.v1

connected to various sensors and devices as a representative IoT setup. By integrating LLMs with IoT infrastructure, we aim to bridge the gap between complex device functionality and user-friendly interaction, allowing users to control and query IoT systems using everyday language. Our research builds upon recent advancements in LLMs, specifically OpenAI's GPT-based models [5], which utilize transformer neural network architectures to capture context and relationships within text data. By applying these powerful language understanding capabilities to IoT interaction, we aim to create a more accessible and flexible approach to device control and monitoring. Our approach, addressing the standardization challenges highlighted by Al-Qaseemi et al. [6], not only enhances accessibility for non-technical users but also opens new avenues for creative and intelligent IoT applications.

Vega's architecture comprises three key components: a physical circuit with input and output devices, an RPi integrating a control server, and a web application incorporating LLM logic. This modular design allows for flexibility and scalability, enabling the system to adapt to various IoT scenarios and user requirements [7]. By utilizing the RPi as a central hub, we can leverage its versatility and widespread adoption in the IoT community [8]. The main contributions of this paper are as follows:

1. We developed a chat web app that executes queries on the RPi which contains a control server that manages the execution on a circuit and communication with the web app.
2. We develop a multi-agent LLM framework that translates natural language commands into executable instructions for IoT devices, capable of handling complex, conditional logic without additional coding on the RPi.
3. We showcase the system's real-world applicability through physical circuit implementations and provide insights into its limitations and potential scalability.
4. We implement and evaluate the system demonstrating the feasibility and effectiveness of LLM-driven IoT control across various task complexities and user scenarios, including an evaluation mode with automated test generation and performance assessment.

The remainder of this paper is organized as follows: Section 2 provides background information and discusses related work in IoT interfaces and NLP. Section 3 details our methodology, including the overall architecture, physical circuit design, RPi configuration, and web application implementation. Section 4 presents our experimental setup, results, and analysis, showcasing the system's performance in handling complex commands and its potential real-world applications. Finally, Section 5 concludes the paper, system's limitations and insight into future research.

2. Background and Related Work

2.1. Industrial Applications of LLM's

LLM's are based on the transformer architecture [9] which uses self-attention mechanisms to analyse large sequences of text data, have been effectively applied across diverse domains, including robotics, software and IoT applications.

Maddigan and Susnjak [10] showcased this versatility with Chat2VIS, leveraging ChatGPT and GPT-3 to generate data visualizations from natural language queries. Their innovative approach demonstrated how LLM's could be effectively used to convert free-form natural language directly into visualization code, even when queries were highly underspecified. Meanwhile, Vemprala et al. [11] explored the application of LLM's, specifically OpenAI's ChatGPT, in robotics applications. Their research presented a strategy combining prompt engineering principles and a high-level function library, enabling ChatGPT to adapt to various robotics tasks, simulators, and form factors. The study evaluated different prompt engineering techniques and dialogue strategies for executing robotics tasks, ranging from basic logical and geometrical reasoning to complex domains like aerial navigation and manipulation.

Recent research has explored the integration of LLMs with robotic systems, paving the way for intuitive human-robot interaction. Singh et al. [12] introduced ProgPrompt, a novel approach leveraging LLMs to generate action sequences based on natural language instructions. By prompting

 doi:10.20944/preprints202505.0380.v1

LLMs with program-like specifications of available actions and objects, along with example programs, their method enables plan generation across diverse environments, robot capabilities, and tasks. This work demonstrated state-of-the-art success rates in virtual household tasks and was successfully deployed on a physical robot arm for tabletop tasks.

Expanding on this concept, Driess et al. [13] proposed PaLM-E, an embodied multimodal language model that incorporates real-world sensor data into language models. PaLM-E is trained on tasks such as robotic manipulation planning and visual question answering, exhibiting positive transfer across language, vision, and visual-language domains. This research highlights the potential of LLMs in grounding language understanding in physical environments, a crucial aspect for IoT applications. Grounding connects language to real-world objects and actions; in Vega, it links user commands to IoT device operations, enabling intuitive control.

In the context of multi-agent systems, which involve multiple autonomous agents collaborating to achieve common goals, Kannan et al. [14] developed SMART-LLM, a framework for embodied multi-robot task planning. This approach uses LLMs to convert high-level task instructions into multi-robot task plans through a series of stages, including task decomposition, coalition formation, and task allocation. The authors created a benchmark dataset for validating multi-robot task planning problems, demonstrating the framework's effectiveness in both simulated and real-world scenarios. Similarly, in the case of Vega, it utilizes multiple agents to handle different scenarios such as task planning, image processing, and chat interaction.

Wu et al. [15] presented TidyBot, a system that combines language-based planning and perception with LLMs to infer generalized user preferences for household clean-up tasks. This research demonstrates the potential of LLMs in personalizing robot assistance, achieving 91.2% accuracy on unseen objects in their benchmark dataset and successfully putting away 85.0% of objects in real-world test scenarios.

While these advancements primarily focus on robotics, they lay a solid foundation for extending similar techniques to IoT scenarios. The ability to interpret natural language instructions, generate action sequences, and integrate multimodal sensor data holds significant potential for enabling intuitive and intelligent control of IoT devices and systems. As research progresses, we anticipate further innovations in LLM-driven IoT interfaces, potentially revolutionizing how users interact with smart environments.

2.2. Natural Language Processing for IoT

NLP has emerged as a transformative technology in IoT applications, enabling intuitive human-machine interactions. The integration of NLP in IoT systems allows users to instruct, control and query devices using everyday language, bridging the gap between complex technological interfaces and user-friendly experiences [16]. This integration is particularly crucial as IoT devices become ubiquitous in various domains, from smart homes to industrial settings, where ease of use and accessibility are paramount.

Recent research has demonstrated the potential of NLP in IoT contexts. For instance, Petrović et al. [17] explored the use of ChatGPT in IoT systems, focusing on Arduino-based applications. Their work highlighted the possibilities of leveraging LLMs for both question-answering and automated code generation in IoT environments. Similarly, Zhong et al. [18] proposed CASIT, a collective intelligent agent system for IoT that utilizes LLMs to process and interpret data from multiple sources efficiently. These studies underscore the growing interest in applying advanced NLP techniques to enhance IoT functionality, operability and user experience.

The integration of LLMs represents a very significant advancement in NLP capabilities for IoT. Traditional NLP methods often struggle with context understanding and complex query interpretation, limitations that LLMs can overcome. LLMs offer improved natural language understanding, enabling more nuanced and context-aware interactions with IoT devices. For example, King et al. [16] demonstrated how LLMs can interpret ill-defined and under-specified commands in smart home environments, translating vague user intentions into specific device actions.

 doi:10.20944/preprints202505.0380.v1

The potential of LLMs in IoT extends far beyond simple command interpretation. They can enable more sophisticated applications such as predictive maintenance, anomaly detection, and personalized user experiences. Sarzaeim et al. [19] explored the use of LLMs in smart policing systems, showcasing their potential in complex data analysis and pattern recognition. This application hints at the broader possibilities of LLMs in IoT, where they could be used to analyse and interpret vast amounts of sensor data, making IoT systems more intelligent and proactive.

However, integrating LLMs into IoT systems also presents challenges, including privacy concerns, computational requirements, and the need for domain-specific training. Despite these challenges, the potential benefits of LLM-enhanced NLP in IoT are significant. As demonstrated by Xu et al. [20], natural language interfaces can greatly improve the usability of IoT platforms, allowing for more complex and nuanced interactions. By leveraging the advanced capabilities of LLMs, future IoT systems could offer unprecedented levels of intuitive control and intelligent automation, paving the way for more accessible and powerful IoT applications across various domains.

2.3. Language Oriented Architectures

Chatbots have gained significant traction across various industries, serving as direct communication channels between companies and end-users [21]. However, existing frameworks often require advanced technical knowledge for complex interactions and lack flexibility in adapting to evolving company requirements. The deployment of chatbot applications typically demands a deep understanding of targeted platforms, particularly back-end connections, which increases development and maintenance costs [21].

To address these challenges, researchers have proposed novel approaches to chatbot development. Xatkit, for instance, offers a set of Domain Specific Languages to define chatbots in a platform-independent manner, along with a runtime engine for automatic deployment and conversation management [21]. Similarly, Jiang et al. [22] propose a multi-agent system enhanced by LLM's for 6G communications allowing users to input task requirements, while addressing challenges such as limited communication knowledge through a combination of specialized agents for data retrieval, collaborative planning, evaluation and reflection. Recent studies have explored multi-modal chatbots in intelligent manufacturing settings, demonstrating the potential for AI-powered dialogue systems to assist users in complex assembly tasks [23]. These systems leverage both textual and visual capabilities to improve intent classification and provide relevant information to users. The development of conversation-driven approaches for chatbot management has also shown promise in evolving chatbot content through the analysis of user interactions, allowing for a cyclic and human-supervised process [24].

In the realm of human-robot interaction, researchers have developed task-oriented dialogue systems for industrial robots, addressing the lack of domain-specific discourse datasets and emphasizing user experience alongside task completion rates [25]. These efforts have resulted in datasets like IRWoZ and frameworks such as ToD4IR, which integrate small talk concepts and human-to-human conversation strategies to support more natural and adaptable dialogue environments. The potential of LLMs in easing IoT oriented chatbot development has been demonstrated through large-scale models that can learn blended conversational skills when provided with appropriate training data and generation strategies [26]. These models have shown improvements in multi-turn dialogue engagement and human related measurements. Vega utilizes these frontiers within its multi-agent intelligent chatbot allowing it to interact with any IoT system and handle complex queries while maintaining a user-friendly interaction.

3. Methodology

3.1. Overall Architecture

The architecture of the Vega system follows key principles of software design to ensure clarity, scalability, maintainability, and robustness [7]. The system adopts a modular approach, dividing func-

 doi:10.20944/preprints202505.0380.v1

tionality into distinct components with specific purposes. This design promotes code reuse, facilitates testing, and enhances overall maintenance. The architecture also implements separation of concerns, where different aspects such as user interface (UI), core functionality, and data management are segregated into distinct layers, improving code organization and enabling independent development.

As shown in Figure 1a, Vega's architecture comprises three main modules: a web application, an RPi, and a physical circuit. These modules interact in a client-server model [27], with the web application serving as the client and the RPi as the server. The physical circuit is connected to the RPi via hardwired connections.

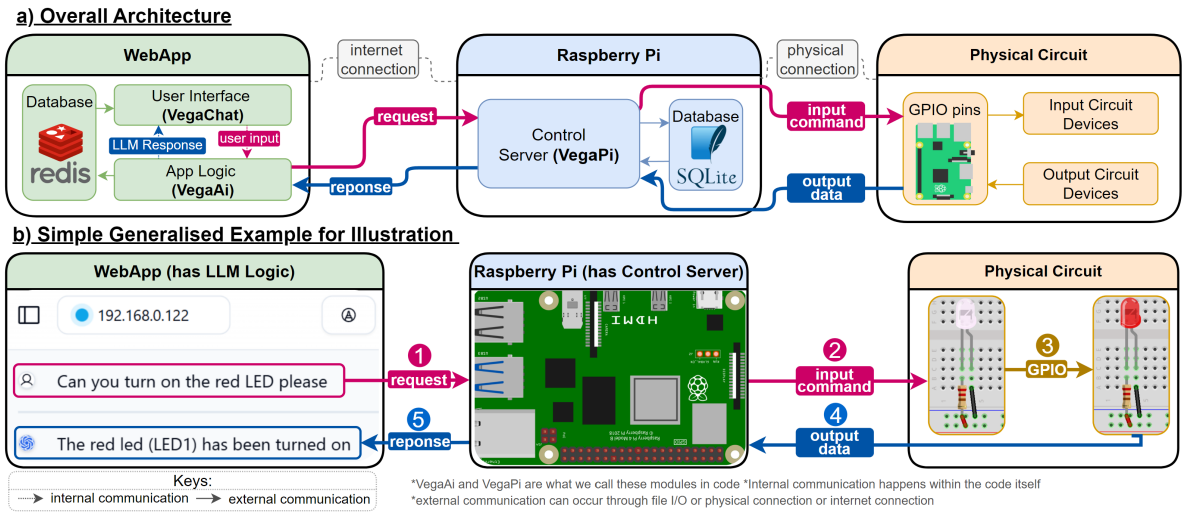

Figure 1. Overall technical architecture of Vega (a) and alongside (b) a simple generalized example.

The web application consists of two primary submodules: the user interface (VegaChat) and the app logic (VegaAi). The app logic incorporates LLM logic containing the multi-agents for translating user input into commands and generating responses. Redis [28] is employed as a non-relational database to store chat history, messages, and RPi connection states.

The RPi module hosts a control server (VegaPi) responsible for parsing requests from the app logic and executing them on the physical circuit. An SQLite database [29] is used to store data extracted from the physical circuit. The physical circuit comprises input devices (sensors) and output devices (motors, displays, etc.) connected to the RPi's General Purpose Input/Output (GPIO) pins.

A typical use case shown in Figure 1b involves a user interacting with the web application interface, sending a natural language command to turn on a red light emitting diode (LED). The LLM interprets this command and sends the appropriate instruction to the RPi's control server. The server then relays the command to the physical circuit via GPIO pins which turns on the LED. Upon execution, the circuit sends feedback to the RPi, which is then communicated back to the user through the web application and the LLM.

The technology stack for Vega has been carefully selected to ensure robustness, scalability, and accessibility [30]. The web application is built using React [31] with TypeScript, employing RadixUI [32] for accessible components and TailwindCSS [33] for responsive design. The App Logic utilizes NextJs [34] and integrates with OpenAI's GPT models [5] for language processing. The RPi control server is developed using Flask [35], while the circuit code leverages RPi libraries for GPIO interaction.

This architecture enables Vega to bridge the gap between complex IoT functionality and user-friendly interaction. By leveraging LLMs for NLP and control, the system opens up new possibilities for intuitive IoT applications in various domains, from smart homes to industrial monitoring and educational environments [8]. The modular design and carefully chosen technology stack ensure that Vega remains adaptable, maintainable, and scalable as IoT applications continue to evolve and expand.

 doi:10.20944/preprints202505.0380.v1

3.2. Physical Circuit Design

The physical implementation of the Vega platform comprises a custom-designed circuit board that interfaces with the RPi, integrating various input and output devices to facilitate IoT and embedded systems applications. This hardware configuration forms the foundation for the natural language-controlled system, enabling users to interact with commonly used physical components through LLM-interpreted commands.

As shown in Figure 2, the circuit board incorporates a diverse array of input devices, including an ultrasonic sensor for distance measurement, a limit switch for binary state detection, a temperature and humidity sensor for environmental monitoring, a GPS module for location tracking, and a push button for direct user input [36]. These components collectively provide a rich set of data sources, enabling the system to respond to complex, context-aware queries and commands.

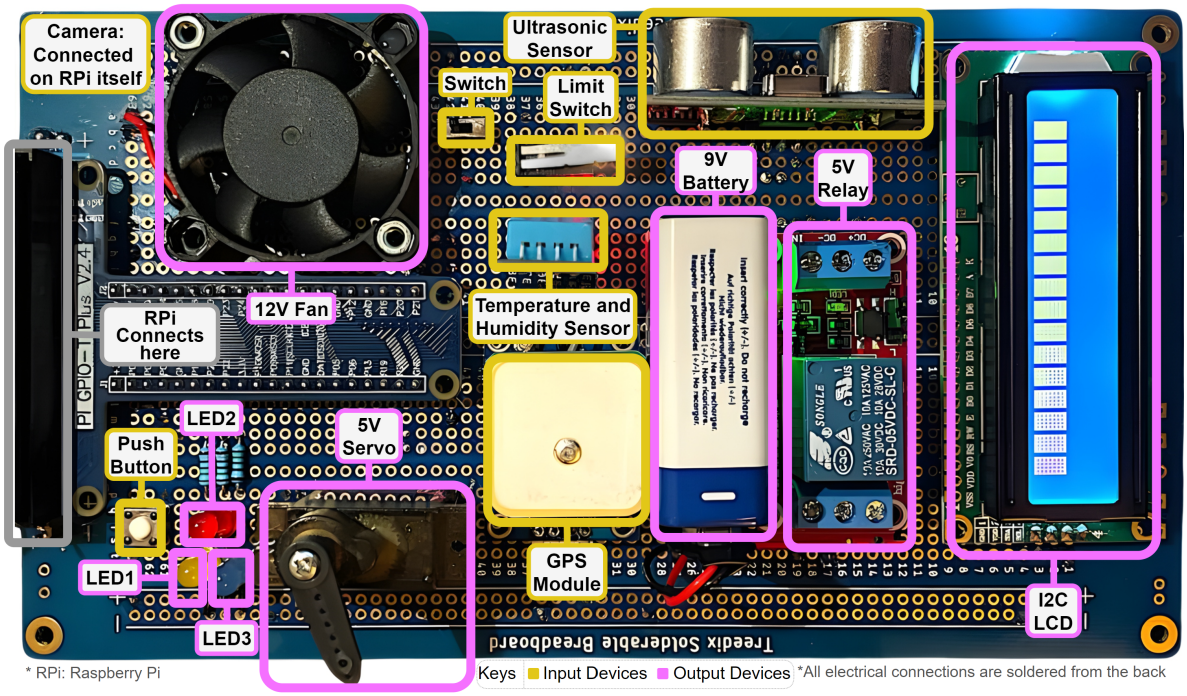

Figure 2. Soldered physical circuit connected to the RPi.

Output devices on the board include a 12V fan for cooling or air circulation, multiple LEDs (yellow, red, and blue) for visual indicators, a 5V servo motor for precise rotational control, and a liquid crystal display (LCD) for text output. A 5V relay is incorporated to control the 12V fan, demonstrating the system's capability to manage higher-voltage components safely [37]. The inclusion of these diverse output devices allows for a wide range of physical responses to user commands, from simple visual feedback to more complex mechanical actions.

Power management is a crucial aspect of the circuit design. While most components operate on the 5V supply provided by the RPi, the 12V fan requires a separate power source. To address this, a 9V battery is utilized in conjunction with the relay, ensuring proper voltage supply while maintaining RPi-based control. This setup illustrates the system's ability to accommodate components with varying power requirements within a unified control structure.

The circuit board is designed to connect directly to the RPi's GPIO pins, streamlining the interface between the physical components and the computational core of the system. A camera module, while not physically present on the circuit board, is connected directly to the RPi, expanding the system's capabilities to include image capture and analysis [38].

This hardware configuration supports a wide range of potential applications. In smart home scenarios, the temperature sensor and fan could be used for automated climate control, while the GPS module could enable location-based automation in mobile or outdoor settings. In industrial

 doi:10.20944/preprints202505.0380.v1

environments, the ultrasonic sensor and limit switch could be employed for proximity detection and safety systems, with the LEDs and LCD providing status information to operators [37].

The versatility of this hardware setup, combined with the LLM-driven control system, enables the exploration of complex, conditional logic without requiring additional RPi-level coding. This integration of diverse sensors and actuators with NLP capabilities represents a significant step forward in creating intuitive, user-friendly interfaces for IoT and embedded systems, bridging the gap between sophisticated device functionality and accessible user interaction.

3.3. Raspberry Pi Design

The architecture of the RPi integration with the existing codebase is designed to enable seamless control and manipulation of the circuit without interfering with pre-existing logic. This approach leverages parallel computing concepts, utilizing processor cores and threads to execute specific logic concurrently with existing code [39].

The system architecture, illustrated in Figure 3, comprises two main threads: the control server thread and the database thread. The control server thread manages a Flask-based web framework, storing predefined functions for a set of circuit devices. These functions are exposed through a REST API, facilitating communication between the web app and the RPi over the internet. The database thread retrieves sensor data at two-second intervals, storing it in an SQLite database. This persistent storage solution ensures data preservation in the event of system failures, enabling data recovery, analytics, and statistical analysis. The stored data can be retrieved upon request and provided to the LLMs in the web application, enhancing system monitoring and diagnostic capabilities.

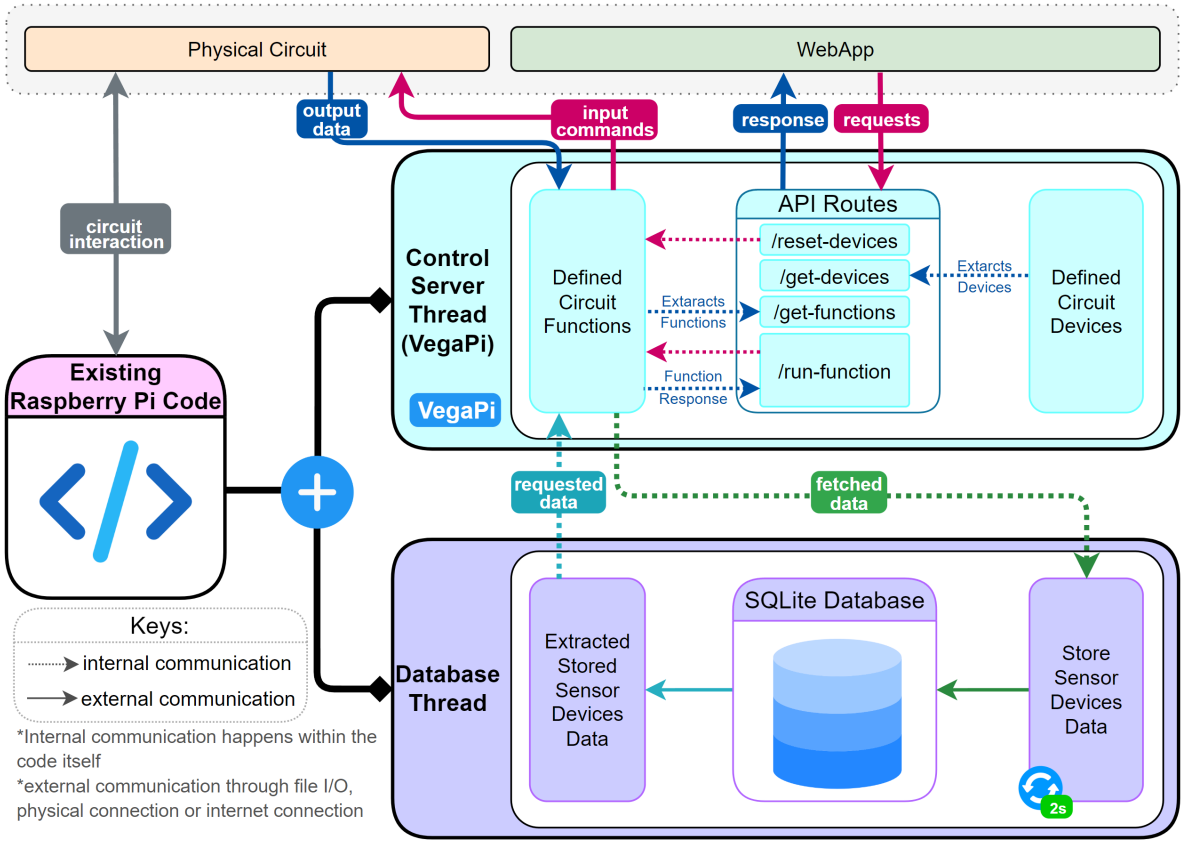

Figure 3. Architecture code design of the RPi including the control server and the database.

Table 1 presents the devices defined in the control server, categorized as inputs or outputs. This information is stored and transmitted in JSON format via the "get-devices" REST API endpoint. Input devices primarily transmit data for database storage, while output devices receive commands for circuit manipulation.

 doi:10.20944/preprints202505.0380.v1

Table 1. Physical devices defined on the control server, which are then utilized by the LLM to interact with the circuit.

Symbol	Pin Type	Description
ULTS	Input	Ultrasonic distance sensor in 'cm'
CAM	Input	Camera device for image acquisition
GPS	Input	GPS device for longitude and latitude coordinates
TMP	Input	Temperature sensor in degrees celsius
FAN	Output	12V fan controlled through digital GPIO in relay
LCD	Output	LCD for displaying text data
SRV	Output	Servo motor rotates to given set of angles
LED1	Output	Yellow LED light
LED2	Output	Red LED light
LED3	Output	Blue LED light

The control server exposes a set of defined functions, listed in Table 2, which the LLM utilizes to determine logic and execute commands on circuit components. These functions are accessible to the LLM through the "get-functions" REST API endpoint. To execute a particular function, the LLM passes the function identifier and required parameters to the web application logic, which then invokes the "run-function" API endpoint.

Table 2. Defined functions on the control server, called by the LLM based on user input, executes on the RPi and processed on the web app.

Function	Description	Use Case
set_led	Toggles specific LED	"Turn on yellow LED"
set_fan	Toggles fan on or off	"Turn on the fan"
get_recorded_sensor_data	Gets interval sensor data from database	"Plot me the distance data in last 30 seconds"
get_raspberry_stats	Gets CPU, RAM, disk of RPi	"What is the current disk usage"
capture_image	Capture and upload image to the cloud	"Capture an image, does it contain a pen?"
get_connected_devices	Fetches data of connected devices	"What is the current humidity and temperature"
get_location_	Gets the current location from GPS	"From the location are we currently in Leeds?"
set_servo_angles	Turn servo to certain angle	"Turn the servo to 10 then 180 degrees"

The choice of REST API over alternative protocols such as MQTT was based on several factors. REST offers simplicity, scalability, and statelessness, making it well-suited for web-based applications [40]. It also provides a uniform interface, enabling easier integration with various client applications. While MQTT excels in low-bandwidth, high-latency environments, the current system architecture prioritizes the flexibility and widespread support offered by REST APIs in web development ecosystems.

The communication flow between the web application and the RPi follows a request-response pattern. The web application sends REST API requests with JSON data specifying the function and arguments for the RPi to execute. The RPi processes these requests, executes the specified functions, and returns JSON responses with the execution status to the web application. This bidirectional communication enables real-time control and monitoring of the IoT devices.

This architecture facilitates a modular and extensible system, allowing for easy addition of new devices and functions. It also provides a layer of abstraction between the physical hardware and the LLM-driven interface, enabling natural language control of IoT devices without requiring users to understand the underlying technical details. The integration of LLMs with this IoT control

 doi:10.20944/preprints202505.0380.v1

system represents a significant step towards more intuitive and accessible IoT interfaces, potentially broadening the application of IoT technologies across various domains [7].

3.4. Web App User Interface

The web application forms the core of the Vega platform, initiating all LLM processing and circuit manipulation tasks. Its architecture is modular, separating the User Interface (VegaChat) from the App Logic (VegaAi). The User Interface shown in Figure 4 comprises a Top Bar with RPi connection management and configuration options, and a Chat UI displaying LLM responses and user messages. A Chat History UI manages previous interactions. The App Logic includes Data Management, RPi Bridge, and LLM Agents components, handling data entities, RPi communication, and LLM processing respectively.

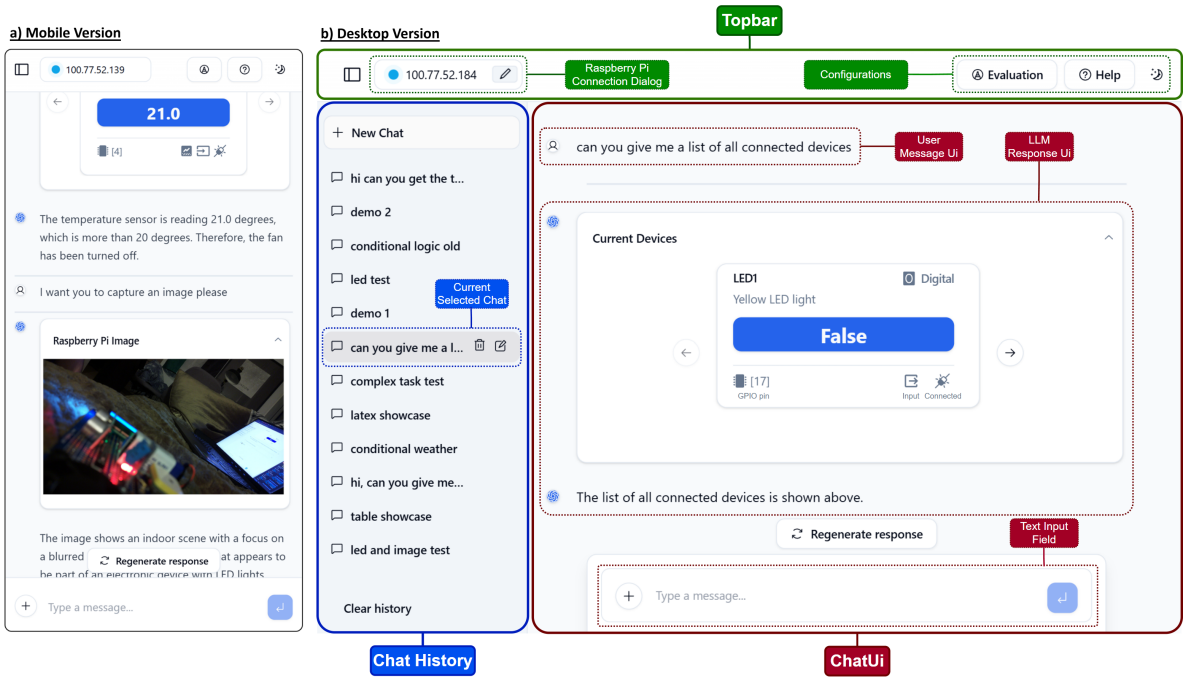

Figure 4. Web app user interface implementation showcasing (a) Mobile Version alongside (b) Desktop Version.

The UI design prioritizes usability, drawing inspiration from established chatbot interfaces [5]. It features a sidebar for chat history, a top bar for configurations, and a main chat area. An automated mode facilitates efficient testing for advanced users while maintaining simplicity for novices. The interface incorporates a Markdown renderer to appropriately display formatted text generated by the LLM.

The platform supports various data types and formats to enhance user interaction. It can display GPS data as maps, sensor readings as plots, and camera module output as images. Additionally, it visualizes LLM-generated plans as flow charts. This versatility allows the interface to accommodate diverse IoT devices and sensors, presenting their data in intuitive, visual formats.

3.5. Web App Logic

The Application Logic component serves as the core operational engine of the Vega system, managing communication with the RPi control server and integrating LLM's for NLP and command interpretation. This component acts as a bridge between external elements, orchestrating the flow of information and translating user input into appropriate actions within the system architecture [7].

By leveraging OpenAI's API, the application accesses LLM capabilities without the substantial computational overhead of local hosting [41]. This design choice simplifies and enhances the platform's accessibility and scalability, enabling its deployment across a wide range of devices, platforms and use cases in IoT and embedded systems development.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 May 2025 doi:10.20944/preprints202505.0380.v1

To establish a connection between the web application and the RPi, the user provides the IP address and port number of the RPi running the control server. The web application then initiates concurrent API calls to fetch circuit functions and device information from the control server. Upon receiving responses, the connection state is updated, synchronizing the "Raspi Devices" and "Raspi Functions" states which are fed to the LLM.

Figure 5 illustrates the system's logic, demonstrating how user commands are processed through various stages involving three main LLM agents: Chat, Planning, and Image. When a user inputs a command (e.g., "turn on red LED and capture image"), the system follows these steps:

1. The Stateless LLM Planning Agent generates a plan which is visualized as a flowchart in the user interface.
2. The Stateless LLM Chat Agent processes the message and determines if a function call to the RPi is necessary.
3. If required, the function is sent and executed on the RPi, which returns a response.
4. For image data, the Stateless LLM Image Agent analyse and generates a description used by the LLM Chat Agent to execute subsequent functions and logic.
5. Results are displayed on the web application's UI, providing feedback to the user.

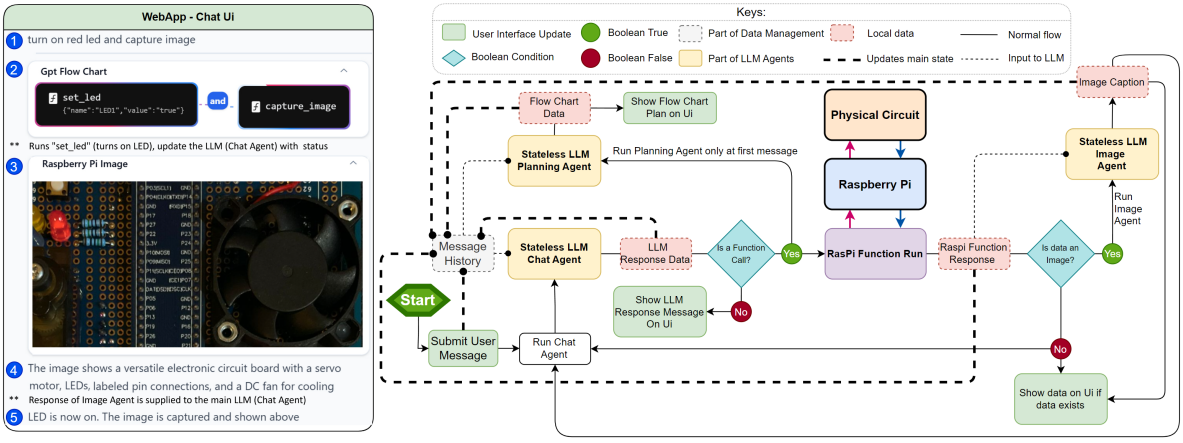

Figure 5. Web app logic in action featuring LLM agents, application states and RPi connection.

The LLM Chat Agent, as depicted in Figure 6, operates in two scenarios. In a normal chat scenario, it processes user input and generates a textual response. In a function call (provided by OpenAI) scenario, it recognizes the need for a hardware action and outputs a JSON-formatted function call for the RPi. The call is triggered using context in the user input, such as "turn on" or "capture". All other agents mentioned earlier work in a similar manner.

This approach of using LLM function call for command interpretation and execution offers several advantages over dynamic code generation methods. It enhances scalability and adaptability, allowing for easy integration of new sensors and data types without significant system modifications [42]. Additionally, it improves system security and maintainability by limiting direct code execution on the RPi, instead relying on predefined functions interpreted by the LLM which prevents dynamic errors.

 doi:10.20944/preprints202505.0380.v1

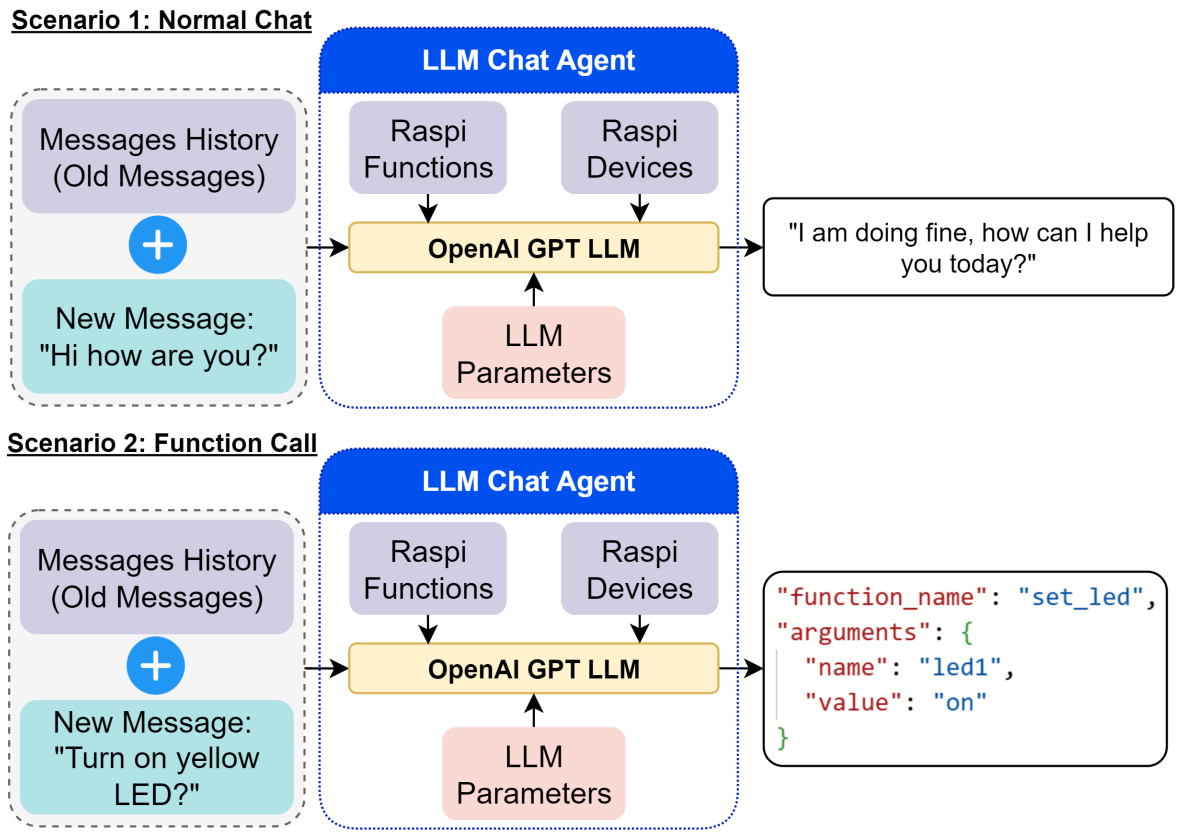

Figure 6. Execution of the LLM Chat Agent.

4. Experiment and Validation

4.1. Representative Real life case study

The experimental phase of this study focuses on demonstrating the system's versatility and real-world applicability through a few typical case studies. Figure 7 illustrates two scenarios where our LLM-powered IoT control system effectively manages complex tasks through natural language interaction. In the first scenario, the system monitors environmental conditions in a typical industrial setting. A user command prompts the system to check temperature and capture an image if it exceeds 20 °C. The LLM interprets this request, triggering the appropriate sensor readings and image capture. When the temperature reaches 27.0 °C, the system captures an image and analyses it for the presence of neon lighting. Upon detecting a neon sign, it displays "hi everyone" on the LCD, and importantly showcasing the system's ability to execute conditional logic based on sensor data and image analysis. The second case demonstrates the system's capability in a more complex scenario involving geolocation, user input tracking, and multiple actuator controls. The LLM interprets a command to check location, monitor button clicks, and adjust servo motors accordingly. This case highlights the system's ability to integrate various data sources and control multiple IoT devices simultaneously.

Contrary to the perception that LLMs require complex prompts, our system demonstrates their superior user-friendliness and responsiveness compared to traditional NLP methods. Unlike rule-based systems needing specific commands, LLMs can interpret a wide range of phrasings and even incomplete instructions. For example, a user might say, "It's a bit chilly in here," and the LLM can infer the need to adjust the thermostat. This flexibility eliminates the need to memorize commands or navigate complex menus. Moreover, LLMs handle follow-up questions and maintain context across interactions, creating a more conversational user experience. Their ability to generalize from training data allows them to handle novel requests without explicit programming, making the system more adaptable [1]. Considering human factors in software engineering [43], our system's natural language interface lowers the barrier for non-technical users, potentially democratizing access to IoT technology.

 doi:10.20944/preprints202505.0380.v1

However, as Kim et al. [44] observed, such systems may elevate user expectations for sophisticated interactions, underscoring the need for careful UI design.

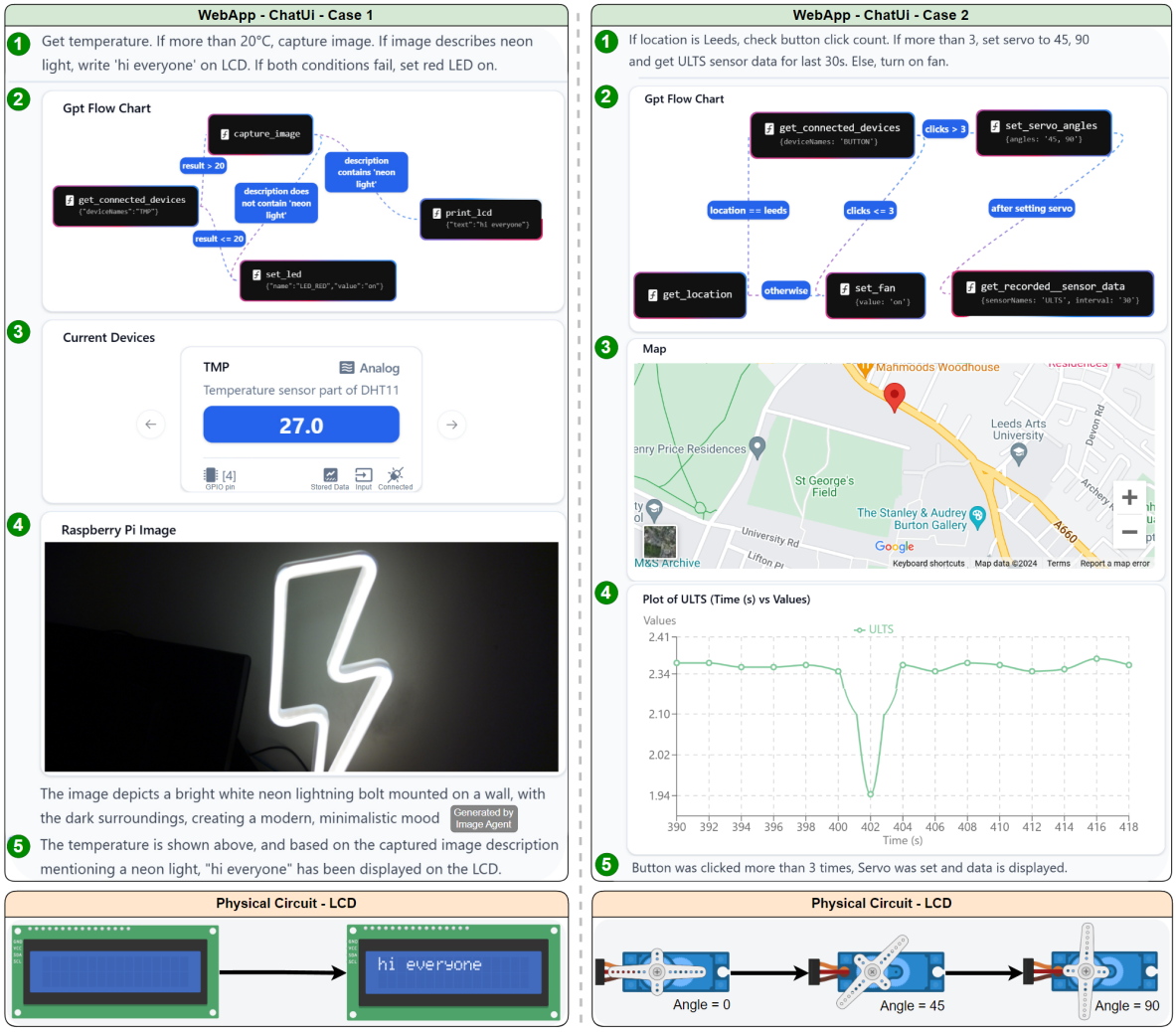

Figure 7. Case study of complex commands in action.

The scalability of our system allows for expansion to different platforms and IoT ecosystems. Future iterations could incorporate efficiency metrics to optimize LLM output, reducing computational requirements and environmental impact. The system's design has the potential to be utilized across various sectors other than IoT. For example, in robotics applications, the system can function as a tool for generating and executing complex tasks, aligning with the capabilities of LLMs in robotics explored by Vemprala et al. [11]. The modularity of the system allows easy extension of functionality, whether through new data visualizations in the web application or specialized functions in the control server. The system's design prioritizes security by avoiding runtime code generation, addressing potential vulnerabilities often associated with LLM applications in robotics [11]. This approach promotes trust and responsible automation practices, crucial for widespread adoption.

4.2. Automated Evaluation

To rigorously assess the performance and robustness of the developed system, an automated evaluation process was implemented. While manual testing provides basic insights, the complexity of the system necessitates a comprehensive automated approach [45]. This method enables the execution of numerous test cases, facilitating a thorough examination of the system's behaviour under various configurations and scenarios.

 doi:10.20944/preprints202505.0380.v1

The automated evaluation process, illustrated in Figure 8, involves supplying a predefined list of user messages to the application. An LLM agent, designated as the Test Generator, generated approximately 622 test messages. Each message had a complexity ranging from 0 to 1 in which 1 indicates a multi-step conditional message. Concurrently, the input parameters of the Chat Agent LLM model such temperature and Top P [46] were varied. The evaluation process mirrored the standard execution flow of the application, with a notable deviation occurring post-message processing on the circuit. At this point, the entire chat conversation was transmitted to an LLM Evaluation Agent, tasked with assessing the Chat Agent LLM's response.

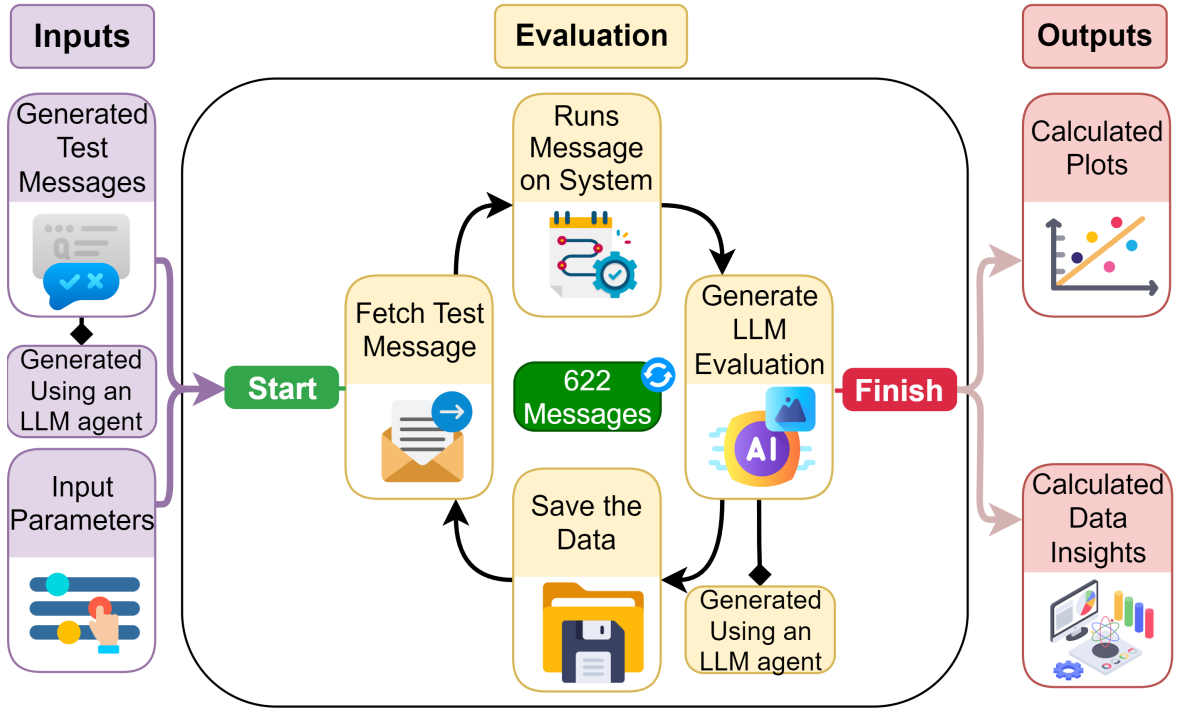

Figure 8. Evaluation Process integrated within Vega.

This approach allowed for a nuanced understanding of the system's performance under different conditions, providing insights into potential areas for optimization. The performance of the Chat Agent LLM was evaluated quantitatively based on multiple metrics, each measured on a scale from 0 to 100. The speed metric assessed response generation time, while the efficiency metric measured the degree to which the LLM avoided invoking unnecessary functions. The success rate indicated the overall rate at which the LLM successfully executed the requested action specified in the input message prompt.

4.3. Result Analysis

Figure 9 illustrates the success rate and speed for each function defined in Table 2, with the temperature parameter set to 0.7. The "set_fan" command exhibited the lowest success rate, likely due to the relay's inherent switching delay causing errors when processing frequent state change requests. This hardware limitation may lead to errors after function execution.

Conversely, the "print_lcd" command achieved the highest success rate, demonstrating the LLM's proficiency with textual arguments. The "set_led" function demonstrated the highest execution speed, attributable to its simplicity, minimal LLM processing requirements, efficient software GPIO port toggling, and basic LED control hardware. In contrast, the "get_recorded_sensor_data" function exhibited the lowest speed, primarily due to performance limitations of Python when retrieving and processing data from the database.

 doi:10.20944/preprints202505.0380.v1

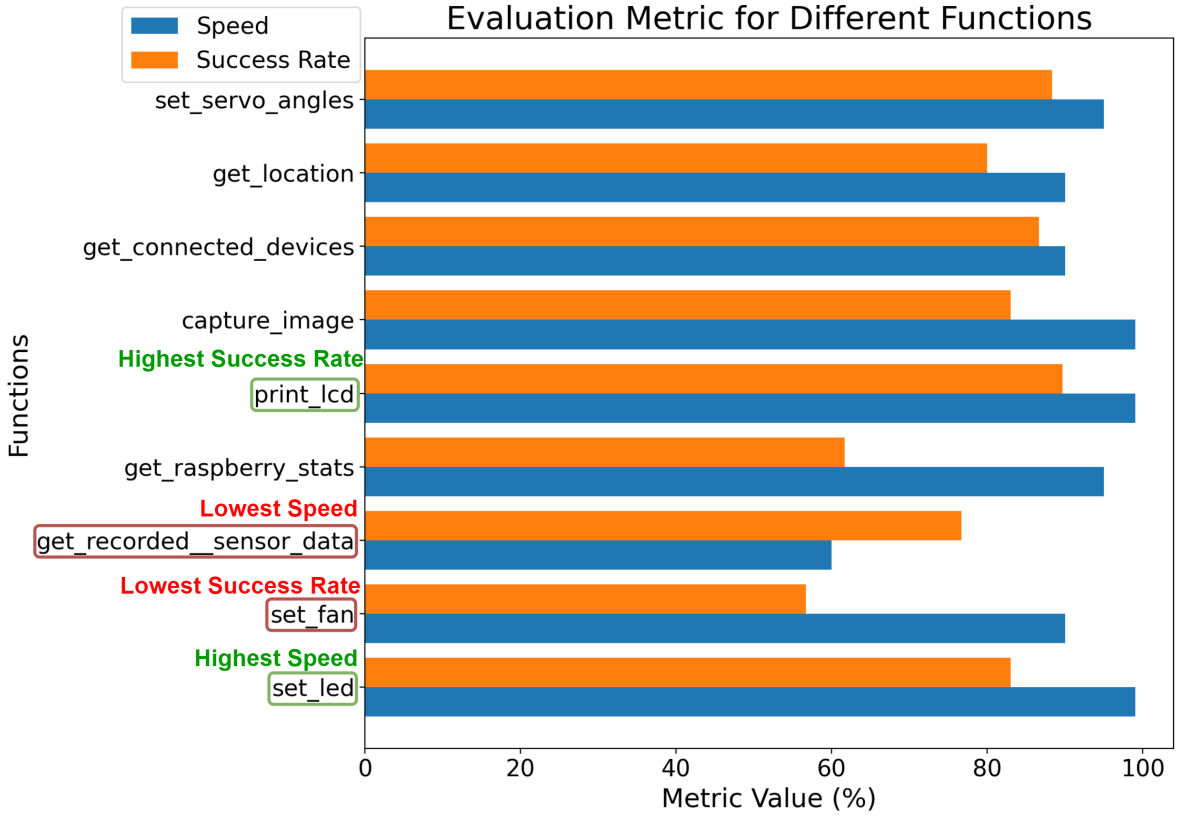

Figure 9. Evaluation metrics for the functions defined in Table 2.

Figure 10 illustrates the correlation between message complexity and the number of functions invoked per message in our system, with the LLM temperature set to 0.3. Message complexity refers to the intricacy of user queries, while LLM temperature controls the randomness in the model's outputs, with lower values producing more deterministic responses [46]. As message complexity increases from 0 to 0.6, we see a rise in function calls per message, indicating more processing for intricate queries. This trend indicates that more intricate user queries require a greater number of system operations to process and respond accurately. The peak at 0.6 complexity may be attributed to an increased number of retries due to function failures, highlighting potential areas for system optimization.

Beyond 0.6, function calls decrease as complexity increases, mimicking the performance characteristics typically associated with lower temperature settings in LLMs. It suggests that for highly complex queries, our system adopts a more focused and deterministic approach, reducing the need for multiple function invocations. This shift could be interpreted as a positive adaptation, indicating that the system becomes more efficient in handling very complex tasks by generating more precise and targeted responses. However, it also raises questions about the system's flexibility and creativity in addressing highly complex scenarios, which might benefit from a more exploratory approach.

Figure 11 represents a heatmap which is a graphical representation of 3-dimensional data in this case, our heatmap shows the interplay between message complexity, LLM's temperature, and the resulting success rates of the LLM in executing IoT control tasks. The heatmap reveals a complex relationship between these variables, with higher success rates, (bright yellow areas) concentrated in regions of moderate to high temperatures (0.7 to 1.0) and moderate complexity (0.4 to 0.6). This pattern suggests that the LLM performs optimally when given some degree of freedom to interpret and respond to moderately complex commands. The high temperature in this optimal zone likely allows the model to explore a wider range of potential interpretations and solutions, which is particularly beneficial when dealing with the nuanced and context-dependent nature of IoT control scenarios [17].

 doi:10.20944/preprints202505.0380.v1

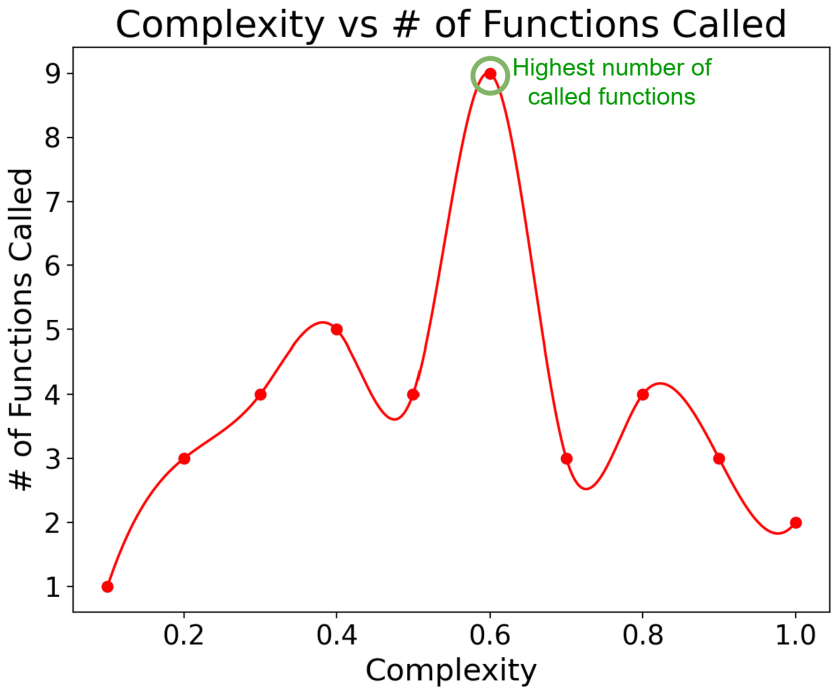

Figure 10. Message complexity against the number of functions called per message.

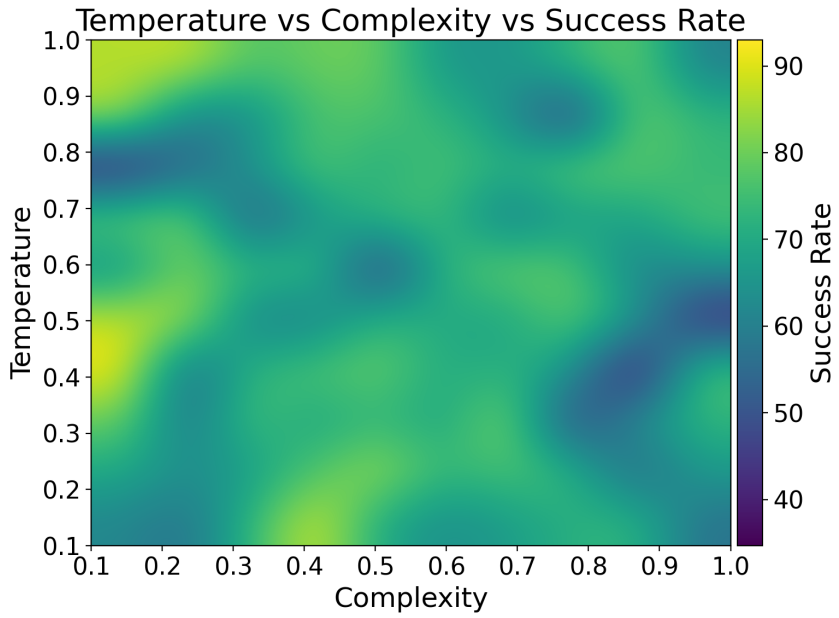

Figure 11. Heatmap for success rate against message complexity and temperature of the LLM.

Conversely, lower success rates are observed in regions of high complexity (0.8 to 1.0) combined with low to medium temperatures (0.1 to 0.6). This pattern indicates that when faced with highly complex instructions, a more constrained or deterministic approach (lower temperature) is less effective. Such scenarios might involve intricate sequences of operations or complex conditional logic that benefit from the model's ability to consider a broader range of possibilities. This finding aligns with the observations of Kannan et al. [14], who noted that LLMs perform better in complex multi-agent robot task planning scenarios when given more freedom to explore diverse solutions.

High success rates are desirable as they translate to more reliable and accurate execution of user commands, leading to improved user experience and system performance. Conversely, low success rates could result in misinterpretation of commands, incorrect device operations, or system failures, potentially leading to user frustration or even safety issues in critical applications.

 doi:10.20944/preprints202505.0380.v1

Understanding these performance characteristics allows for strategic tuning of the LLM's parameters based on the expected complexity of user inputs. For instance, when anticipating complex, multi-step commands, increasing the temperature parameter could potentially boost the system's success rate. This insight could guide the design of user interfaces and command structures, encouraging users to frame their instructions in ways that align with the LLM's strengths.

Figure 12 illustrates the relationship between message complexity and evaluation metrics with the temperature parameter set to 0.5. As complexity increases, the system's performance decrease. The system achieves peak success rate and efficiency at a complexity level around 0.6, where the system can handle sophisticated user requests while maintaining high reliability. A notable shift occurs at a complexity of 0.8, where the success rate diverges from efficiency and speed. This divergence is attributed to the LLM correctly executing functions but struggling with conditional and sequential order. Such behaviours underscore the challenges in maintaining coherent task execution as complexity scales up, even when individual components are processed accurately. Beyond a complexity of 0.9, all three metrics exhibit an upward trend, indicating the model's capability to handle highly complex tasks while maintaining high success rates albeit with potential trade-offs in efficiency and speed. These findings highlight the delicate balance required in system design: while moderate complexity yields optimal overall performance, the system can adapt to highly complex inputs at the cost of reduced efficiency. This insight is crucial for tailoring the Vega system to different use cases, from simple smart home controls to complex industrial applications, where the balance between task complexity and system performance may vary based on specific requirements and operational contexts.

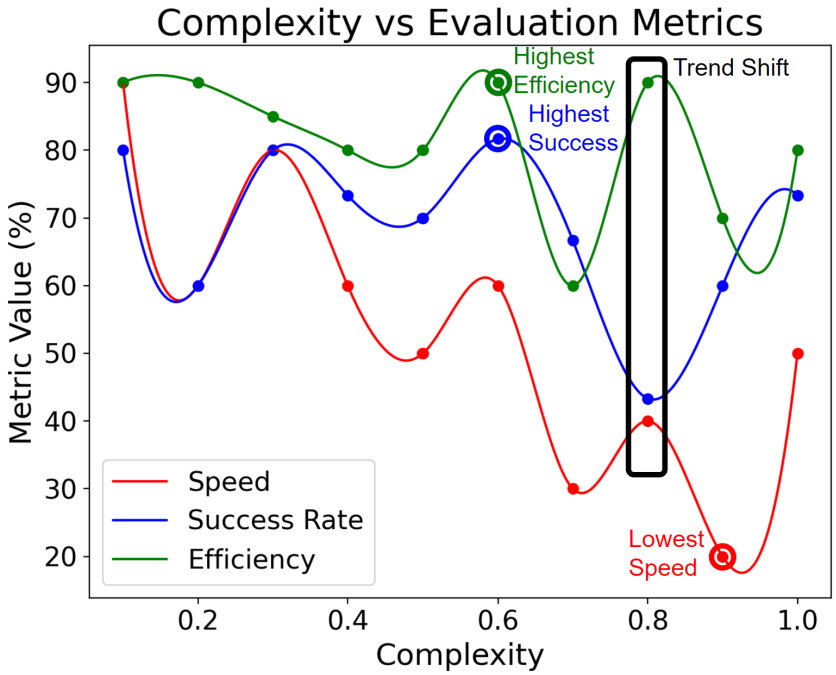

Figure 12. Heatmap for success rate against message complexity and temperature of the LLM.

Figure 13, provides additional insights into the relationship between message complexity and Top P. Top P is an LLM parameter limits token selection to the most probable choices that sum to a specified probability threshold [46]. The heatmap reveals that high success rates (85-90%) occur at low complexity (0.1-0.3) with mid-to-high Top P (0.6-0.9) and at moderate complexity (0.4-0.6) with mid-to-high Top P (0.7-0.9). This pattern is consistent with the findings of Singh et al. [12], who observed that programmatic prompts with well-defined action specifications lead to more successful plan generation in situated robot environments.

 doi:10.20944/preprints202505.0380.v1

Figure 13. Heatmap for success rate against message complexity and Top P of the LLM.

The primary LLM employed for evaluation in this project was GPT-3.5, developed by OpenAI. However, additional manual evaluations conducted using GPT-4 exhibited an appreciable increase in success rate, attributable to its larger training dataset. This observation is in line with the findings of Wu et al. [15], who reported improved performance in personalized robot assistance tasks when using more advanced LLMs.

These findings have significant implications for LLM-driven IoT control systems. The heatmap provide valuable guidance for understanding optimal configurations across different IoT scenarios, contributing to the design of more robust and intuitive LLM-powered interfaces. By carefully tuning parameters such as temperature, Top P, and task complexity, it is possible to achieve higher success rates and more reliable system performance.

The results also highlight the importance of considering the trade-offs between exploration (controlled by temperature and Top P) and task difficulty (complexity) in IoT control scenarios. This aligns with the observations of Vemprala et al. [11], who emphasized the need for balancing exploration and exploitation in robotics applications using LLMs.

Moreover, the non-uniform distribution of success rates across the parameter space underscores the need for adaptive parameter selection strategies in real-world IoT deployments. This is particularly relevant in dynamic environments where task complexity may vary, as noted by Singh et al. [12] in their work on integrating action knowledge and LLMs for task planning in open worlds.

The observed performance improvements with GPT-4 suggest that future advancements in LLM architectures and training methodologies may lead to even more capable IoT control systems. This potential for improvement is consistent with the varied capabilities of LLMs across different tasks, as highlighted by Kumar [1] in their comprehensive survey on the evaluation of LLMs across multiple domains and reasoning types.

However, it is important to note that while LLMs show promise in IoT control applications, they also present challenges related to reliability, interpretability, and security. As pointed out by Sarzaeim et al. [19] in their work on LLM-assisted smart policing systems, careful consideration must be given to the ethical implications and potential biases of LLM-driven decision-making in critical systems.

The experimental results provide valuable insights into the performance characteristics of LLM-powered IoT control systems. By understanding the relationships between input parameters and success rates, developers can optimize these systems for improved reliability and effectiveness across a wide range of IoT applications.

 doi:10.20944/preprints202505.0380.v1

Figure 14, titled "Error Occurrences," shows a pie chart of errors in the Vega platform. LLM-related issues dominate, with wrong format and incorrect logic making up 59% of errors, highlighting the challenges of LLM integration and the need for better prompt engineering or parsing. The error distribution reflects the platform's complexity, where OpenAI timeout errors (14%) suggest potential API-related performance bottlenecks, and web app runtime errors (18%) point to user interface stability issues. The lower rate of RPi device errors (9%) indicates reliable hardware, with most challenges residing in software and AI integration.

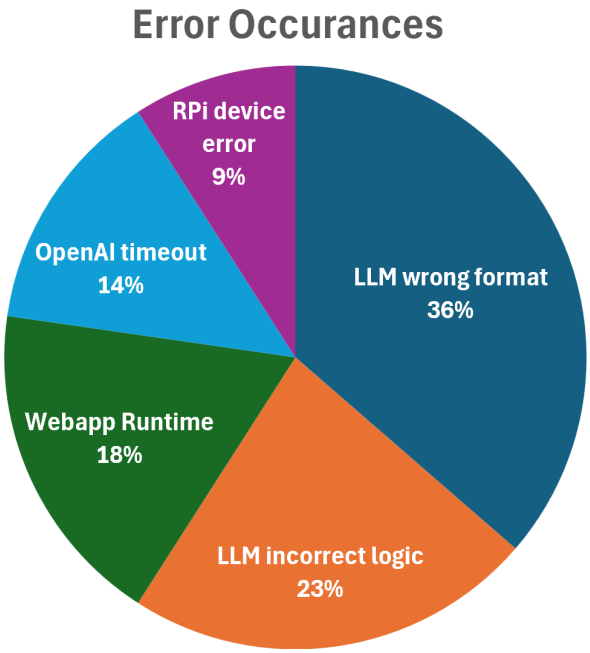

Figure 14. What types of errors occurred throughout testing.

Figure 15 illustrates the success rate of message interpretation by the system across three message length categories: minimal (73.0%), normal (68.0%), and descriptive (84.0%). Notably, the descriptive messages achieve the highest success rate, indicating that more detailed inputs significantly improve performance. Interestingly, minimal messages outperform normal ones, recommending that concise commands may reduce ambiguity. The lower success rate for normal messages implies a potential trade-off between brevity and specificity, emphasizing that either highly detailed or very concise communication may be optimal. While the normal category exhibits a comparatively lower success rate, it still maintains a commendable performance. This suggests that even in less optimal conditions, the system demonstrates robust functionality. Furthermore, the potential integration of more advanced language models, such as GPT-4, could significantly enhance these success rates across all categories, potentially pushing the system's overall performance to new heights.

The LLM, while highly capable in general language understanding, may lack specialized knowledge in IoT hardware control. This could lead to misinterpretations or errors when dealing with intricate or domain-specific instructions. To mitigate this, our system employs additional layers of interpretation and validation, ensuring that the LLM's outputs are appropriately contextualized for IoT applications. While utilizing OpenAI's API incurs ongoing costs, it eliminates the need for extensive local computational resources and the time-intensive process of model training. The high success rates achieved, even with potential limitations, suggest that the benefits outweigh the costs for many applications. However, for scenarios requiring extremely high accuracy or dealing with highly specialized IoT vocabularies, future iterations of the system might benefit from fine-tuning or developing domain-specific models.

 doi:10.20944/preprints202505.0380.v1

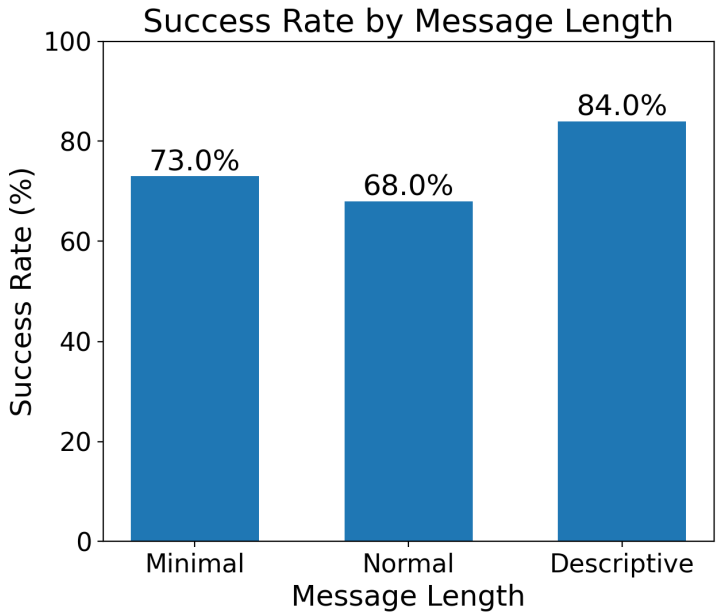

Figure 15. Success rate of different tones of the same message.

In summary, our experimental results reveal several key insights. The system demonstrates a high level of proficiency in interpreting and executing complex commands, particularly excelling with descriptive inputs, achieving an impressive 84% success rate. We observed that performance fluctuates based on message complexity and LLM parameters, with optimal results occurring at moderate complexity levels (0.4-0.6) and higher temperature settings (0.7-1.0). Notably, different IoT functions exhibit varying degrees of success and execution speeds, with text-based operations such as LCD control performing exceptionally well.

5. Conclusions

5.1. Limitations

The current implementation of our system, while innovative, faces several limitations that warrant acknowledgment. Our choice to utilize GPT-3.5 instead of GPT-4, driven by cost considerations at the time of the study, may have constrained the system's overall performance and capability. As our analysis uncovered that the majority of observed errors (59%) stem from LLM-related issues, including incorrect formatting and logic, which clearly indicates areas for future improvement and refinement of the system, however utilizing GPT-4 could help mitigate those issues. A significant concern arises from the reliance on OpenAI's cloud-based service, which introduces potential data privacy issues as user interactions are processed externally. Despite extensive testing across various scenarios, the inherent unpredictability of LLMs remains a challenge, with the possibility of misinterpreting user commands or producing inconsistent responses. The prohibitive cost of fine-tuning at scale presents a barrier to improving the system's accuracy and reliability. Moreover, the current architecture lacks support for real-time feedback, limiting the fluidity of user interactions. The system's dependency on specific hardware (RPi) and software (Python-based server) configurations may restrict its applicability in diverse IoT environments.

5.2. Future Work

Future work aims to address these limitations and expand the system's functionality and applicability. A primary objective is the implementation of real-time feedback mechanisms, enabling live interactions between the LLM, web application, and user, thus enhancing the responsiveness and intuitiveness of the interface. Developing a framework for repeatable logic execution would allow complex commands to run periodically on the IoT device without constant LLM oversight, improving efficiency and reducing computational load. Expanding support to C/C++ based IoT platforms such

 doi:10.20944/preprints202505.0380.v1

as STM32 and Arduino would significantly broaden the system's compatibility with diverse hardware ecosystems. The integration of an MQTT server alongside the existing Flask server could enhance IoT interoperability, allowing for more flexible and real time device communication. Exploring options for local LLM hosting and investigating alternative, potentially open-source LLM solutions could mitigate privacy concerns and potentially reduce operational costs. Additionally, future research could focus on developing more sophisticated natural language understanding capabilities, enabling the system to handle increasingly complex and context-dependent user queries. Lastly future work should focus on developing adaptive parameter selection strategies, improving LLM performance on high-complexity tasks, and addressing the ethical and security considerations associated with LLM deployment in IoT environments. These enhancements would collectively contribute to a more versatile, secure, and user-friendly IoT interface, paving the way for broader adoption in smart homes, industrial settings, and educational environments.

5.3. Conclusion

Overall, this project successfully developed a system that integrates NLP with an IoT infrastructure. The system consists of a web application interfacing with a LLM to interpret user commands, which are executed on an RPi controlling a physical circuit. It features a modular and scalable architecture with comprehensively documented components. The system includes capabilities such as multimodal input through image recognition, complex task interpretation, an accessible and user-friendly chat application supporting various data visualizations, including plots and flow charts, a modular server on the RPi, and compatibility with a wide range of circuit devices. Evaluation results demonstrate the system's effectiveness in handling complex tasks with high success rates, particularly when appropriate LLM parameter settings are used. Case studies showcase real-world applicability like machinery monitoring and drone delivery systems. The system can easily integrate different circuit components and employs intelligent agents for enhanced robustness, making it a comprehensive and user-friendly solution for IoT and embedded systems development. The true innovation lies in the novel combination of components and the intelligent layer that bridges them. By leveraging LLMs for intuitive IoT control, Vega enhances human-machine interaction. Its ability to interpret natural language commands and translate them into specific device actions represents a significant advancement in making IoT technologies more accessible to non-technical users. The system prioritizes modularity, security, and scalability, accommodating new circuit components and visualizations. Societal benefits include improved automation efficiency, technological literacy, responsible practices, and customization to address ethical concerns. Overall, this project marks a significant step toward integrating natural language processing and IoT infrastructures, laying a foundation for intelligent automation and advancements in human-machine interaction.

Acknowledgments: Authors would like to thank the technical staff at the University of Leeds, especially the lab technician Scot Simpson, as he supplied plenty of electronic components that were utilized in the final circuit design. Huge shoutout to Vercel for providing an open source chat app which was a big inspiration for the web application.

References

1. Kumar, P. Large language models (LLMs): survey, technical frameworks, and future challenges. *Artificial Intelligence Review* **2024**, *57*, 260.
2. Liao, Y.; de Freitas Rocha Loures, E.; Deschamps, F. Industrial Internet of Things: A Systematic Literature Review and Insights. *IEEE Internet of Things Journal* **2018**, *5*, 4515–4525. https://doi.org/10.1109/JIOT.2018.2834151.
3. Flohr, L.A.; Kalinke, S.; Krüger, A.; Wallach, D.P. Chat or Tap? – Comparing Chatbots with 'Classic' Graphical User Interfaces for Mobile Interaction with Autonomous Mobility-on-Demand Systems. In Proceedings of the Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction, New York, NY, USA, 2021; MobileHCI '21. https://doi.org/10.1145/3447526.3472036.

 doi:10.20944/preprints202505.0380.v1

4. Kassab, W.; Darabkh, K.A. A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations. *Journal of Network and Computer Applications* **2020**, *163*, 102663. https://doi.org/https://doi.org/10.1016/j.jnca.2020.102663.
5. OpenAI. *Generative Pre-trained Transformer (GPT) Models*, 2023.
6. Al-Qaseemi, S.A.; Almulhim, H.A.; Almulhim, M.F.; Chaudhry, S.R. IoT architecture challenges and issues: Lack of standardization. In Proceedings of the 2016 Future Technologies Conference (FTC), 2016, pp. 731–738. https://doi.org/10.1109/FTC.2016.7821686.
7. Taylor, R.N.; Medvidovic, N.; Dashofy, E.M. *Software Architecture: Foundations, Theory, and Practice*; John Wiley & Sons, 2010; p. 736.
8. Kadiyala, E.; Meda, S.; Basani, R.; Muthulakshmi, S. Global industrial process monitoring through IoT using Raspberry pi. In Proceedings of the 2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), 2017, pp. 260–262. https://doi.org/10.1109/ICNETS2.2017.8067944.
9. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *CoRR* **2017**, *abs/1706.03762*, [1706.03762].
10. Maddigan, P.; Susnjak, T. Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models. *IEEE Access* **2023**, *11*, 45181–45193. https://doi.org/10.1109/ACCESS.2023.3274199.
11. Vemprala, S.H.; Bonatti, R.; Bucker, A.; Kapoor, A. ChatGPT for Robotics: Design Principles and Model Abilities. *IEEE Access* **2024**, *12*, 55682–55696. https://doi.org/10.1109/ACCESS.2024.3387941.
12. Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; Garg, A. ProgPrompt: program generation for situated robot task planning using large language models. *Autonomous Robots* **2023**, *47*, 999–1012. https://doi.org/10.1007/s10514-023-10135-3.
13. Driess, D.; Xia, F.; Sajjadi, M.S.M.; Lynch, C.; Chowdhery, A.; Ichter, B.; Wahid, A.; Tompson, J.; Vuong, Q.; Yu, T.; et al. PaLM-E: an embodied multimodal language model. In Proceedings of the Proceedings of the 40th International Conference on Machine Learning. JMLR.org, 2023, ICML'23.
14. Kannan, S.S.; Venkatesh, V.L.N.; Min, B.C. SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models, 2024, [arXiv:cs.RO/2309.10062].
15. Wu, J.; Antonova, R.; Kan, A.; Lepert, M.; Zeng, A.; Song, S.; Bohg, J.; Rusinkiewicz, S.; Funkhouser, T. TidyBot: personalized robot assistance with large language models. *Autonomous Robots* **2023**, *47*, 1087–1102. https://doi.org/10.1007/s10514-023-10139-z.
16. King, E.; Yu, H.; Lee, S.; Julien, C. Sasha: Creative Goal-Oriented Reasoning in Smart Homes with Large Language Models. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* **2024**, *8*. https://doi.org/10.1145/3643505.
17. Petrović, N.; Koničanin, S.; Suljović, S. ChatGPT in IoT Systems: Arduino Case Studies. In Proceedings of the 2023 IEEE 33rd International Conference on Microelectronics (MIEL), 2023, pp. 1–4. https://doi.org/10.1109/MIEL58498.2023.10315791.
18. Zhong, N.; Wang, Y.; Xiong, R.; Zheng, Y.; Li, Y.; Ouyang, M.; Shen, D.; Zhu, X. CASIT: Collective Intelligent Agent System for Internet of Things. *IEEE Internet of Things Journal* **2024**, *11*, 19646–19656. https://doi.org/10.1109/JIOT.2024.3366906.
19. Sarzaeim, P.; Mahmoud, Q.H.; Azim, A. A Framework for LLM-Assisted Smart Policing System. *IEEE Access* **2024**, *12*, 74915–74929. https://doi.org/10.1109/ACCESS.2024.3404862.
20. Xu, Z.; Wu, H.; Chen, X.; Wang, Y.; Yue, Z. Building a Natural Language Query and Control Interface for IoT Platforms. *IEEE Access* **2022**, *10*, 68655–68668. https://doi.org/10.1109/ACCESS.2022.3186760.
21. Daniel, G.; Cabot, J.; Deruelle, L.; Derras, M. Xatkit: A Multimodal Low-Code Chatbot Development Framework. *IEEE Access* **2020**, *8*, 15332–15346. https://doi.org/10.1109/ACCESS.2020.2966919.
22. Jiang, F.; Dong, L.; Peng, Y.; Wang, K.; Yang, K.; Pan, C.; Niyato, D.; Dobre, O.A. Large Language Model Enhanced Multi-Agent Systems for 6G Communications, 2023, [arXiv:cs.AI/2312.07850].
23. Chen, T.Y.; Chiu, Y.C.; Bi, N.; Tsai, R.T.H. Multi-Modal Chatbot in Intelligent Manufacturing. *IEEE Access* **2021**, *9*, 82118–82129. https://doi.org/10.1109/ACCESS.2021.3083518.
24. Santos, G.A.; de Andrade, G.G.; Silva, G.R.S.; Duarte, F.C.M.; Costa, J.P.J.D.; de Sousa, R.T. A Conversation-Driven Approach for Chatbot Management. *IEEE Access* **2022**, *10*, 8474–8486. https://doi.org/10.1109/ACCESS.2022.3143323.
25. Li, C.; Zhang, X.; Chrysostomou, D.; Yang, H. ToD4IR: A Humanised Task-Oriented Dialogue System for Industrial Robots. *IEEE Access* **2022**, *10*, 91631–91649. https://doi.org/10.1109/ACCESS.2022.3202554.

 doi:10.20944/preprints202505.0380.v1

26. Roller, S.; Dinan, E.; Goyal, N.; Ju, D.; Williamson, M.; Liu, Y.; Xu, J.; Ott, M.; Smith, E.M.; Boureau, Y.L.; et al. Recipes for Building an Open-Domain Chatbot. In Proceedings of the Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume; Merlo, P.; Tiedemann, J.; Tsarfaty, R., Eds., Online, 2021; pp. 300–325. https://doi.org/10.18653/v1/2021.eacl-main.24.
27. Tanenbaum, A.S.; Van Steen, M. *Distributed Systems: Principles and Paradigms*, 1st ed.; Prentice Hall: Upper Saddle River, NJ, 2002.
28. Han, J.; E, H.; Le, G.; Du, J. Survey on NoSQL database. In Proceedings of the 2011 6th International Conference on Pervasive Computing and Applications, 2011, pp. 363–366. https://doi.org/10.1109/ICPCA.2011.6106531.
29. Bhosale, S.; Patil, M.; Patil, P. International Journal of Computer Science and Mobile Computing SQLite: Light Database System. *International Journal of Computer Science and Mobile Computing* **2015**, *44*, 882–885.
30. Nikulchev, E.; Ilin, D.; Gusev, A. Technology Stack Selection Model for Software Design of Digital Platforms. *Mathematics* **2021**, *9*. https://doi.org/10.3390/math9040308.
31. Team, R. *React - A JavaScript library for building user interfaces*. Meta Platforms, Inc., 2024.
32. WorkOS. *Radix UI*, 2022. Open-source UI component library for building high-quality, accessible design systems and web apps.
33. Labs, T. *Tailwind CSS*, 2023.
34. Vercel. *Next.js Documentation*, 2024.
35. Ronacher, A. *Flask: Web Development, One Drop at a Time*, 2024.
36. Wings, E. Sensors and Modules, 2023.
37. Smith, J.; Davis, M. Testing and Debugging IoT Projects with Raspberry Pi. *Journal of Internet of Things* **2020**, *8*, 123–135.
38. Brown, M.; Green, S. Integrating Camera Modules with Raspberry Pi for Image Capture Applications. *IEEE Transactions on Consumer Electronics* **2018**, *64*, 145–152.
39. Wilkinson, B.; Allen, M. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, 2nd ed.; Pearson/Prentice Hall, 2005.
40. Wytrębowicz, J.; Cabaj, K.; Krawiec, J. Messaging Protocols for IoT Systems—A Pragmatic Comparison. *Sensors* **2021**, *21*. https://doi.org/10.3390/s21206904.
41. Kim, T.; Wang, Y.; Chaturvedi, V.; Gupta, L.; Kim, S.; Kwon, Y.; Ha, S. LLMem: Estimating GPU Memory Usage for Fine-Tuning Pre-Trained LLMs, 2024, [arXiv:cs.AI/2404.10933].
42. Yang, H.; Yue, S.; He, Y. Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions, 2023, [arXiv:cs.AI/2306.02224].
43. John, M.; Maurer, F.; Tessem, B. Human and social factors of software engineering. In Proceedings of the 27th International Conference on Software Engineering, Los Alamitos, CA, USA, may 2005; p. 686. https://doi.org/10.1109/ICSE.2005.1553657.
44. Kim, C.Y.; Lee, C.P.; Mutlu, B. Understanding Large-Language Model (LLM)-powered Human-Robot Interaction. In Proceedings of the Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction, New York, NY, USA, 2024; HRI '24, p. 371–380. https://doi.org/10.1145/3610977.3634966.
45. Babar, M.; Zhu, L.; Jeffery, R. A framework for classifying and comparing software architecture evaluation methods. In Proceedings of the 2004 Australian Software Engineering Conference. Proceedings., 2004, pp. 309–318. https://doi.org/10.1109/ASWEC.2004.1290484.
46. Ruman. Setting top-k, top-P and temperature in LLMS. https://rumn.medium.com/setting-top-k-top-p-and-temperature-in-llms-3da3a8f74832, 2024. Accessed: 19 August 2024.