On-Device Vision Language Model Based Natural Language Mission Execution Framework

Seunggyu Lee
Department of Aerospace Mechanical Engineering
Korea Aerospace University
Goyang-Si, South Korea
ruga1026@kau.kr

Seungjun Oh
Department of Aerospace Mechanical Engineering
Korea Aerospace University
Goyang-Si, South Korea
awana970630@kau.kr

Sangwoo Jeon
Department of Aerospace Mechanical Engineering
Korea Aerospace University
Goyang-Si, South Korea
jsw5413@naver.com

Youngju Park
Department of Aerospace Mechanical Engineering
Korea Aerospace University
Goyang-Si, South Korea
qkrdudwngg@naver.com

Siwon Lee
Department of Electrical and Electronic Engineering
Korea Aerospace University
Goyang-Si, South Korea
leesiwon0305@kau.kr

Ghilmo Choi
Department of Aerospace Mechanical Engineering
Korea Aerospace University
Goyang-Si, South Korea
ggghilmo@naver.com

Dae-Sung Jang
School of Aerospace and Mechanical Engineering
Korea Aerospace University
Goyang-Si, South Korea
dsjang@kau.ac.kr

***Abstract*—This work presents a framework that enables robots to assist with everyday tasks by leveraging Vision-Language Models (VLMs). Our framework takes both environmental information and natural language commands from users as inputs. An on-device VLM generates and validates a behavioral plan, represented as a sequence of pseudo-functions, using an actor–critic structure. These pseudo-functions are then connected to the low-level control system of a UGV to demonstrate the feasibility of the approach. Experimental results show that the framework can reliably interpret and execute user commands, highlighting its potential as a step toward practical robot assistance in daily life.**

Keywords—vision language model, on-device, robot, planning

I. Introduction

As machine learning technology advances, robots are expected to play an increasingly important role in assisting people with everyday tasks. Achieving this vision requires robots to effectively perceive their surroundings, interpret human instructions, and execute appropriate actions in real-world environments.

Recent progress in Vision-Language Models (VLMs) has demonstrated strong capabilities in grounding natural language instructions to visual inputs [1]. However, applying VLMs directly to embodied decision-making remains challenging, particularly in bridging the gap between high-level reasoning and low-level control. Existing approaches often struggle with generating reliable action sequences that can be executed on physical robotic platforms.

In this work, we present a framework that integrates on-device VLMs with an Actor–Critic structure, similar to the actor–critic design used in [2], to generate and validate behavioral plans. User commands, combined with environmental information, are translated into sequences of pseudo-functions that serve as intermediate representations. These pseudo-functions are then mapped to the low-level control system of an unmanned ground vehicle (UGV, implemented using a small RC car [3]) to evaluate the feasibility of our approach. Experimental results demonstrate that the system can interpret and execute user commands effectively, showing promise for practical robot assistance in daily scenarios.

II. System Framework

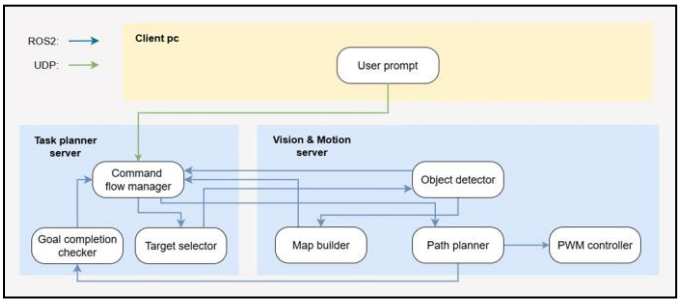

Fig. 1. Overall software architecture of the proposed framework.

A. Overview of the Framework

The proposed framework enables a mobile robot to interpret natural language instructions and execute them in real-world environments by leveraging a VLM with an actor–critic structure. Fig. 1 illustrates the overall software architecture.

B. Input Processing

The system accepts two inputs: (1) a natural language command from the user and (2) environmental data generated through Simultaneous Localization and Mapping (SLAM) [4] and object detection. The environmental data is represented as a top-view map image with detected objects as well as structured JSON file, namely *World JSON*. The *Inventory JSON* contains information about the objects the robot is holding. The *World JSON* stores spatial information about all detected objects, while the *Inventory JSON* keeps track of the

items currently held by the robot. Both types of JSON files contain ID of the each object and their class, as shown in Fig.2.

```
<World JSON>
    "objects": [
  {"id": "laundry#1",  "class": "laundry"},
  {"id": "washing_machine#1", "class": "washing_machine"}
    ]
<Inventory JSON>
    "holding": [
  { "id": "laundry#1",  "class": "laundry"},
  { "id": "sponge#1", "class": "sponge"}
    ]
```

Fig. 2. Example of World JSON and Inventory JSON file.

Using the World JSON, we construct a top-view map image as shown in Fig.3 to make the VLM better understand the surroundings. Using the odometry data derived from SLAM, it also shows the current position and orientation of the robot.

Fig. 3. Example of the top-view map image.

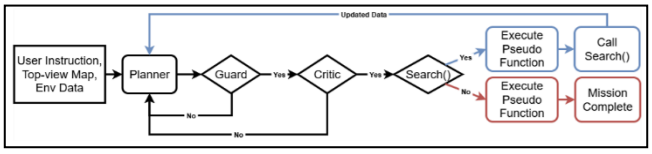

Fig. 4. Overall software architecture of the proposed framework.

C. Planner

Fig.4 illustrates the flow of Actor-Guard-Critic structure. The Planner, implemented with an on-device VLM, generates a behavioral plan in the form of a sequence of pseudo-functions. These pseudo-functions abstract high-level actions such as move_to(object), pick(object), place(object), search(object), and return_to_base() (Table I). By using pseudo-functions as intermediate representations, the framework bridges natural language understanding with low-level robot control.

TABLE I. AVAILABLE PSEUDO-FUNCTIONS

Function(argument)	Description
move_to(object)	Navigates the robot to the specified object's location on the map.
pick(object)	Grasps and lifts the target object; executable only when the robot is near the object.
place(object)	Places a held object at the current location or designated position.
search(object)	Actively explores the environment to locate the specified object and update the map.
return_to_base()	Returns the robot to its starting position (base station).

D. Guard

To ensure plan reliability, a rule-based Guard module is introduced before the Critic stage. The Guard checks whether the pseudo-functions generated meet minimal validity constraints (e.g., verifying if an object to be placed exists in the inventory). Invalid plans are rejected, and the Planner is prompted to re-generate an alternative plan with corrected reasoning.

E. Critic

The Critic evaluates the proposed plan with respect to the current environment and execution history. Unlike the Guard, which performs rule-based filtering, the Critic performs contextual validation to identify logical inconsistencies or hallucinated actions. If the plan is deemed invalid, feedback is sent to the Planner for replanning. This actor–critic cycle improves the robustness of action generation in uncertain or partially observable environments.

Fig. 5. Overall software architecture of the proposed framework.

F. Execution on UGV

Finally, we implemented the validated pseudo-functions on a small UGV by mapping them to its low-level control system. In the ROS 2 environment, these functions were converted into motor control commands using Nav2 package [5], allowing the robot to autonomously execute navigation tasks in the test field.

III. EXPERIMENTS

A. Experimental Setup

To evaluate the feasibility of the proposed framework, we conducted experiments on the UGV platform described above. The robot was equipped with an onboard computer system consisting of a Jetson Orin NX 16GB [6] for running the VLM and a Jetson Orin Nano Super 8GB [7] for auxiliary processes. The platform operated under ROS2 and was integrated with object detection modules and a SLAM package to construct a two-dimensional map of the environment. Environmental information was maintained in structured data formats (World JSON and Inventory JSON), which were continuously updated during navigation.

B. Test Scenarios

To validate the framework, we designed scenarios that mimic everyday assistance tasks in household environments. Each scenario was given as a natural-language instruction; the Planner converted the instruction into pseudo-functions, which were then validated by the Guard and Critic before execution.

A representative case is the instruction, 'Go to the trash bin.' Here, the information about the object 'trash bin#1' was provided. The initial plan generated by the Planner was:

1) move_to("trash bin#1")
2) place("trash bin#1")
3) return_to_base()

The Guard detected that the place() action was invalid since no object was being carried in the inventory. It rejected the plan and prompted replanning. The revised plan was:

1) move_to("trash bin#1")
2) return_to_base()

We also considered multi-step instructions requiring dynamic replanning. Given "Bring the dog and the apple to the person." the system proceeded iteratively as follows:

1) search("dog") → locate the dog

2) move_to("dog#1"), pick("dog#1"), move_to("person-#1"), place("dog#1")

3) search("apple") → locate the apple

4) move_to("apple#1"), pick("apple#1"), move_to("person#1"), place("apple#1"), return_to_base()

C. Complex Task Execution

During mission execution, the command-flow manager maintained the World JSON and Inventory JSON while updating a top-view map built from SLAM and object detection. The Planner produced pseudo-function sequences, which were screened by the Guard before the Critic performed contextual checks using the current environment and execution history. When the Planner proposed logically inconsistent actions (e.g., place() without a held object), the Guard generated an error message and triggered replanning, preventing such plans from reaching the Critic or the robot controller.

For partially observable scenes, the Planner explicitly invoked search() to acquire additional observations, augment the map/JSON, and restructure the plan. Treating search() as a pivot step reduced the burden of generating long one-shot plans and enabled reliable multi-iteration completion of complex instructions (e.g., sequentially delivering the dog and the apple to a person). The overall Actor–Guard–Critic loop followed the command-flow design shown in the project poster and used the predefined function set (move_to, pick, place, search, return_to_base) as the execution vocabulary.

D. Results

Guard-mediated validation. Across simple "go-to" commands and multi-step delivery tasks, the Guard consistently filtered invalid plans (e.g., rejecting place() when the inventory was empty) and prompted the Planner to regenerate corrected plans, which then executed successfully on the UGV. This reduced unnecessary escalations to the Critic and improved the overall efficiency of the loop.

Adaptivity under partial observability. For longer-horizon tasks, the Planner issued search() to discover required objects, after which the map/JSONs were updated and subsequent steps executed on the UGV.

IV. Discussion

A. Design choices and implications

Using pseudo-functions as an intermediate representation bridges natural-language intent and executable robot actions while enabling lightweight validity checks prior to execution. This abstraction works well with the command-flow manager that maintains a merged 2D map alongside World and Inventory JSONs, allowing plans to be reasoned about with explicit object-level semantics.

B. Role of the Guard versus the Critic

Placing a rule-based Guard before the Critic prevents trivial logical errors (e.g., placing an item not in the inventory) from reaching contextual evaluation. In our trials, this reduced unnecessary Critic invocations and guided the Planner to regenerate feasible alternatives with low computational overhead—an important property for on-device settings.

C. Information shaping via search()

Beyond enabling exploration under partial observability, specifying target object types in search() lets the system filter the surrounding objects recorded in the World and Inventory JSONs and the merged map, passing only task-relevant information to the VLM. This helps maintain focus and reduces context dilution during planning, especially in multi-object missions.

D. Current limitations and future directions

Operating with an on-device VLM constrains model size and occasionally yields hallucinated or instruction-deviating proposals; long-horizon missions may require multiple re-planning iterations. A promising direction is to leverage advanced training strategies, such as fine-tuning and reinforcement learning, to better align smaller models with task requirements and reduce failure modes. These approaches may help overcome the inherent limitations of compact architectures while maintaining deployability on resource-constrained platforms. From a systems perspective, our end-to-end mapping from high-level specifications to low-level control on a physical UGV confirms feasibility and provides a foundation for scaling to more cluttered environments and longer missions.

V. Conclusion

We presented an on-device VLM-based framework that converts natural-language commands into sequences of validated pseudo-functions and executes them on a physical UGV. The Actor–Guard–Critic design improved reliability by filtering invalid plans early and enabling contextual checks before execution, while the search() pivot supported staged planning and adaptive map/JSON updates under partial observability. Experiments demonstrated successful execution of both simple "go-to" commands and multi-step delivery tasks on hardware, establishing the practicality of the approach from language to motor commands. Future work will explore advanced training strategies, such as fine-tuning and reinforcement learning, to better align compact models with task requirements and mitigate limitations while preserving deplorability on resource-constrained platforms.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016.

[2] H. Song, K. Yin, Y. Ma, D. Khashabi, and D. Roth, "Prompt with actor-critic editing for instruction tuning," arXiv preprint arXiv:2308.10088, 2023.

[3] TRAXXAS, "TRX-4 Land Rover Defender," [Online], https://traxxas.com/82056-4-trx-4-land-rover-defender [Accessed: Jul. 18, 2025].

[4] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in Proc. AAAI Conf. on Artificial Intelligence, 2002.

[5] Open Navigation Group, "Navigation2 Documentation," [Online]. Available: https://docs.nav2.org/. [Accessed: Jul. 18, 2025].

[6] NVIDIA, *Jetson Orin NX Series Data Sheet*, v0.5, DS-10712-001, [Online]. Available: https://developer.download.nvidia.com/assets/embedded/secure/jetson/orin_nx/docs/Jetson_Orin_NX_DS-10712-001_v0.5.pdf. [Accessed: Jul. 18, 2025].

[7] NVIDIA, "Jetson Orin Nano Developer Kit," [Online]. Available: https://www.nvidia.com/en-us/ autonomous-machines/embedded-systems/jetson-o rin/nano-super-developer-kit/. [Accessed: Jul. 18, 2025].