Final Report

Probationary Faculty Development Grant (PFDG) 2026

Project Title

"AI-Powered Large Language Model Framework for Natural Language Robot Control (LLM-NLRC)"

Rohollah Moghadam, Ph.D., SMIEEE

Assistant Professor of Electrical and Electronic Engineering

College of Engineering and Computer Science

California State University, Sacramento

June 2026

TableP of Contents

Acknowledgments

The principal investigator (PI) gratefully acknowledges the support of the Sacramento State Division of Academic Affairs and the Diversity and Inclusion Awards Committee for supporting this project through the Probationary Faculty Development Grant program. Their support made it possible to acquire and integrate robotics, artificial intelligence, sensing, and embedded-computing materials into a working instructional platform. The project reflects the university's commitment to faculty development, educational innovation, inclusive excellence, and expanded student access to emerging technologies. The PI also extends appreciation to the Department of Electrical and Electronic Engineering and the College of Engineering and Computer Science for supporting the development of hands-on instructional resources in robotics and artificial intelligence. The project is intended to benefit students in robotics-related courses and to provide additional opportunities for students involved in the Sacramento State Competitive Robotics Club.

Executive Summary

This final report summarizes the development of an AI-powered natural language robot control platform. The completed platform uses a working mobile robot as the physical system and connects it with modern AI-based command interpretation. The hardware setup includes an NVIDIA Jetson Orin Nano, a Pololu Romi Robot Kit for FIRST, a Raspberry Pi 4, a 32U4 control board, an Intel RealSense D435 camera, an RPLIDAR A1 sensor, a Gens Ace 3300 mAh 3S 11.1 V LiPo battery, Six AA rechargeable batteries and an XT60-to-Orin jack switch assembly. This combination provides a compact and expandable robotics platform that supports motion control, perception, communication, and future autonomous behavior.

The software framework is implemented as ARIA (Autonomous Robot Intelligence Assistant), a full-stack platform built on Streamlit for the browser-based user interface, Google Gemini 2.5 Flash for AI language interpretation, Pydantic for structured output validation, and MQTT for lightweight publish/subscribe communication between the web interface and the robot. A user can issue commands such as “move forward slowly for two seconds,” “turn right 45 degrees,” “stop,” “take a picture,” or “scan the room.” The system translates the natural language request into a validated JSON command object and transmits it to the Jetson Orin Nano over MQTT. For sensor commands, the robot returns live data: the RPLIDAR A1 produces a 360-degree polar scan that is visualized as an interactive chart in the browser, and the Intel RealSense D435 combined with YOLOv8 produces an annotated image showing detected objects with class names and depth distances. Safety constraints are enforced at the schema level: speed is capped at 0.3 m/s, duration at 3.0 seconds, and only supported command types are accepted, with ambiguous or unsafe inputs automatically defaulting to stop.

The educational value of the project is its ability to lower the entry barrier to robotics. Students who may not yet be comfortable with programming can still interact with a robot, observe physical behavior, and then gradually connect those observations to programming, sensors, embedded systems, and AI concepts. The platform is appropriate for future integration into EEE 187 Robotics,

EEE 225 Advanced Robotics and Control, independent student projects, and robotics club workshops. It also supports the university's diversity and inclusion goals by making advanced robotics more approachable for students with different levels of prior technical preparation.

1. Introduction

Robotics education has traditionally required students to learn several technical layers before they can operate a robot. These layers include programming languages, microcontroller interfaces, motor drivers, sensor communication, and system-level debugging. While these topics are important for engineering students, they can also create a steep initial barrier. Students may become discouraged when early robotics experiences focus more on syntax and configuration than on the behavior and purpose of robotic systems.

Recent advances in artificial intelligence provide an opportunity to change how students first encounter robotics. Large Language Models have demonstrated strong natural language understanding and can produce structured outputs from conversational input. When connected carefully to a physical system, an LLM can act as an interface that converts ordinary language into a well-defined robot command (Tellex et al., 2020). This does not eliminate the need for engineering knowledge, but it allows students to begin with meaningful interaction and then investigate the underlying technical mechanisms.

The LLM-NLRC project was developed around this idea. The project combines a conversational interface, structured robot command generation, a validation layer, embedded computing, and a working mobile robot platform. The goal is not to create a fully autonomous commercial robot, but to create a reliable instructional platform that helps students understand how AI can connect to hardware and how natural language can be translated into physical action (Ahn et al., 2022).

The project is also connected to inclusive teaching. Students come to engineering courses with different levels of prior preparation. Some have already programmed robots or microcontrollers, while others may be encountering these topics for the first time. A natural language robot-control interface provides a more welcoming starting point while still preserving opportunities for deeper technical learning. This makes the project well aligned with the intent of an internal university grant focused on faculty development, instructional improvement, and student access.

This project also supports Sacramento State's broader interest in integrating AI into teaching and learning. By connecting a language model to a physical robot, students move beyond purely text-based AI use and experience embodied AI, where commands produce direct, observable physical outcomes.

2. Project Objectives

The first objective was to create a natural language interface for mobile robot control. The system was designed so that users could provide commands in plain English rather than writing code or using a specialized control interface. This objective is central to the instructional purpose of the project because it allows students to interact with a robot at the beginning of a learning experience and then use that interaction as a pathway into more technical topics.

The second objective was to integrate the natural language interface with a working robot platform. The project was not intended to remain a software-only demonstration. The robot is operational, and the final hardware configuration includes a Jetson Orin Nano, Romi mobile platform, Raspberry Pi 4, 32U4 controller, RealSense D435 camera, RPLIDAR A1, LiPo battery, and switching power hardware. The integration of these parts allows the project to demonstrate the full pathway from user command to physical robot behavior.
The third objective was to organize the system in a way that supports instruction. The software and hardware architecture were intentionally designed to be modular. Students can study the natural language processing layer, the command schema, the validation logic, the motor-control interface, or the sensor modules separately. This modularity makes the platform useful for demonstrations, laboratories, independent projects, and future course assignments.
The fourth objective was to support future educational and outreach activities. The project is intended to contribute to EEE 187 Robotics, EEE 225 Advanced Robotics and Control, and activities associated with the competitive robotics club. The platform provides a hands-on example of AI-assisted robot control and can be used to introduce students to human-robot interaction, embedded computing, sensor integration, and safety-aware command execution.

3. System Architecture

The LLM-NLRC system is organized as a layered architecture and is implemented as ARIA (Autonomous Robot Intelligence Assistant), a full-stack educational robot control platform, as shown in Figure 1. At the highest level, the user interacts with the system through a browser-based ***Streamlit*** web interface. The user enters a command in ordinary English. The AI interpretation layer, powered by Google Gemini 2.5 Flash, identifies the user's intended action and extracts structured parameters including direction, speed, duration, turn angle, and requested sensor operation. The command is then represented as a validated JSON object conforming to a strict ***Pydantic*** schema before being transmitted to the robot.
A validation layer is placed between the AI interpretation layer and the physical robot. This layer is important because natural language can be ambiguous and because a robot operating in a classroom environment must be constrained for safety. The validation layer is implemented using Pydantic, a Python data-validation library that enforces type constraints at runtime. Specifically, speed is bounded between 0.0 and 0.3 m/s, duration is capped at 3.0 seconds, and turn angle is restricted to a range of 0 to 180 degrees. Commands belonging to a set of supported actions such as move_forward, move_backward, turn_left, turn_right, stop, get_lidar, and get_image, are accepted. Any value outside the bounds, or any unrecognized command type, is rejected before it reaches the motor-control layer. The system defaults to a stop command when the user input is ambiguous or unsafe.
The lower level of architecture is implemented on the robot hardware running on the NVIDIA Jetson Orin Nano. A receiver script (aria_mqtt_receiver.py) runs permanently on the Jetson, listening for validated JSON commands arriving over an MQTT messaging channel. ***MQTT*** (Message Queuing Telemetry Transport) is a lightweight publish/subscribe protocol designed for embedded and IoT devices. The web interface and the robot communicate exclusively through an MQTT broker using

two topics: *aria/robot/cmd* for commands flowing from the web interface to the robot, and *aria/robot/data* for sensor data flowing back from the robot to the web display. This decoupled architecture means the web interface and the robot do not need to know each other's network addresses, making the system portable and easy to demonstrate in different classroom environments. The 32U4 control board handles timing-sensitive low-level motor control, while the Jetson manages AI inference, sensor orchestration, and network communication. The RPLIDAR A1 and Intel RealSense D435 provide environmental sensing, object detection, and depth information that are returned to the web interface and visualized in real time.

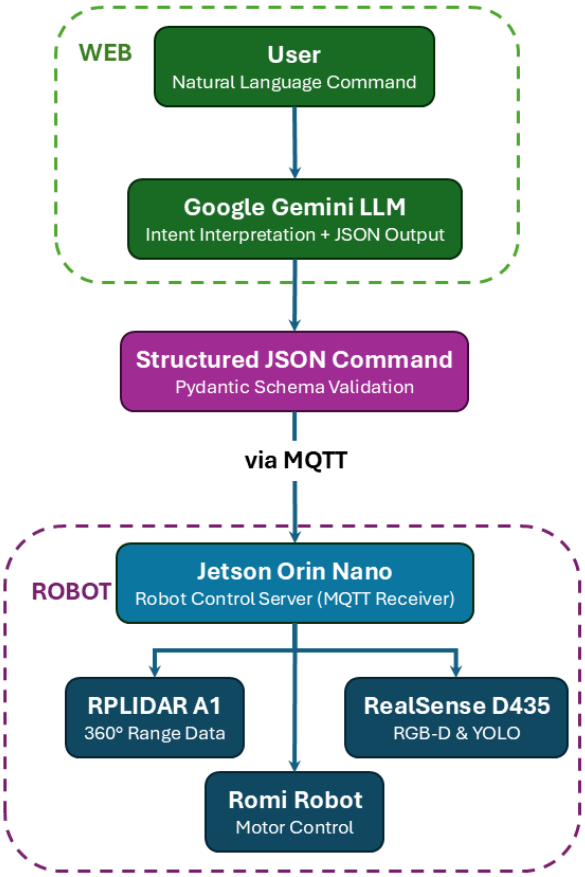

Figure 1. Overall System Architecture: user interaction, AI translation and embedded execution

4. Hardware Setup

The hardware setup was selected to balance capability, affordability, and educational value as given in Table 1 and Figures 2 and 3. The confirmed part list includes the NVIDIA Jetson Orin Nano, Pololu Romi Robot Kit for FIRST, Raspberry Pi 4, 32U4 control board, Intel RealSense D435, RPLIDAR A1, Gens Ace G-Tech 3300 mAh 3S 11.1 V 60C LiPo battery pack with XT60 plug, and an XT60-to-Orin jack switch. This combination creates a working mobile robot with AI computing, low-level motor control, camera-based perception, lidar-based sensing, and portable power. The

Jetson Orin Nano is used as the main AI and robotics control and processing unit. NVIDIA describes the Jetson Orin Nano developer platform as an energy-efficient platform for AI and robotics applications, and it is well suited for student projects that require local processing rather than relying only on cloud computation. The Jetson provides a Linux environment and supports robotics software, Python-based development, AI libraries, and future vision or inference pipelines. The Pololu Romi Robot Kit for FIRST provides the mobile robot base. The Romi platform is appropriate for educational use because it is compact, accessible, and built around a chassis and 32U4 control board that can support student programming activities. The 32U4 control board handles low-level motor-control tasks, which helps separate timing-sensitive hardware control from high-level AI and communication logic. The Intel RealSense D435 adds RGB-D sensing. This allows the platform to support image capture and depth-based perception experiments. The RealSense camera is particularly valuable pedagogically because it helps students connect robotics with computer vision and spatial sensing. The RPLIDAR A1 adds 360-degree ranging capability, allowing students to explore obstacle detection, scanning, and future mapping applications. The battery and XT60 switching hardware support mobile operation and make the robot easier to use in classroom demonstrations.

Table 1. Hardware components and instructional role.

Component	Role in Project	Instructional Value
Jetson Orin Nano	Main AI and robot-control computer	Introducing edge AI, Linux robotics, and local processing
Pololu Romi Robot Kit for FIRST	Mobile robot base	Provides a compact educational robot platform
Raspberry Pi 4	Supporting computing and I/O platform	Supports distributed computing and peripheral experiments
32U4 control board	Low-level motor-control interface	Shows separation of high-level and low-level control
Intel RealSense D435	RGB-D camera	Supports image capture, depth sensing, and future computer vision
RPLIDAR A1	360-degree laser scanner	Supports obstacle detection, scanning, and mapping concepts
3S LiPo battery and XT60 switch	Portable power system	Enables mobile classroom demonstrations
6x AA Rechargeable Batteries	Romi Chassis Power Supply	Demonstrates separate power domains for computing and actuation

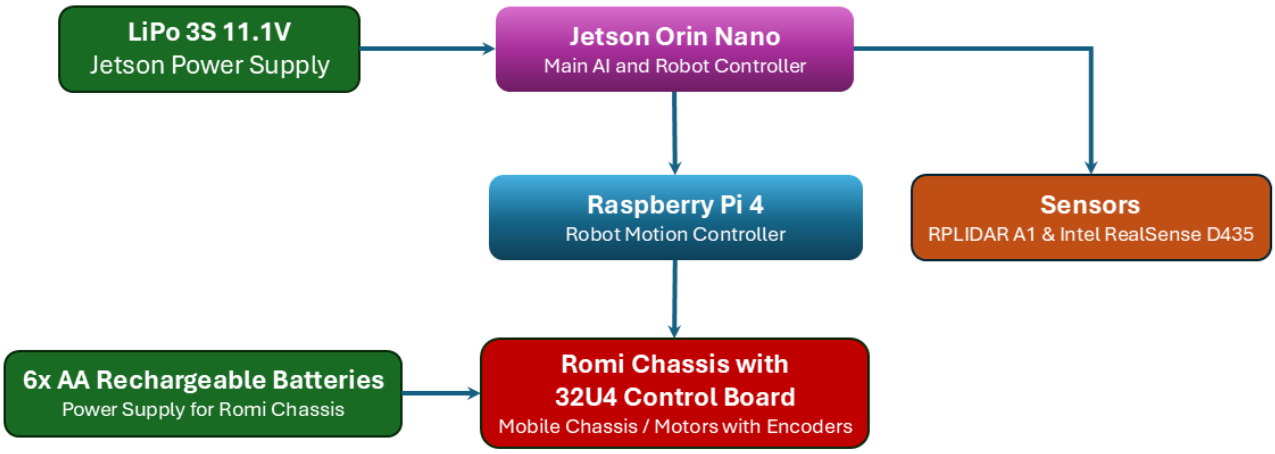

Figure 2. Hardware setup: data and power flow among computing, sensing, motion components.

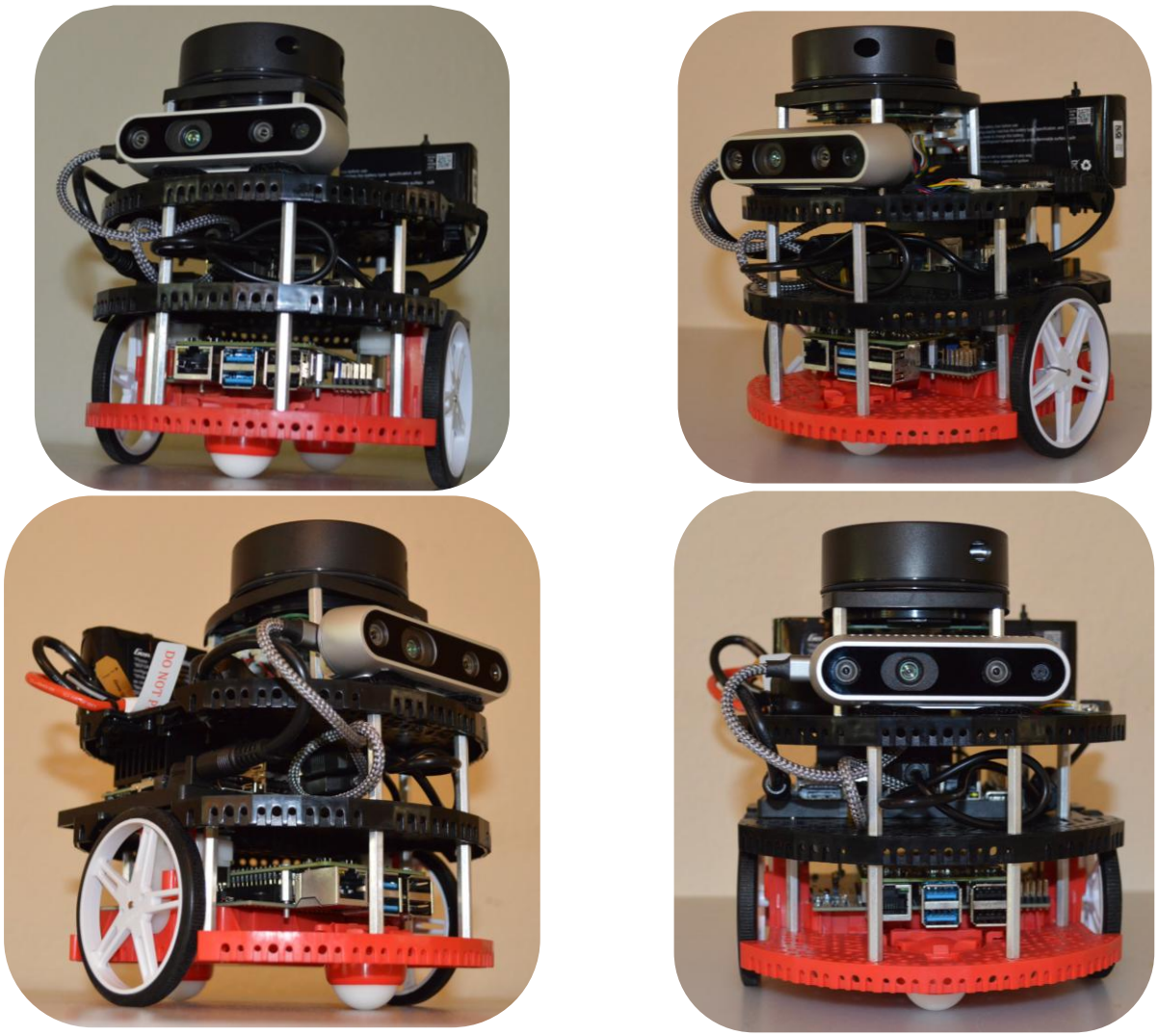

Figure 3. Autonomous Robot Intelligence Assistant (ARIA)

5. Software and Implementation

The software requirements were defined to support a simple but reliable educational workflow. The system must receive a natural language command, translate it into a structured command, validate the command, transmit it to the robot over a network, execute the requested behavior, and return useful sensor feedback to the user. The software was organized into three main files that divide responsibilities cleanly between the user-facing web interface and the robot-side hardware handler. The web interface (app.py) runs on any machine with internet access. The robot-side handler (aria_mqtt_receiver.py) runs permanently on the Jetson Orin Nano. The vision module (camera_yolo_module.py) also runs on the Jetson and handles Intel RealSense camera capture and YOLOv8 object detection. The full technology stack is summarized in Figure 4.

The first module is the natural language interface built using Streamlit, a Python library that generates a browser-based control panel with minimal code. When a user types a command, the interface passes it to Google Gemini 2.5 Flash using a system prompt that defines ARIA's role and constraints. The system prompt instructs the AI to act as a robot command assistant, to interpret natural language into exactly one safe command, and to default to stop whenever a request is unclear or unsafe. The AI returns its response in a constrained JSON format defined by a Pydantic schema, a technique called structured output or constrained generation. This ensures the AI response always conforms to the exact data types and value ranges required by the robot controller, eliminating the need to parse free-form text. The RobotCommand schema defines the following fields: command (one of supported action types), speed (0.0 to 0.3 m/s), duration (0.0 to 3.0 seconds), angle (0.0 to 180.0 degrees), and reason (a plain English explanation of how the command was interpreted). Pydantic validates this schema at runtime and raises an error if any field falls outside its defined range, providing a software-level safety boundary independent of the AI model output.

The second module is the MQTT communication layer. Once a valid RobotCommand object is produced, the web interface serializes it to a JSON string and publishes it to the MQTT broker on the topic *aria/robot/cmd* using ***paho-mqtt***. The robot-side receiver script subscribes to this topic and dispatches the command to the appropriate hardware handler. After executing a sensor command, the robot publishes results back on *aria/robot/data*. The web interface runs a background listener thread with a configurable timeout (15 seconds for lidar, 20 seconds for images) that waits for this response and updates the display. The use of MQTT rather than HTTP was a deliberate design choice: MQTT's publish/subscribe architecture decouples the sender and receiver so that neither side needs to know the other's IP address, which simplifies classroom deployment. MQTT is also extremely lightweight and well suited to embedded systems with limited bandwidth.

The third module is the robot-control layer running on the Jetson Orin Nano. The receiver script (*aria_mqtt_receiver.py*) listens continuously for JSON commands and dispatches them to the appropriate handler. Motion commands (move_forward, move_backward, turn_left, turn_right, stop) are translated into low-level motor-control actions executed through the 32U4 controller. Sensor commands (get_lidar, get_image) are routed to dedicated modules. The lidar module captures a 360-degree scan from the RPLIDAR A1 and returns the scan as arrays of angles and distances, which the web interface visualizes as an interactive polar chart using Plotly. The camera/vision module

(*camera_yolo_module.py*) captures an RGB-D frame from the Intel RealSense D435, runs YOLOv8 object detection on the color image, and samples median depth values from a 9 × 9 pixel patch centered on each detected bounding box to produce robust distance estimates. The annotated image is JPEG-compressed and base64-encoded so it can be transmitted as a JSON string over MQTT and displayed in the web interface without saving to disk. A singleton pattern is used to keep the camera pipeline and YOLO model loaded in memory across multiple requests, avoiding the device-busy errors that occur when the camera is repeatedly opened and closed.

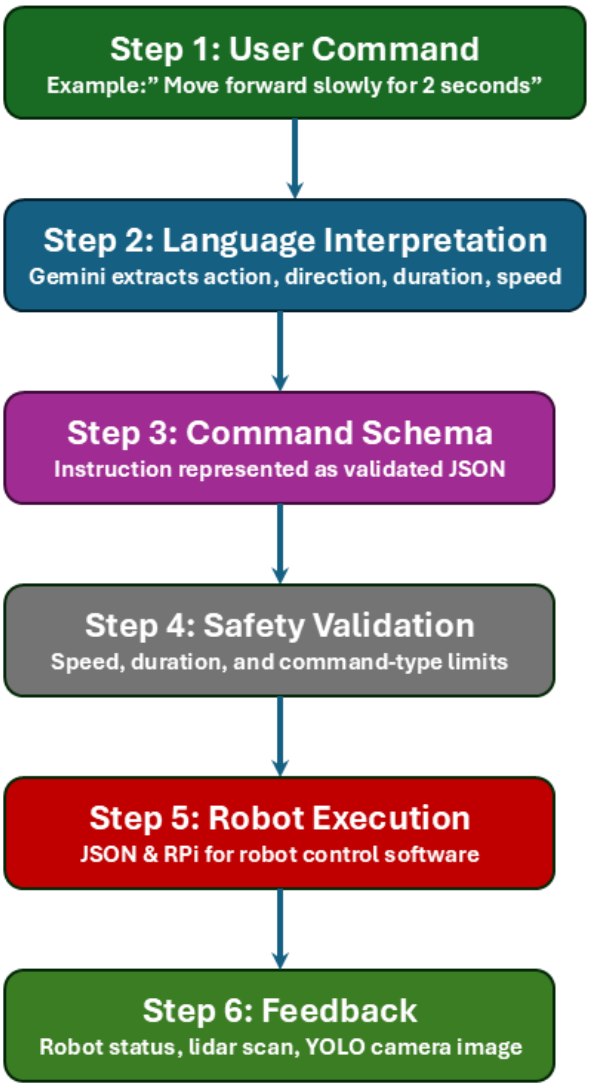

Figure 4. Software process transforming conversational language into validated robot actions

6. Natural Language Robot Control Procedure

The natural language robot-control procedure begins when a user enters a command through the conversational interface. The user does not need to know the exact syntax required by the robot. Instead, the user can describe the desired action in ordinary language. This is important educationally

because it allows students to focus first on robot behavior and system response before studying the underlying code. After the user command is entered, the AI interpretation layer identifies the command type and extracts relevant parameters. A motion command may include direction, speed modifier, and duration. A sensor command may request lidar data or image capture. The command is then converted into a structured JSON object. For example, the input "move forward slowly for 2 seconds" is interpreted as: command: move_forward, speed: 0.1, duration: 2.0, angle: 0.0. The input "take a picture" maps to command: get_image. The input "scan the room" maps to command: get_lidar. This structured representation is the interface between the language model and the robot-control program, and it ensures that the robot never receives ambiguous or open-ended text directly. The structured command is validated by Pydantic before transmission. If the command is complete and all fields are within their defined bounds, it is serialized to JSON and published to the MQTT broker on the *aria/robot/cmd* topic. The robot-side receiver picks up the message and dispatches the appropriate handler. For a motion command, the Jetson sends speed and direction instructions to the 32U4 control board, which drives the Romi platform motors. For a lidar command, the RPLIDAR A1 performs a scan, and the results are packaged as angle and distance arrays and published back to the web interface. For an image command, the RealSense camera captures an aligned RGB-D frame, YOLOv8 processes the color image to detect objects and estimate their distances, and the annotated frame is returned as a base64-encoded JPEG. The system then displays the lidar polar chart or annotated camera image in the browser alongside a summary of the command that was executed. If the command is outside the allowed range or unrecognized, the system defaults to stop and can optionally display the reason field from the structured response, which explains in plain language how the AI interpreted the original input.

During development, special attention was given to consistent handling of speed parameters across different command types. One observed issue involved the interpretation of speed modifiers such as "slowly," "quickly," and "at medium speed." The Gemini model consistently mapped these qualitative descriptors to numeric speed values within the defined range (0.0–0.3 m/s), with "slowly" typically mapping to approximately 0.1 m/s and "quickly" mapping near 0.25 m/s. A separate challenge arose from the RealSense D435 camera initialization: when the camera pipeline was opened and closed on each request, the operating system sometimes failed to release the device before the next request arrived, causing errno=16 (device busy) errors. This was resolved by implementing a singleton pattern that initializes the camera pipeline and the YOLO model once on the first request and keeps them resident in memory for subsequent calls, with a proper shutdown hook to release the device on program exit. These integration challenges provided valuable real-world debugging examples that are directly applicable to embedded systems coursework and student project mentorship.

7. Safety and Validation Considerations

Safety is an essential part of any classroom robotics platform. Even small robots can create problems if they move unexpectedly, collide with objects, or respond incorrectly to ambiguous commands. For this reason, the LLM-NLRC framework includes a validation layer between AI interpretation and

robot execution. The purpose of the validation layer is to ensure that generated commands are understandable, supported, and bounded before they reach the motor-control layer.
The validation framework is implemented at two levels. The first level is the Pydantic schema enforced on the web interface side before any command is transmitted. The speed field is restricted to a maximum of 0.3 m/s, which keeps the Romi platform at a safe demonstration speed in a classroom setting. The duration field is capped at 3.0 seconds, preventing sustained motion that could cause the robot to leave the demonstration area or collide with furniture. The angle field for turn commands is bounded at 180 degrees. The command field only accepts one of seven enumerated action strings; any other value is rejected by the schema before it reaches the network layer. If the AI model produces a value outside the allowed range due to an unusually phrased input, Pydantic raises a validation error, and the system falls back to a stop command. The second level is the system prompt provided to the Gemini model, which explicitly instructs the AI to return a stop command whenever the user's intent is unclear, potentially hazardous, or outside the set of supported actions.
Sensor data also supports safe operation. The RPLIDAR A1 provides 360-degree ranging data that can be used to detect nearby obstacles before executing motion commands. In the current implementation, lidar data is returned on demand and visualized as an interactive polar chart that clearly shows object proximity in all directions around the robot. The Intel RealSense D435 provides both color and depth data, allowing the system to report the distance to detected objects with sub-meter accuracy using the YOLOv8 vision pipeline. The depth alignment step in the RealSense pipeline ensures that depth values correspond precisely to the correct color pixels, which is important for accurate distance reporting. Future versions of the platform can use real-time lidar data to implement automatic obstacle-stop behavior, where the robot halts if any obstacle is detected within a configurable minimum range. These capabilities make the platform increasingly suitable for more advanced autonomous behavior while maintaining the safety constraints appropriate for a classroom environment.

8. Educational Impact

The most important outcome of this project is its educational value. The platform provides a bridge between artificial intelligence and robotics that students can experience directly. Instead of encountering AI as an abstract software tool, students can see AI connected to a physical robot. This makes the learning experience more concrete and engaging.
The natural language interface helps lower the barrier to robotics participation. Students who have limited programming experience can begin by interacting with the robot conversationally. Once they observe how the robot responds, instructors can guide them toward the underlying technical layers. This creates a scaffolded learning pathway: first interaction, then command structure, software implementation, hardware control, and finally advanced topics such as sensing and autonomous behavior.
The project is suitable for future use in EEE 187 Robotics. In that course, students could use the platform to study basic robot motion, sensor integration, motor control, and introductory AI-assisted control. Students could compare natural language commands with equivalent Python or embedded-

code instructions, helping them understand how high-level commands are converted into low-level actions.
The platform is also suitable for EEE 225 Advanced Robotics and Control. Graduate students could use the system to explore more advanced topics such as sensor fusion, lidar-based navigation, computer vision, command validation, and human-robot interaction. Because the system includes both a Jetson Orin Nano and perception sensors, it provides a flexible base for student projects and demonstrations.
Beyond formal courses, the system can support the Sacramento State Competitive Robotics Club. Club members can use the platform for workshops, demonstrations, outreach events, and early-stage prototyping. The natural language interface may also attract students from outside traditional robotics backgrounds and help them engage with the club's technical activities.

9. Project Outcomes

The project produced a fully functional robotics platform and a complete software stack for natural language robot control. The robot hardware was assembled using the confirmed component set provided in the Hardware Set up section. This platform provides a stable, expandable base for classroom demonstrations and future student projects in robotics, embedded systems, and AI.
The project produced a complete software architecture named ARIA (Autonomous Robot Intelligence Assistant) that connects natural language commands to physical robot behavior through a multi-layer pipeline. The architecture includes a *Streamlit* browser-based web interface, a Google Gemini 2.5 Flash AI interpretation layer, a *Pydantic* schema-based validation module, an MQTT publish/subscribe communication system, a robot-side command dispatcher running on the Jetson Orin Nano, a YOLOv8 object detection and Intel RealSense depth-sensing pipeline, an RPLIDAR A1 scanning module with Plotly polar visualization, and a low-level motor-control interface through the 32U4 board. The complete source code is organized into three primary files: app.py (Streamlit web interface), aria_mqtt_receiver.py (robot-side MQTT command handler), and camera_yolo_module.py (vision and object detection module). These components will be used as demonstration material in EEE 187 and EEE 225, or provided as starting code for student capstone and independent study projects. In addition to the system itself, the project generated substantial instructional value through the development process. The debugging and integration work produced several real-world examples applicable directly to coursework: the resolution of the RealSense camera device-busy problem using a singleton pattern illustrates resource management in embedded Linux systems; the Pydantic constrained output approach demonstrates how runtime data validation provides a hardware-independent safety layer; the MQTT decoupled architecture shows how distributed systems can be designed so that components communicate without tightly coupled network dependencies; and the base64 image encoding workflow illustrates how binary data is safely transmitted over text-based protocols. The choice of **YOLOv8s-WorldV2,** a small-footprint open-vocabulary detection model, for object detection and the use of median-patch depth sampling for robust distance estimation are both topics directly relevant to computer vision instruction in advanced robotics courses. Together, these implementation decisions and the problems encountered during development constitute a rich

set of case study material for teaching systems integration, embedded computing, and AI-hardware interfacing.

10. System Performance Demonstrations

To demonstrate the capabilities of the developed AI-Powered Large Language Model Framework for Natural Language Robot Control (LLM-NLRC), several experimental tests were conducted using the ARIA (Autonomous Robot Intelligence Assistant) interface. The objective of these demonstrations was to validate the complete workflow from natural language command interpretation to robot execution and sensor data acquisition.

10.1 LiDAR Data Acquisition and Visualization

Figure 5 illustrates a LiDAR data acquisition test. In this experiment, the user entered the natural language command:

"Provide LiDAR data"

The ARIA interface converted the command into a validated robot instruction:

```
{
  "command": "get_lidar",
  "speed": 0.2,
  "duration": 0,
  "distance_m": 0,
  "angle": 0
}
```

After validation, the command was transmitted to the robot where the onboard Jetson Orin Nano requested a complete scan from the RPLIDAR A1 sensor. The collected range measurements were returned to the ARIA interface and displayed as a polar plot.
The resulting polar scan provides a 360-degree representation of surrounding obstacles and environmental features. Each point represents an object detected by the LiDAR sensor, with angular position corresponding to direction and radial distance corresponding to measured range. This capability forms the foundation for future navigation, mapping, and obstacle avoidance functions.

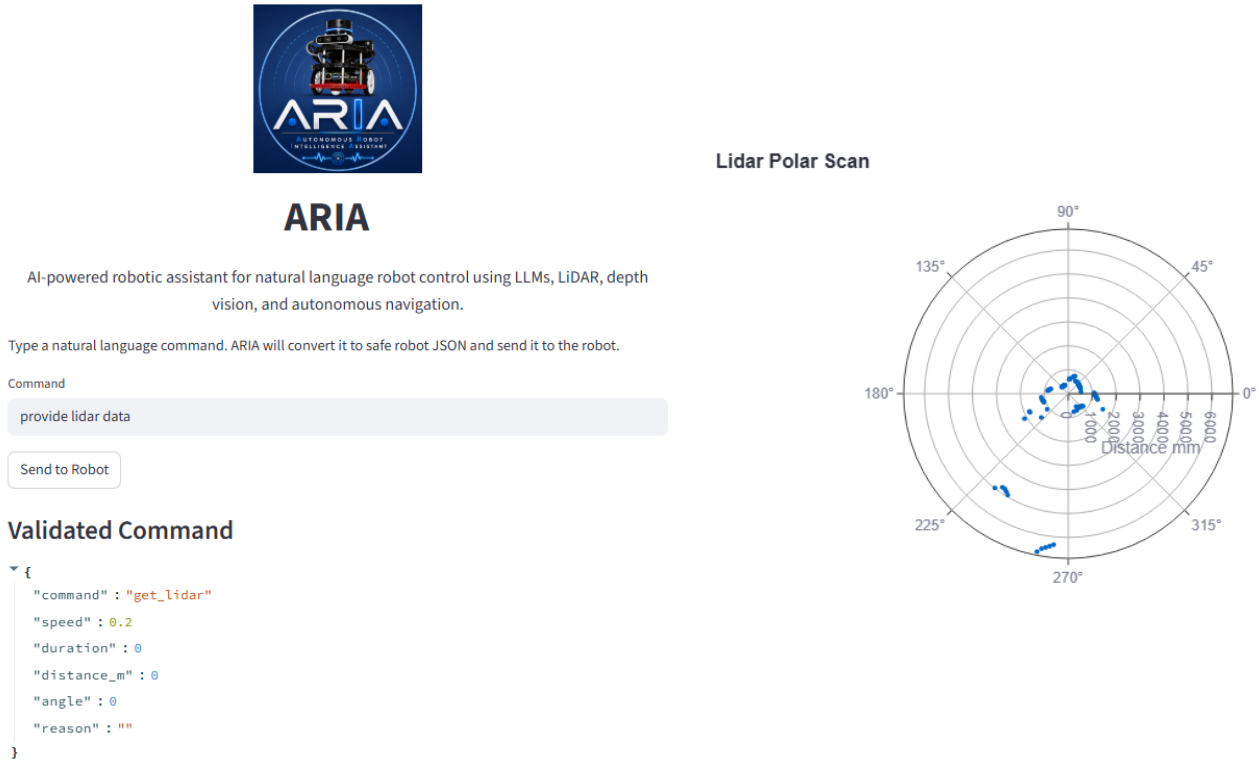

Figure 5. ARIA natural language interface requesting LiDAR data and displaying the resulting polar scan.

10.2 Vision-Based Object Detection Using Natural Language Commands

Figure 6 demonstrates the vision subsystem integrated into the framework. In this experiment, the user issued the command:

"Take a photo"

The LLM translated the request into the following validated instruction:

```
{
  "command": "get_image",
  "speed": 0.2,
  "duration": 0,
  "distance_m": 0,
  "angle": 0
}
```

Upon receiving the command, the Jetson Orin Nano captured an image from the Intel RealSense depth camera and processed the image using the YOLOv8 object detection framework. The detected objects were identified and annotated with bounding boxes, confidence scores, and estimated distances obtained from depth measurements. The experimental results show successful identification of multiple objects including Speaker, Monitor, Laptop, Book and Cup. For each detected object, the system reports Object class label, Detection confidence score and Estimated distance from the robot.

The integration of depth sensing with object detection allows the robot to understand both the identity and spatial location of surrounding objects. This information can later be utilized for autonomous navigation, object search, human-robot interaction, and task-oriented robotic behaviors. The object detection results shown in Figure 6 demonstrate that the developed framework successfully combines natural language understanding, computer vision, and robotic sensing into a unified educational platform.

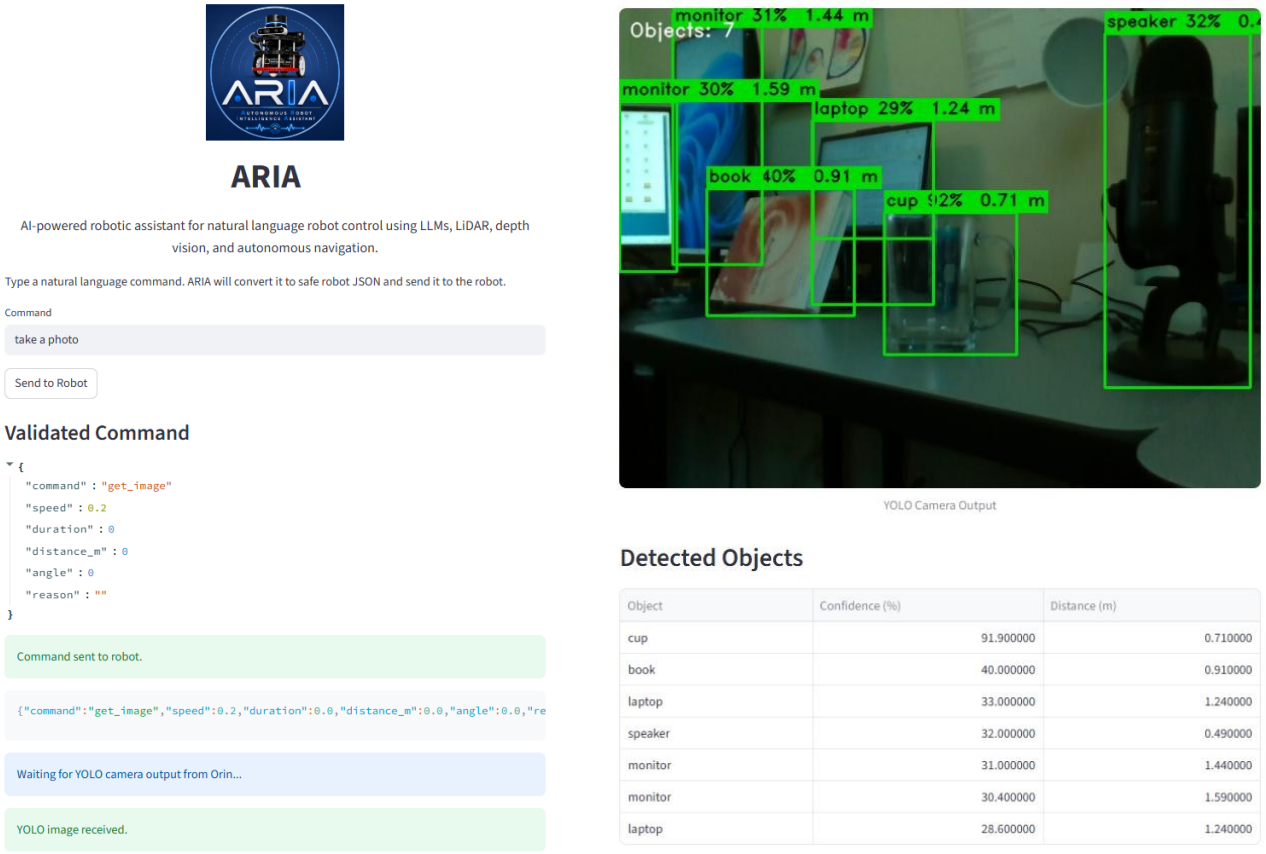

Figure 6. ARIA natural language command requesting an image capture. The robot acquires an image, performs YOLO-based object detection, and returns detected objects with confidence scores and distance estimates.

11. Conclusion

The LLM-NLRC project successfully developed ARIA, a working educational robotics platform that connects conversational AI with physical robot control through a complete, deployable software and hardware stack. The project demonstrates how a modern large language model can serve as an accessible front end to an embedded robotics system, allowing users to interact with a robot through ordinary language while strong validation and safety constraints protect the physical hardware and the classroom environment. The system was designed from the beginning as an educational tool rather than a research prototype: every architectural decision, the modular file structure, the Pydantic schema boundaries, the MQTT decoupled communication, the singleton camera management pattern

provides a concrete teaching example applicable to courses in embedded systems, robotics, computer vision, and AI. The completed system integrates a Jetson Orin Nano, Pololu Romi robot platform, Raspberry Pi 4, 32U4 motor controller, Intel RealSense D435 RGB-D camera, RPLIDAR A1 laser scanner, LiPo battery, and XT60 switching hardware with a three-file Python software stack (app.py, aria_mqtt_receiver.py, camera_yolo_module.py) that implements the full command pipeline from natural language input through AI interpretation, Pydantic validation, MQTT transmission, sensor execution, and real-time visualization. The platform supports seven command types, returns live lidar polar charts and YOLOv8-annotated depth images, and is suitable for immediate classroom deployment in EEE 187 Robotics and EEE 225 Advanced Robotics and Control beginning Fall 2026, as well as for workshops and demonstrations with the Sacramento State Competitive Robotics Club. The project directly fulfills the goals of the Probationary Faculty Development Grant by strengthening instructional innovation in AI and robotics, supporting the professional development of a probationary faculty member, and creating durable educational materials that will benefit multiple student cohorts. It advances Sacramento State's diversity and inclusion mission by lowering the programming barrier to robotics participation and making advanced AI-hardware interaction accessible to first-generation students, transfer students, and students without prior robotics experience. The project also positions the Department of Electrical and Electronic Engineering and the College of Engineering and Computer Science as contributors to campus-wide AI integration goals and establishes a platform for future external funding proposals in human-robot interaction, AI-enhanced pedagogy, and embedded AI systems.

References

Tellex, S., Gopalan, N., Kress-Gazit, H., & Matuszek, C. (2020). Robots that use language. Annual Review of Control, Robotics, and Autonomous Systems, 3, 25-55.

Ahn, M., et al. (2022). Do As I Can, Not As I Say: Grounding language in robotic affordances.