

Development and Integration of a ROSGPT-based Semantic Navigation System with Multi-user Concurrent Tasking under ROS 2

SHENG-YUNG PENG, JIN-SIANG SHAW
Graduate Institute of Manufacturing Technology,
National Taipei University of Technology,
Taipei,
TAIWAN

Abstract: - This paper presents a ROSGPT-based semantic navigation system for multi-user autonomous mobile robot (AMR) operation under ROS 2. An environment-specific prompt converts Traditional Chinese, English, and mixed-language requests into structured MOVE, ROTATE, and GOTO commands. JSON-format constraints, station-name whitelisting, and physical-parameter bounds are applied before command dispatch. A middleware mechanism comprising a first-in, first-out (FIFO) Task Queue, a Worker Thread, and a Dispatcher decouples request admission from cloud-based LLM inference and maintains enqueue-order task processing. Validated GOTO commands are dispatched to the Nav2 Action Server, whereas MOVE and ROTATE commands are executed by an odometry-feedback closed-loop controller. The system was evaluated through three experiments covering semantic parsing, concurrent middleware processing, and continuous physical navigation. For a predefined dataset of 150 commands, the Improved Prompt increased the Exact Match Rate from 7.3% to 100%. In a five-client workload comprising 60 requests, all requests completed semantic parsing and structured-command dispatch without observed middleware-level loss. In addition, 39 of 40 continuous physical navigation tasks were completed successfully, corresponding to a task completion rate of 97.5%. These results provide preliminary evidence of the feasibility of the proposed architecture for multi-user semantic navigation in indoor environments.

Key-Words: - Robot Operating System 2 (ROS 2), Autonomous Mobile Robot (AMR), ROSGPT, Large Language Model (LLM), Prompt Engineering, Semantic Navigation, Multi-User Task Scheduling.

Received: April 12, 2026. Revised: June 19, 2026. Accepted: July 21, 2026. Published: October 20, 2026.

1 Introduction

Autonomous mobile robots (AMRs) have been increasingly deployed in manufacturing, warehousing, indoor logistics, healthcare, and service environments. Compared with conventional automated guided vehicles, AMRs can perform localization, path planning, and obstacle avoidance while adapting their motion to changes in the operating environment [1]. Nevertheless, practical AMR operation commonly requires users to understand map coordinates, navigation goals, velocity parameters, or Robot Operating System (ROS) communication mechanisms. These engineering-oriented interfaces increase the operational barrier for personnel without robotics expertise. Consequently, intuitive interaction and appropriate safety mechanisms have become important requirements for extending robot applications beyond trained system developers [2].

Large language models (LLMs) provide a potential means of reducing this barrier by

interpreting natural-language instructions and translating them into structured robot tasks. Recent studies have investigated the integration of LLMs with robot perception, task planning, communication, and control [3]. ROSGPT demonstrated that ChatGPT can be connected to ROS-based applications as a natural-language interface, allowing user utterances to be transformed into executable robot commands [4]. Other embodied-language studies have addressed the relationship between language-model reasoning and physical robot capabilities. SayCan combines language-model predictions with robot affordance values to select actions that are both linguistically relevant and physically executable [5], whereas Code as Policies generates executable robot policies from natural-language instructions through program synthesis [6]. These studies demonstrate the potential of LLMs for improving human-robot interaction and reducing the need for users to directly manipulate low-level robot interfaces.

However, applying probabilistic LLM outputs to a physical AMR still presents several practical challenges. A generated command may contain an unsupported action, an incomplete data structure, an undefined destination, an inconsistent unit, or a motion parameter that exceeds the configured operating range. Directly transmitting such outputs to the robot-control layer may therefore lead to execution failures or unsafe behavior. Moreover, cloud-based LLM inference introduces variable processing and network delays. When multiple users submit requests within a short period, synchronous processing may block request handling, complicate task ordering, and create competition for shared navigation resources. Existing LLM-robot frameworks mainly emphasize language interpretation or task generation, whereas deterministic command validation, multi-user request scheduling, and physical navigation execution are not always addressed as an integrated middleware problem.

To address these limitations, this study develops a ROSGPT-based semantic navigation system for multi-user AMR operation under ROS 2. The proposed middleware constrains natural-language interpretation to three predefined command types—MOVE, ROTATE, and GOTO—and applies structured-command validation, station-name whitelisting, and physical-parameter constraints before dispatch. A Task Queue, Worker Thread, and Dispatcher mechanism processes requests according to their enqueue order and decouples request admission from cloud-based semantic parsing. Validated commands are subsequently routed according to their execution characteristics: GOTO commands are dispatched to the Nav2 Action Server for map-based navigation, whereas MOVE and ROTATE commands are executed by an odometry-feedback closed-loop controller.

The principal contribution of this study is an end-to-end architecture that connects probabilistic language interpretation with deterministic ROS 2 command execution. In addition to environment-specific prompt engineering, the system integrates deterministic validation, multi-user task scheduling, execution-path separation, and task-status management. The proposed architecture is evaluated through three experiments: multilingual semantic parsing, a five-client concurrent workload, and continuous physical navigation comprising global GOTO tasks and local MOVE or ROTATE adjustments. The results are used to assess structured command generation, request admission, enqueue-order middleware processing, and physical task completion under the evaluated conditions.

2 System Description

The proposed system integrates a ROSGPT-based natural-language interaction mechanism, multi-user task scheduling middleware, deterministic command validation, and two robot execution paths under ROS 2. ROS 2 provides the distributed node-based communication architecture used to connect these functional modules [7]. The following subsections describe the underlying ROSGPT interaction architecture, physical AMR platform, localization and navigation configuration, and experimental environment.

2.1 ROSGPT Architecture

The proposed system is built upon the ROSGPT interaction architecture, which connects natural-language user requests with ROS-compatible robot commands. As illustrated in Figure 1, users provide task instructions through text or voice input. Voice commands are first converted into textual requests before being forwarded to the ROSGPT middleware. This interaction mechanism allows users to describe robot tasks in natural language without directly specifying ROS messages, topics, or low-level motion parameters.

Within the robotic system, the ROSGPT middleware consists primarily of the ROSGPTProxy and ROSParser modules [4]. The ROSGPTProxy constructs the request sent to the cloud-based large language model by combining the user instruction with a predefined system prompt. The system prompt defines the expected role, command vocabulary, and response structure, while the user prompt contains the task requested by the operator. The combined prompt is transmitted to the ChatGPT REST API for semantic interpretation.

The language model returns one or more candidate actions in a structured JSON format. These JSON actions describe the intended robot operation and the associated command parameters. The ROSParser then interprets the returned structure and converts it into commands that can be processed through the ROS or ROS 2 communication layer. The resulting commands are subsequently delivered to the robot hardware for execution.

The original ROSGPT architecture therefore provides the fundamental connection between natural-language interaction, cloud-based language-model inference, structured action generation, and ROS-compatible execution. However, this basic interaction chain does not explicitly address multi-user request scheduling, deterministic command validation, semantic-station restrictions, or differentiated execution paths for map-based

navigation and local motion. In this study, these functions are added around the ROSGPT interaction mechanism through the Task Queue and Dispatcher, validation modules, and command-routing architecture described in Section 3.

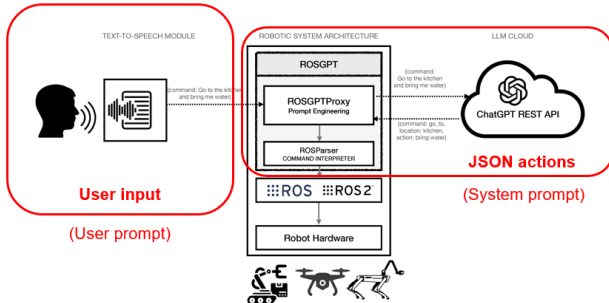


Fig. 1: ROSGPT architecture [4]
Source: [4]

2.2 Physical AMR Platform and Hardware

The physical AMR platform and its primary hardware components are shown in Figure 2. The experimental system was implemented using Ubuntu 24.04 and ROS 2 Jazzy on the robot computing platform. A wireless network connection was used to communicate with the cloud-based LLM service and to support the web-based command interface.

The sensing system consists of a Velodyne VLP-16 three-dimensional LiDAR, Hokuyo two-dimensional laser scanners, and a CH110 inertial measurement unit. The VLP-16 provides three-dimensional point-cloud measurements of the surrounding environment, while the inertial measurement unit supplies angular velocity and acceleration information for LiDAR-inertial odometry. These measurements are jointly processed by DLIO to estimate continuous robot motion and to support map construction [8].

The Hokuyo laser scanners provide two-dimensional range measurements for map-based localization and obstacle representation. Their scan data are processed and supplied to the ROS 2 navigation stack for AMCL localization and costmap updates. This sensing configuration enables the AMR to maintain continuous local motion estimation while correcting its global pose relative to the prebuilt map.

The computing platform executes the semantic middleware, Task Queue, Worker Thread, and Dispatcher, validation modules, Command Router, localization processes, and navigation components. Velocity commands generated by Nav2 or the OdomController are published through the ROS 2 control interface to drive the physical platform. This hardware configuration therefore supports both

map-based GOTO navigation and odometry-feedback MOVE and ROTATE execution within the same AMR system.

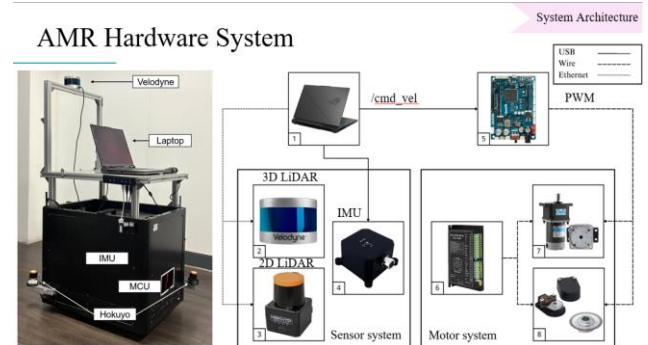


Fig. 2: Hardware architecture and physical layout of the AMR platform.
Source: created by the authors

2.3 Localization and Navigation Architecture

The navigation subsystem combines LiDAR-inertial odometry, map-based localization, global path planning, and local motion control. The coordinate relationship follows the map → odom → base_link hierarchy. The map frame provides a globally consistent reference for navigation goals, the odom frame preserves locally continuous motion, and the base_link frame represents the current pose of the AMR body.

DLIO fuses measurements from the three-dimensional LiDAR and inertial measurement unit to provide continuous odometry and support map construction [8]. Its output estimates the short-term motion of the AMR in the odom frame. Because odometric estimates may gradually accumulate drift during extended operation, AMCL uses the prebuilt two-dimensional map and laser-scan observations to estimate the global robot pose and correct the relationship between the map and odom frames [9]. In this configuration, DLIO provides locally continuous motion estimation, whereas AMCL maintains global consistency relative to the navigation map.

Map-based GOTO commands are executed through the Navigation2 framework [10]. Nav2 coordinates global planning, local control, recovery behavior, navigation feedback, and action execution through a modular behavior-tree structure [11]. After receiving a validated target pose in the map frame, the navigation stack generates and follows a collision-free route from the current robot position to the specified destination.

Theta* is adopted as the global planner because it supports any-angle path generation without restricting the resulting path to grid edges [12]. Velocity-space methods such as the Dynamic

Window Approach represent a classical form of reactive local collision avoidance in which dynamically admissible translational and rotational velocities are evaluated [13]. The present system does not use DWA; instead, MPPI is adopted as the local controller to evaluate sampled control sequences according to the global path, robot state, and costmap information [14]. MPPI continuously generates velocity commands for path tracking and local obstacle avoidance.

MOVE and ROTATE commands bypass the global path-planning process because they specify local displacement or angular motion rather than a semantic destination. These commands are executed by the OdomController using DLIO odometry feedback. This separation allows global destination navigation and local geometric motion to operate through control mechanisms appropriate to their respective execution requirements.

2.4 Experimental Environment

The physical experiments were conducted in an indoor corridor environment containing rooms, intersections, and predefined semantic destinations. Figure 3 presents the two-dimensional map used for localization and navigation. The map was constructed using LiDAR-inertial measurements and was subsequently used by AMCL and Nav2 as the global reference for robot pose estimation, path planning, and destination assignment.

Semantic stations were configured in a station file, with each station associated with a predefined pose in the map frame. When a user issued a GOTO command, the parsed station name was checked against the station whitelist and converted into the corresponding position and orientation. This configuration allowed users to specify destinations through meaningful place names rather than manually entering map coordinates.

The experimental environment included static structures represented in the prebuilt map as well as objects that were not previously included in the map. During navigation, laser-scan observations were continuously incorporated into the local costmap so that the AMR could respond to nearby obstacles while following the global route. The same environment was also used to evaluate local MOVE and ROTATE commands through DLIO-based odometry feedback.

A wireless network connected the web-based interface and semantic middleware to the cloud-based LLM service. Therefore, the experimental setup included both physical navigation uncertainties and variable cloud communication and inference delays. This environment provided a

practical basis for evaluating semantic command interpretation, concurrent request processing, closed-loop local motion, and continuous physical navigation under consistent indoor operating conditions.

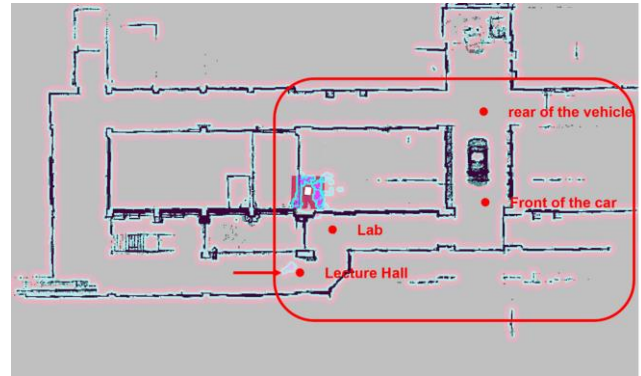


Fig. 3: Experimental 2D map and the four semantic stations.

Source: created by the authors

3 Methodology

3.1 Improved Prompt Engineering and Semantic Parsing

Within the ROSGPT interaction chain described in Section 2.1, the ROSGPTProxy was customized using an environment-specific Improved Prompt Engineering framework. Rather than retraining or fine-tuning the language model, the proposed approach constrains semantic interpretation through a structured system prompt designed for indoor AMR navigation. Prompt engineering provides a practical method for guiding LLM behavior through task instructions, output-format restrictions, and representative examples [15]. The proposed semantic parsing, deterministic validation, and command-routing workflow is illustrated in Figure 4.

The framework applies six alignment strategies. First, role and context assignment defines the LLM as an AMR command-generation module and instructs it to transform user requests rather than produce conversational responses. Second, a strict output-format constraint requires the response to contain only a JSON array, without Markdown syntax, explanatory text, or additional natural-language content. This restriction improves compatibility with the subsequent parsing process.

Third, command-ontology narrowing limits the expected output to three predefined command classes: MOVE, ROTATE, and GOTO. The prompt therefore guides the model toward the supported

robot capabilities instead of allowing arbitrary action descriptions. Fourth, dynamic semantic-station grounding inserts the currently available station names into the system prompt. Destination expressions are consequently expected to be mapped to predefined station identifiers rather than to model-generated coordinates or unsupported locations.

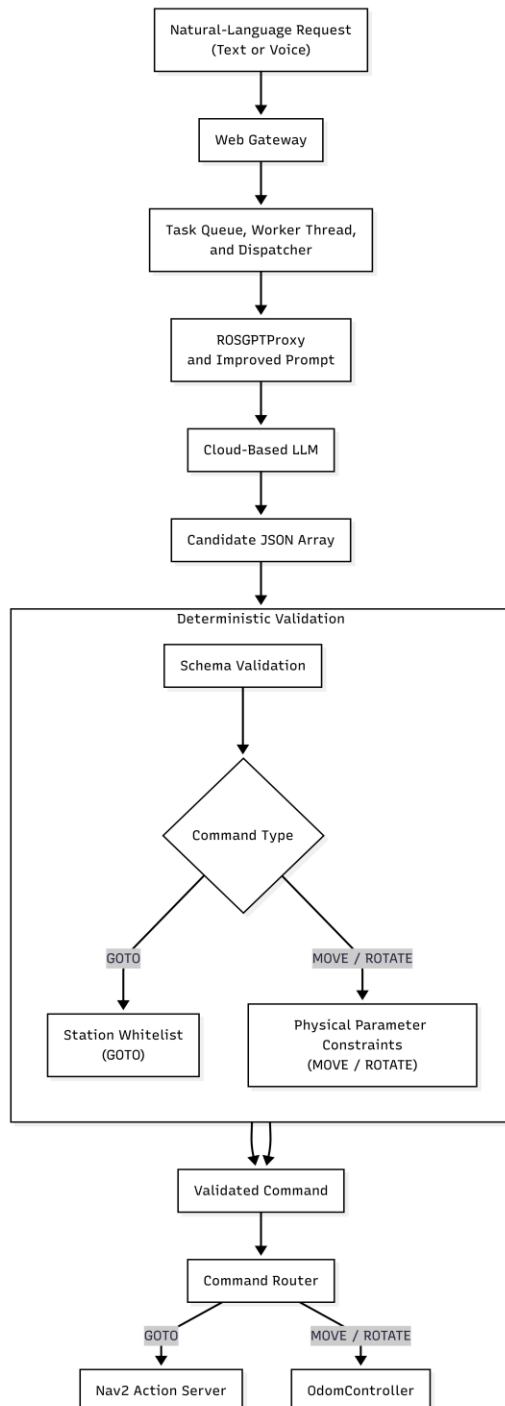


Fig. 4: Workflow of improved prompt-based semantic parsing, deterministic validation, and command routing.

Source: created by the authors

Fifth, few-shot prompting and structural anchoring provide representative input–output examples for individual and compound instructions. These examples specify the expected field names, JSON hierarchy, command order, and parameter structure. Sixth, vocabulary and unit normalization maps Traditional Chinese, English, and mixed-language expressions to canonical command fields. Directional expressions, distance units, and angular units are normalized according to the command format required by the middleware. For ROTATE commands, positive angular values are defined as clockwise rotation at the semantic-command level.

As shown in Figure 4, a natural-language request is received through the Web Gateway and processed according to the enqueue order maintained by the Task Queue and Worker Thread. The request is then combined with the customized system prompt and submitted to the cloud-based LLM through the ROSGPTProxy. The returned JSON array is treated as a set of candidate commands rather than as immediately executable robot instructions. Each candidate command is subsequently passed to the deterministic validation layer described in Section 3.2.

3.2 Command Ontology and Deterministic Validation

The JSON array generated by the semantic parsing module is treated as a set of candidate commands and must pass deterministic validation before entering the robot execution layer. The validation mechanism separates probabilistic language interpretation from physical command execution by enforcing a predefined data schema, command ontology, semantic constraints, and physical operating limits. As shown in Figure 4, deterministic validation is positioned between candidate JSON generation and command routing.

After JSON deserialization, each candidate object is checked against its corresponding schema. The validation process verifies the command type, required fields, data types, and parameter completeness. Missing parameters, unsupported action names, malformed structures, and incompatible field combinations cause the command to be rejected.

The command ontology contains three supported action classes: MOVE, ROTATE, and GOTO. A MOVE command specifies travel distance, linear velocity, and motion direction. A ROTATE command specifies rotation angle, angular velocity, and rotation direction. A GOTO command identifies either a predefined semantic station or a structured target pose in the map frame.

Commands outside these three classes are not accepted by the middleware, even when the generated output appears semantically meaningful.

Compound user requests are represented as ordered command objects in the JSON array. Each command object is validated independently before the complete sequence is passed to the dispatch stage.

Semantic validation is subsequently applied according to the command type. For GOTO commands, station-based destinations must match an identifier in the predefined station whitelist. A valid station identifier is converted into its stored position and orientation in the map frame, whereas an undefined or misspelled station identifier is rejected before the command reaches the Nav2 Action Server. This mechanism prevents undefined semantic destinations from being dispatched as navigation goals.

For MOVE and ROTATE commands, normalized motion parameters are checked against the operating constraints defined in the middleware. Distance, angle, linear velocity, and angular velocity values are converted to the units required by the execution controller. In this study, a positive angle denotes clockwise rotation. When an angle specification conflicts with the associated direction field, the command is rejected as internally inconsistent. Structurally valid parameters that exceed the configured operating range are clamped to the corresponding safety bounds rather than being forwarded without restriction.

Only commands that successfully complete schema, semantic, and motion-parameter validation are marked as validated commands and passed to the Command Router. GOTO commands are subsequently directed to Nav2, whereas MOVE and ROTATE commands are sent to the OdomController. This deterministic validation layer prevents unvalidated LLM-generated outputs from directly invoking unsupported actions, undefined destinations, or unconstrained robot motion.

3.3 Web-based Semantic Command Interface

A web-based semantic command interface was developed to provide users with direct access to the ROSGPT middleware without requiring knowledge of ROS 2 topics, services, actions, or command-line tools. As shown in Figure 5, users submit natural-language requests through the interface, and the corresponding structured command result is presented after semantic parsing. The interface accepts Traditional Chinese, English, and mixed-language instructions supported by the Improved Prompt Engineering framework.

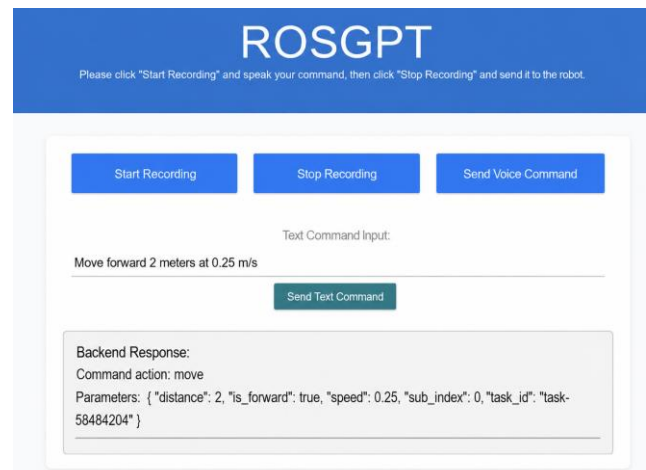


Fig. 5: Web-based interface for submitting natural-language commands and viewing structured semantic parsing results.

Source: created by the authors

After a request is submitted, the Web Gateway receives the HTTP message and encapsulates the user instruction as a middleware task. The task is then inserted into the Task Queue for subsequent semantic parsing, deterministic validation, and command dispatch. This design separates the user-facing interface from cloud-based LLM inference and physical robot execution. Consequently, the front end does not communicate directly with Nav2, the OdomController, or the robot hardware.

The interface provides a visible representation of the transformation from natural-language input to the structured MOVE, ROTATE, or GOTO command. Displaying the parsed result allows the command type and normalized parameters to be inspected through the interface. For station-based GOTO requests, the returned result contains the semantic destination identifier rather than requiring the user to enter coordinates manually. For MOVE and ROTATE requests, the structured result presents the motion parameters generated according to the predefined command ontology.

The task state returned through the Web Gateway represents the processing progress of the semantic middleware. In accordance with the terminology used in this study, the done state indicates that semantic parsing, deterministic validation, and command dispatch have been completed. It does not indicate that the AMR has completed the corresponding physical movement or reached the navigation destination. Physical navigation completion is evaluated separately through the execution subsystem.

The web interface therefore serves as an interaction and observation layer between users and the semantic middleware. It simplifies multilingual

command submission while preserving the separation between probabilistic language interpretation, deterministic command validation, and ROS 2-based robot execution.

3.4 Concurrent Task Queue and Scheduling

Cloud-based LLM inference introduces variable processing and communication delays. Processing each HTTP request synchronously would cause the Web Gateway to remain occupied until semantic parsing and command dispatch were completed, reducing its ability to accept requests from multiple users. To separate request admission from time-consuming LLM processing, the proposed middleware adopts a producer-consumer architecture composed of the Web Gateway, Task Queue, Worker Thread, and Dispatcher.

The Web Gateway acts as the producer. After receiving a natural-language request, it creates a task object containing the user instruction, task identifier, timestamps, and initial processing state. The task is then inserted into the Task Queue and assigned the pending state. Because task creation and enqueueing do not require completion of cloud-based semantic parsing, the gateway can rapidly return an admission response and continue accepting subsequent requests.

An independent Worker Thread acts as the consumer and retrieves tasks according to the first-in, first-out (FIFO) principle. FIFO scheduling is applied according to the order in which tasks are successfully enqueued; it does not imply that requests are processed according to the exact time at which different users begin transmitting them. After a task is removed from the queue, its state is changed to processing, and the associated natural-language instruction is submitted to the ROSGPTProxy for semantic parsing.

The candidate JSON output is subsequently passed through the deterministic validation mechanism described in Section 3.2. If the command is valid, the Dispatcher forwards it to the Command Router and updates the middleware task state through the remaining dispatch stages. The done state indicates that semantic parsing, validation, and command dispatch have been completed. It does not represent completion of the corresponding physical robot motion. A task is assigned the failed state when parsing, validation, or middleware dispatch cannot be completed.

Shared queue data, task-state records, and dispatch operations are protected using mutual-exclusion mechanisms, including Mutex and reentrant lock (RLock) structures. These synchronization mechanisms prevent multiple

request-handling operations from simultaneously modifying shared middleware resources. The RLock also permits controlled nested access by the same execution thread when task-state updates and dispatch procedures invoke protected operations internally.

This scheduling architecture preserves ordered middleware processing while preventing cloud inference latency from blocking request admission. It also enables request admission latency and request-to-dispatch time to be measured separately during the concurrent workload experiment. After validation and scheduling are completed, the Command Router selects the appropriate execution path as described in Section 3.5.

3.5 Command Routing and Closed-Loop Execution

After a command has completed semantic parsing, deterministic validation, and queue scheduling, the Command Router selects the execution subsystem according to the validated action type. This routing mechanism separates map-based destination navigation from local geometric motion. GOTO commands are dispatched to the Nav2 Action Server, whereas MOVE and ROTATE commands are forwarded to the OdomController.

For a station-based GOTO command, the validated station identifier is first converted into its predefined position and orientation in the map frame. The resulting target pose is then submitted to Nav2 as a navigation goal. Nav2 coordinates localization, global path planning, local trajectory control, obstacle avoidance, and recovery behavior during execution [10]. Theta* generates the global route, while MPPI produces velocity commands for path tracking and local obstacle avoidance. Navigation feedback and the final action result are used to determine the physical execution status independently of the middleware done state.

MOVE and ROTATE commands bypass the global planner because they specify local displacement or angular motion rather than a semantic destination. These commands are executed by the OdomController using feedback from the DLIO odometry topic [8]. The controller operates at a fixed frequency and continuously publishes velocity commands through the ROS 2 velocity-control interface. Closed-loop feedback is used instead of duration-based open-loop motion so that execution is determined by the measured robot pose rather than by an assumed travel time.

For MOVE execution, the controller records the initial position and orientation and evaluates the robot displacement relative to the initial heading.

The traveled distance is calculated through longitudinal projection, while lateral deviation and heading error are monitored as additional convergence conditions. The commanded linear velocity is reduced as the remaining distance decreases, and an angular correction is applied to maintain the initial motion direction. The command is completed only when the projected displacement, lateral error, and heading error satisfy the predefined tolerances.

For ROTATE execution, the target heading is calculated from the initial yaw and the requested angular displacement. In accordance with the semantic-command convention used in this study, a positive angle represents clockwise rotation. This semantic-level convention is adopted to maintain consistency between the angle sign and the explicit rotation-direction field and is distinct from the standard ROS yaw convention, in which positive yaw represents counterclockwise rotation. This convention is used consistently when calculating the target heading and normalized yaw error in the OdomController. The controller continuously adjusts the angular velocity as the robot approaches the target orientation. Execution terminates when the remaining angular error falls within the configured tolerance.

The OdomController also monitors execution time and motion progress. A command is stopped and reported as unsuccessful when the controller detects prolonged stagnation, inconsistent direction information, or an execution timeout. When the convergence conditions are satisfied, a zero-velocity command is published to stop the robot. This closed-loop execution architecture provides separate control paths for global semantic navigation and local motion while maintaining feedback-based completion criteria for both command categories.

4 Results and Discussion

4.1 Experimental Design and Evaluation Metrics

The proposed system was evaluated through three experiments covering semantic interpretation, multi-user middleware processing, and continuous physical navigation. The semantic parsing experiment was conducted at the software level; the concurrent workload experiment evaluated the deployed middleware without requiring physical robot motion; and the navigation experiment was performed using the ROS 2 Jazzy-based AMR platform and indoor experimental environment

described in Section 2. The cloud-based LLM was accessed through an API, while the middleware, station configuration, command ontology, and robot-execution architecture remained unchanged except for the prompt configuration intentionally compared in the semantic parsing experiment.

The first experiment evaluated the effectiveness of Improved Prompt Engineering through a comparative software-level test. A predefined dataset of 150 commands was constructed to represent indoor AMR operating expressions in Traditional Chinese, English, and mixed-language forms. The dataset included individual MOVE, ROTATE, and GOTO instructions as well as compound command sequences. Each command was processed using both the Baseline Prompt and the Improved Prompt, and the generated outputs were evaluated using three hierarchical metrics: Valid JSON Rate, Intent Match Rate, and Exact Match Rate.

The Valid JSON Rate measures whether the generated output can be successfully deserialized by a standard JSON parser without Markdown wrappers or additional natural-language explanations. The Intent Match Rate evaluates whether the generated action type correctly represents the user intention and belongs to the predefined MOVE, ROTATE, or GOTO ontology. The Exact Match Rate (EMR) applies the strictest criterion: the number and order of command objects, action types, and all required normalized parameters—including displacement, rotation angle, direction, velocity-related fields, and semantic station identifiers—must exactly match the predefined ground truth. Unit-conversion errors, reversed directions, incorrect station mappings, missing fields, additional fields, or incorrect command ordering are counted as failures.

The second experiment evaluated the Task Queue, Worker Thread, and Dispatcher under a multi-user concurrent workload. A Python test program simulated five independent clients that submitted a total of 60 natural-language requests over a 300-s test period. Middleware event logs were used to trace each task from HTTP request arrival through structured-command dispatch. The principal metrics were task drop rate, Request Admission Latency, and Request-to-Dispatch Time. In the event logs, `queue_latency` represents the elapsed time from HTTP request arrival at the Web Gateway to successful task creation and enqueueing, whereas `response_time` represents the elapsed time from HTTP request arrival to completion of structured-command dispatch. Neither metric includes the duration of physical robot motion.

The third experiment performed continuous physical validation using 40 sequential navigation tasks among four predefined semantic stations. The tasks contained global GOTO navigation and, where specified, local MOVE or ROTATE adjustments. For this experiment, the maximum displacement of an individual MOVE command was limited to 1.0 m, and the angular velocity of ROTATE commands was limited to 0.3 rad/s. The primary metric was the physical task completion rate, supplemented by qualitative observations of obstacle avoidance and safe command execution. This metric was evaluated separately from the middleware done state, which only indicates that semantic parsing, validation, and command dispatch have been completed.

4.2 Semantic Parsing Results

The Improved Prompt produced substantially more consistent semantic parsing results than the Baseline Prompt on the predefined 150-command dataset. Under the baseline configuration, the Intent Match Rate reached 98.0% (147/150), indicating that the LLM could generally identify the requested MOVE, ROTATE, or GOTO action. However, the Exact Match Rate was only 7.3% (11/150). Therefore, recognizing the general action intent alone was insufficient for producing commands that satisfied the middleware's strict structural and parameter requirements. The substantial gap between the 98.0% Intent Match Rate and the 7.3% Exact Match Rate demonstrates that high-level semantic understanding does not guarantee command-form compliance for robot execution.

The large difference between intent recognition and exact matching was mainly associated with parameter completion and output-structure inconsistencies. When users omitted velocity information in common expressions such as "move forward two meters," the Baseline Prompt could generate missing, null, or unsuitable default values. Some outputs also introduced fields not defined by the command schema, such as additional unit attributes. For compound instructions, the baseline occasionally generated nested sequence structures instead of the ordered JSON command array required by the middleware. Station-based requests could similarly retain conversational destination expressions rather than mapping them to the identifiers defined in the station configuration.

After the six alignment strategies described in Section 3.1 were applied, all 150 outputs generated using the Improved Prompt satisfied the Valid JSON, Intent Match, and Exact Match criteria, corresponding to rates of 100%. Strict output-format constraints reduced additional explanatory text and

incompatible structures, while command-ontology narrowing restricted the generated actions to MOVE, ROTATE, and GOTO. Few-shot structural examples guided the completion of omitted parameters according to the predefined command format, and dynamic station grounding mapped destination expressions to valid station identifiers.

The increase in Exact Match Rate from 7.3% to 100% demonstrates that the environment-specific prompt substantially improved the consistency between natural-language requests and the middleware command representation within the evaluated dataset. In particular, the results show that a high Intent Match Rate does not necessarily indicate that an LLM output is directly suitable for physical execution; parameter values, field names, command hierarchy, units, and station identifiers must also match the expected schema.

Nevertheless, the 100% result applies only to the predefined Traditional Chinese, English, and mixed-language command set used in this experiment. It should not be interpreted as a guarantee of error-free parsing for arbitrary user expressions, unseen environments, or unrestricted robot actions. The deterministic validation layer described in Section 3.2 therefore remains necessary even when the Improved Prompt achieves complete matching on the evaluated dataset.

4.3 Concurrent Workload Results

The concurrent workload experiment evaluated whether the Task Queue and Dispatcher could continue admitting requests while the Worker Thread performed cloud-based semantic parsing. Five simulated clients submitted a total of 60 requests over a 300-s test period. Event logs confirmed that all 60 requests completed semantic parsing and structured-command dispatch. No request or dispatch loss was observed at the middleware level, corresponding to an observed task drop rate of 0% under the evaluated workload.

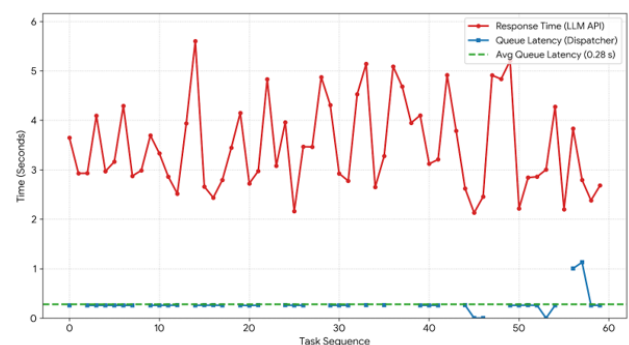


Fig. 6: Latency comparison under the five-client concurrent workload.

Source: created by the authors

Figure 6 compares the two latency measurements recorded for the 60 tasks. In the original figure, “Queue Latency (Dispatcher)” corresponds to the Request Admission Latency defined in this study, whereas “Response Time (LLM API)” corresponds to Request-to-Dispatch Time. Request Admission Latency, recorded in the event logs as `queue_latency`, remained near 0.25–0.26 s for most tasks, although a small number of deviations were observed near the end of the task sequence. Overall, 95% of the tasks were successfully created and enqueued within 0.261 s after the HTTP request reached the Web Gateway. This result indicates that incoming requests could generally be admitted without waiting for completion of the preceding semantic-processing procedure.

In contrast, Request-to-Dispatch Time, recorded as `response_time`, varied from approximately 2.1 to 5.14 s. This metric includes request admission, queue handling, cloud-based LLM inference, deterministic validation, and structured-command dispatch, but excludes the subsequent physical movement of the AMR. The observed variation is consistent with the combined effects of cloud inference, network communication, queue processing, and validation, although their individual contributions were not measured separately.

The substantial separation between the two latency measurements demonstrates the intended decoupling effect of the producer–consumer architecture. Although individual tasks required different amounts of time to reach the dispatch stage, the Web Gateway could continue accepting and enqueueing subsequent requests while previously admitted tasks were processed by the Worker Thread.

These results provide preliminary evidence that the asynchronous Task Queue, Worker Thread, and Dispatcher can support multi-user request admission and enqueue-order middleware processing under the evaluated five-client workload. However, the absence of observed loss in 60 requests should not be interpreted as a guarantee of loss-free operation under unlimited users, higher arrival rates, network failures, or substantially longer cloud-service delays.

Because the current architecture uses a single Worker Thread to process queued requests sequentially, its scalability is primarily limited by the effective task-processing rate. When the request arrival rate remains below the processing rate, the queue can continue to process incoming tasks without persistent accumulation. However, as the arrival rate approaches or exceeds the processing

rate, queue waiting time and Request-to-Dispatch Time are expected to increase, even if Request Admission Latency remains low. Therefore, larger workloads may require additional semantic-processing workers or other parallel-processing mechanisms while maintaining controlled command dispatch to the AMR.

4.4 Continuous Physical Navigation Results

The end-to-end physical evaluation consisted of 40 continuous navigation tasks among the four predefined semantic stations shown in Figure 3. Each task began with a natural-language request and included semantic parsing, deterministic validation, command dispatch, and physical execution by the AMR. Of the 40 submitted tasks, 39 were completed successfully, corresponding to an observed task completion rate of 97.5% in the evaluated indoor environment.

The tasks were distributed among the four semantic stations shown in Figure 3: “Lab,” “Lecture Hall,” “Front of the car,” and “rear of the vehicle.” All 9 tasks directed to “Lab,” all 13 tasks directed to “Lecture Hall,” and all 7 tasks directed to “rear of the vehicle” were completed. For “Front of the car,” 10 of 11 tasks were completed. The resulting station-specific outcomes were therefore 9/9, 13/13, 10/11, and 7/7, respectively.

The only unsuccessful task produced the station identifier `front_of_the_car`, which did not match the whitelist identifier corresponding to the displayed station name “Front of the car.” The command was therefore rejected during semantic-station validation and terminated before a navigation goal was dispatched to the Nav2 Action Server. This event is counted as an end-to-end system task failure because the requested physical navigation was not completed. However, the failure also demonstrates the protective role of deterministic validation: the unmatched destination did not generate an arbitrary navigation target or cause unintended robot motion.

Across the 40 submitted tasks, the mean end-to-end task time was approximately 52.53 s. The mean cloud-based semantic parsing stage accounted for approximately 2.79 s, whereas the mean physical execution stage accounted for approximately 49.74 s. The physical execution time was substantially longer because it included travel between semantic stations and, where specified, subsequent local MOVE or ROTATE adjustments. Differences among tasks were therefore influenced primarily by route length, target location, and the required physical motion rather than by semantic parsing alone.

During the successfully dispatched tasks, the navigation stack used AMCL-based global localization, Theta* global path generation, and MPPI local control to execute the GOTO portions of the commands. Local MOVE and ROTATE operations were completed through the DLIO-feedback OdomController. These results provide preliminary end-to-end validation that the proposed middleware can connect multilingual natural-language requests with both map-based navigation and local closed-loop motion in the evaluated corridor environment.

Nevertheless, the observed 39/40 outcome should not be generalized as a universal reliability guarantee. The experiment was conducted using a fixed indoor map, a predefined station set, one physical AMR, and a limited number of continuous tasks. The failure case also indicates that consistency between prompt-injected station names, parser output, and canonical station identifiers remains critical to successful semantic navigation.

4.5 Discussion

The principal results of the three system-level experiments are summarized in Table 1. Collectively, the results indicate that the proposed architecture can connect multilingual natural-language interaction with structured command generation, multi-user middleware processing, and physical AMR execution under the evaluated conditions. However, each experiment assesses a different part of the system and should therefore be interpreted using its corresponding evaluation scope.

Table 1. Summary of the experimental results

Experiment	Test condition	Principal result
Semantic parsing	150 multilingual commands	Exact Match Rate increased from 7.3% to 100%
Concurrent workload	Five simulated clients; 60 requests over 300 s	60/60 requests parsed and dispatched
Continuous physical navigation	40 sequential tasks among four semantic stations	39/40 tasks completed (97.5%)

Source: created by the authors

The semantic parsing results demonstrate that high intent recognition alone is insufficient for executable robot-command generation, because semantic understanding does not guarantee compliance with the required command structure and parameters. Although the Baseline Prompt achieved a high Intent Match Rate, most outputs did not exactly satisfy the required fields, normalized

parameters, station identifiers, or JSON structure. The Improved Prompt reduced these inconsistencies by applying the six alignment strategies described in Section 3.1. Nevertheless, prompt engineering remains a probabilistic guidance mechanism rather than a deterministic safety guarantee. The validation layer is still required to reject malformed commands, undefined stations, and internally inconsistent parameters before physical execution.

The concurrent workload results demonstrate the decoupling effect of the Task Queue, Worker Thread, and Dispatcher. Request Admission Latency remained substantially lower than Request-to-Dispatch Time for most tasks, indicating that request admission was generally decoupled from the longer semantic-processing and dispatch procedure. This separation allowed the Web Gateway to continue creating and enqueueing tasks without waiting for previously admitted requests to complete semantic processing. However, the experiment evaluated only five simulated clients and 60 requests. The observed 0% middleware task drop rate therefore represents the tested workload and does not establish the maximum supported arrival rate, queue capacity, or behavior under prolonged network or cloud-service failures. In particular, the single-worker architecture may become a processing bottleneck when the request arrival rate exceeds the rate at which semantic parsing and dispatch can be completed. Under such conditions, requests would remain in the FIFO queue for longer periods, increasing queue waiting time and overall Request-to-Dispatch Time. Future scalability evaluation should therefore include larger client populations, higher request arrival rates, and measurements of queue growth and worst-case latency under sustained workloads.

The continuous physical navigation experiment further demonstrates that the middleware can connect validated semantic commands to two different execution paths. GOTO commands were executed through Nav2 for map-based navigation, whereas MOVE and ROTATE commands were completed through odometry-feedback closed-loop control. The only unsuccessful task was rejected because the generated station identifier did not match the canonical station whitelist. From an end-to-end perspective, the task was not completed; from a safety perspective, however, the validation mechanism prevented an undefined destination from being converted into unintended robot motion. This result also reveals that consistency among the prompt-injected station vocabulary, parser output, and stored station identifiers remains an important system dependency.

Several limitations should be considered when interpreting the results. The semantic test used a predefined dataset, the concurrent workload was limited in scale and duration, and the physical experiments were conducted using one AMR, one indoor map, and a fixed set of semantic stations. The individual contributions of network communication, cloud-based LLM inference, queue processing, deterministic validation, and command dispatch to Request-to-Dispatch Time were not measured separately. The results therefore provide preliminary evidence of feasibility rather than a general reliability guarantee for unrestricted language, larger user populations, dynamic station vocabularies, multi-map operation, or multi-robot deployment.

5 Conclusion

This study developed and experimentally evaluated a ROSGPT-based semantic navigation system for multi-user AMR operation under ROS 2. The proposed architecture extends the basic ROSGPT interaction mechanism by integrating environment-specific Improved Prompt Engineering, a constrained MOVE, ROTATE, and GOTO command ontology, deterministic command validation, a Task Queue, Worker Thread, and Dispatcher, and differentiated execution paths for map-based navigation and local closed-loop motion. GOTO commands are executed through the Nav2 Action Server, whereas MOVE and ROTATE commands are handled by the DLIO-feedback OdomController.

The semantic parsing experiment showed that the Improved Prompt increased the Exact Match Rate from 7.3% to 100% on the predefined 150-command multilingual dataset. In the concurrent workload experiment, five simulated clients submitted 60 requests over 300 s, and all requests completed semantic parsing and structured-command dispatch without observed middleware-level loss. Request Admission Latency remained substantially lower than Request-to-Dispatch Time for most tasks, indicating that request admission was generally decoupled from the longer semantic-processing and dispatch procedure. In the end-to-end physical evaluation, 39 of 40 continuous navigation tasks were completed successfully, corresponding to an observed task completion rate of 97.5%. The unsuccessful task was rejected by the station whitelist before unintended navigation could occur.

These results provide preliminary evidence that probabilistic natural-language interpretation can be

integrated with deterministic validation, asynchronous task scheduling, and feedback-based physical execution in an indoor AMR system. Nevertheless, the conclusions are limited to the evaluated command dataset, five-client workload, fixed semantic station set, single indoor map, and single physical AMR. Future work will investigate larger and more intensive concurrent workloads, dynamic semantic-station management, local LLM deployment, physical execution-status feedback to the web interface, multi-map navigation, and coordinated multi-robot operation.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, the author(s) utilized artificial intelligence (AI) technologies solely for the purpose of linguistic refinement and grammatical correction. Following the application of these tools, the author(s) comprehensively reviewed, verified, and edited the resulting text. The author(s) maintain full accountability and responsibility for the accuracy, integrity, and final content of the publication.

References:

- [1] G. Fragapane, R. De Koster, F. Sgarbossa, and J. O. Strandhagen, "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda," *European Journal of Operational Research*, Vol. 294, No. 2, 2021, pp. 405–426. DOI: 10.1016/j.ejor.2021.01.019.
- [2] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, Vol. 55, 2018, pp. 248–266. DOI: 10.1016/j.mechatronics.2018.02.009.
- [3] Y. Kim, D. Kim, J. Choi, J. Park, N. Oh, and D. Park, "A survey on integration of large language models with intelligent robots," *Intelligent Service Robotics*, Vol. 17, No. 5, 2024, pp. 1091–1107. DOI: 10.48550/arXiv.2404.09228.
- [4] A. Koubaa, A. Ammar, and W. Boulila, "Next-generation human-robot interaction with ChatGPT and robot operating system," *Software: Practice and Experience*, Vol. 55, No. 2, 2025, pp. 355–382. DOI: 10.1002/spe.3377.
- [5] M. Ahn et al., "Do as i can, not as i say: Grounding language in robotic affordances," *Proceedings of The 6th Conference on Robot*

Learning, Auckland, New Zealand, 2023: PMLR Vol. 205, pp. 287-318.

- [6] J. Liang et al., "Code as policies: Language model programs for embodied control," in 2023 IEEE International Conference on Robotics and Automation (ICRA), London, England, 2023: IEEE, pp. 9493–9500.
- [7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, Vol. 7, No. 66, 2022, p. eabm6074. DOI: 10.1126/scirobotics.abm6074.
- [8] K. Chen, R. Nemiroff, and B. T. Lopez, "Direct lidar-inertial odometry: Lightweight lio with continuous-time motion correction," in 2023 IEEE International Conference on Robotics and Automation (ICRA), London, England, 2023: IEEE, pp. 3983–3989.
- [9] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *Proceedings of the AAAI Conference on Artificial Intelligence*, Orlando, Florida, USA, 1999: American Association for Artificial Intelligence, pp. 343–349.
- [10] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, Nevada, USA, 2020: IEEE, pp. 2718–2725.
- [11] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. ISBN: 0367571331, CRC Press, 2020.
- [12] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal of Artificial Intelligence Research*, Vol. 39, No. 1, 2010, pp. 533–579. DOI: 10.1613/jair.2994.
- [13] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, Vol. 4, No. 1, 1997, pp. 23–33. DOI: 10.1109/100.580977.
- [14] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Transactions on Robotics*, Vol. 34, No. 6, 2018, pp. 1603–1622. DOI: 10.1109/TRO.2018.2865891.
- [15] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language

models: Techniques and applications," *arXiv preprint arXiv:2402.07927*, Vol. 1, 2024. DOI: 10.48550/arXiv.2402.07927.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

- S.-Y. Peng designed and implemented the ROSGPT-based semantic navigation system, developed the prompt-engineering, deterministic-validation, multi-user task-scheduling, command-routing, and closed-loop execution modules, integrated the system with Nav2 and the physical AMR platform, conducted the experiments, analyzed the results, and wrote the manuscript.
- J.S. Shaw conceived the experiment and reviewed the manuscript.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

This work was supported by the National Science and Technology Council, Taiwan (NSTC 115-2221-E-027-146).

Conflict of Interest

The authors declare that they have no conflict of interest.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0

https://creativecommons.org/licenses/by/4.0/deed.en_US