BORISAgent: An Architecture for LLM-Based Task Planning in Domestic Service Robots

Gabriel A. Dorneles[0009−0001−6554−5832], Richard J. C. Assis[0009−0001−0911−7382], Cris L. Froes[0009−0007−4475−4874], Enrico F. Torres[0009−0003−7530−0470], Vitor W. Anello[0009−0007−0710−0820], Guilherme M. Burkert[0009−0001−9152−0807], Murilo C. Bicho[0009−0007−3867−5387], Alisson H. Kolling[0000−0002−3416−069X], Rodrigo S. Guerra[0000−0003−4011−0901], and Paulo L.J. Drews-Jr[0000−0002−7519−0502]

NAUTEC, Center for Computational Sciences - C3,
Universidade Federal do Rio Grande - FURG, Brazil
gdorneles@furg.br
http://www.c3.furg.br

Abstract. We present a modular architecture for general purpose domestic robots that confines non-deterministic reasoning to a large language model (LLM) acting as a code-generating planner, while delegating execution, monitoring, and recovery to a deterministic skill layer built on ROS 2 state machines. This approach requires no fine-tuning, relying entirely on the reasoning capabilities of foundation models guided by structured prompting. We evaluate the architecture on a 100-command benchmark derived from the General Purpose Service Robot (GPSR) task, comparing eight foundation models ranging from 3B to 70B parameters as well as cloud-hosted APIs. The best model achieves a 95.0% task planning success rate, while an ablation study shows that code generation consistently outperforms structured tool calling across all model scales. We further validate the system through 72 end-to-end trials on the BORIS physical platform, comparing cloud and on-device inference and achieving 83.3% task completion with a structured failure analysis distinguishing planning from execution errors. More information can be found on the project's website: https://fbot-research.github.io/BORISAgent.

Keywords: Task Planning · Large Language Models · Domestic Service Robots · Code Generation Agents · RoboCup@Home.

1 Introduction

General-purpose domestic robots must interpret natural-language instructions and translate them into sequences of actions in home environments. This problem combines challenges from natural language understanding, multi-modal perception, spatial memory, and low-level control. For example, a command such as "bring me something to read" may involve resolving semantic ambiguity about the intended object, searching for suitable candidates across multiple rooms,

generating safe motions while monitoring execution, and recovering from failures under partial observability.

In the last few years, several end-to-end systems that allow robots to execute long-horizon, real-world tasks guided by natural language instructions have emerged [1–5]. However, despite the successes achieved in the linguistic and visual-linguistic domains, a generalist robot policy that can reliably operate in unstructured real-world environments remains an open challenge. A promising direction to improve reliability, analogous to tool-augmented and retrieval-augmented language models in natural language processing (NLP), is to allow a high-level agent to invoke reusable low-level skills rather than relying on end-to-end policies [6].

This paper introduces a modular, multi-modal architecture for domestic instruction following that grounds large language models (LLMs) in a structured skill layer. Given a single natural-language command, the system produces a sequence of grounded actions through three stages: speech and text understanding, LLM-based planning that decomposes the command into high-level subgoals, and deterministic execution, in which each tool wraps a ROS 2 state-machine library. By isolating open-ended reasoning within the planner and delegating execution, monitoring, and recovery to a deterministic layer, the architecture can incorporate new capabilities without compromising safety or reliability in real domestic settings.

This paper makes the following contributions:

- An architecture for autonomous decision-making that combines LLM-based code-generating planners with deterministic ROS 2 state-machine skills for reliable execution, monitoring, and recovery in domestic service robots.
- A comparative evaluation of eight foundation models (3B–70B parameters and cloud APIs) on a 100-command GPSR benchmark, together with an ablation showing that code generation consistently outperforms structured tool calling across model scales.
- An end-to-end real-world evaluation on a physical robot comparing cloud and on-device inference across 72 trials with failure analysis distinguishing planning from execution errors, further validated by deployments at RoboCup@Home 2025 and the Brazilian Robotics Competition (CBR) 2025.

2 Related Work

The introduction of LLMs has accelerated progress in Physical AI, the area of research that unites sensing, planning, and actuation in real-world agents. Because foundation models encode broad world knowledge and reasoning, researchers have begun using them as high-level planners, task and instruction translators, and natural-language interfaces for human–robot interaction [6].

Early work demonstrated that pre-trained LLMs could decompose natural language instructions into robot-executable subgoals without any fine-tuning. Huang et al. [7] showed that frozen LLMs could extract actionable plans through prompting alone, while SayCan [8] grounded these plans by scoring candidate

skills with learned value functions, constraining LLM outputs to a robot's actual capabilities, an idea later formalized through programmatic prompt templates in ProgPrompt [9]. Our work follows this prompting approach, combining it with a structured tool-calling interface inspired by ReAct [10], which mixes reasoning with discrete skill calling.

End-to-end approaches such as PaLM-E [11], RT-1 [1], π_0 [2], and $\pi_{0.5}$ [5] achieve strong performance by training directly on large-scale robot demonstrations, but at significant data and compute cost. GR00T N1 [3] extends this direction to generalist humanoid control. Our modular approach deliberately avoids end-to-end training, preserving skill composability and enabling drop-in replacement of the LLM backend.

The most architecturally similar prior work to ours, ROS-LLM [12] integrates LLMs with ROS behavior trees and state machines through an atomic action library with execution feedback, though evaluated on much simpler task sets using only a robotic manipulator. Closest to our deployment setting, RDMM [13] fine-tunes quantized LLMs on a proprietary dataset of 27k planning instances curated from RoboCup@Home scenarios, achieving 92.98% planning accuracy on edge hardware while allowing a robot to reason about its own capabilities and identity. However, this approach requires task-specific training data and a dedicated fine-tuning pipeline, limiting generalizability without dataset collection and retraining; furthermore, the evaluation is limited to planning accuracy, with no structured end-to-end evaluation on a physical robot. In contrast, our method requires no prior training or domain-specific data, building directly on the reasoning capabilities of foundation models.

3 System Overview

This section describes the proposed architecture. We first introduce the robot platform used for development and evaluation, then detail the design: a code-generating LLM planner at the top and a deterministic skill layer built on ROS 2 state machines.

3.1 Development Platform

The Brazilian Open Robot for Indoor Service (BORIS) is a 1.48m tall domestic service robot built on top of a SHARK mobile base [14]. Onboard compute is split between a Mini PC for higher-level processes and an NVIDIA Jetson Orin AGX for real-time perception and on-device inference. The platform carries a 5-DOF Interbotix WidowX-200 arm for manipulation, and an animatronic head with a 2-DOF neck and 12 actuated features (jaw, eyelids, eyebrows, eyes) that supports seven pre-programmed expressions (happy, sad, neutral, angry, surprised, suspicious, sleepy), providing a social interaction component. A more detailed description of the platform is provided in the 2026 Team Description Paper for the RoboCup@Home competition [1].

[1] Found in our GitHub organization: https://github.com/fbotathome

3.2 Agent Architecture

The proposed architecture follows a two-layer design that separates non-deterministic reasoning from deterministic execution. At the top layer, a large language model acts as a code-generating planner that decomposes natural-language instructions into sequences of tool invocations. At the bottom layer, each tool encapsulates a finite state machine (FSM) built on YASMIN [15] and grounded in ROS 2 action and service interfaces. Figure 1 shows this architecture in detail.

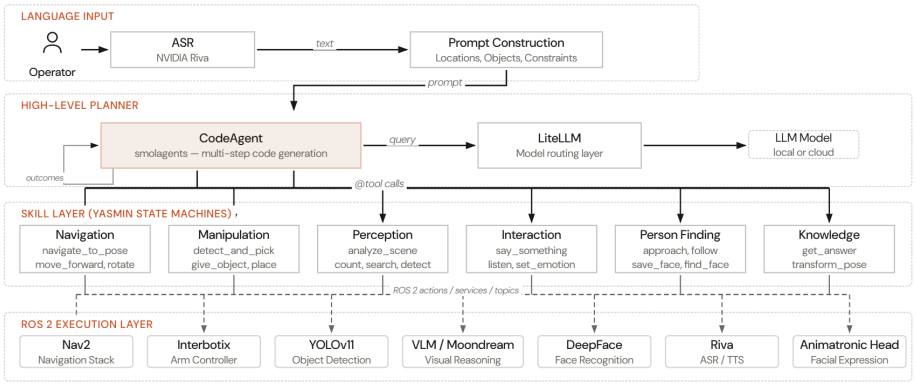


Fig. 1: System architecture. A natural-language command is transcribed by the ASR module and augmented with environment context in the prompt construction stage. The CodeAgent decomposes the task into a sequence of `@tool` calls, each of which executes a YASMIN state machine that interfaces with the robot's actuators and sensors through ROS 2.

High-Level Planner The planner is implemented as a `CodeAgent` from the *smolagents* framework [16]. Unlike approaches that restrict the LLM to selecting from a fixed action list, a `CodeAgent` generates executable Python code at every reasoning step. This formulation includes both chain-of-thought reasoning [17] and tool-augmented generation [10], since the model can combine arbitrary computation (e.g., string manipulation, conditional branching) with grounded skill calls within the same code block. The agent operates in a multi-step loop: at each step it observes the return values of previously executed tools, generates a new code snippet, and continues until it issues a `final_answer` call.

The planner is accessed through a ROS 2 service that receives a natural-language command string and returns the final execution result. Upon receiving a request, the system constructs a structured prompt that provides the LLM with a role description, establishing the robot's identity and behavioral constraints, the set of named locations available in the current map, a categorized inventory of detectable object classes, and the task instruction itself.

LLM inference is handled through LiteLLM [18], an abstraction layer that exposes a unified API across different model providers. This design allows for

switching between on-device models served via Ollama and cloud-hosted endpoints without modifying application code, which is instrumental for the comparative evaluation presented in Section 4.

Table 1: Available Robot Skills

Skill	**Args**	**Description**
Navigation		
`navigate_to_pose`	name	Go to named location
`move_forward`	dist	Move forward by meters
`rotate`	angle	Rotate in place (rad)
`follow_person`	desc	Follow person by appearance
`approach_object`•	obj	Detect and approach object
`approach_person`⋆	desc	Approach by description
`find_person_by_name`⋆	name	Find unknown person by asking
`find_saved_person`	uuid	Find previously known person
Perception		
`analyze_scene`⋆	question	Answer question about scene
`count_objects`•	class	Count objects by label/class
`count_people`⋆	desc	Count people by description
`search_person`⋆	name	Locate person, return pose
`save_person_face`	name	Register face for later recall
Manipulation		
`detect_and_pick`	obj	Detect object and grasp it
`give_object`	—	Hand object to user
`place_object`	loc	Place held object at location
Interaction		
`say_something`	text	Speak text via TTS
`listen_something`	[words]	ASR with optional boost list
`get_answer`⋆	question	Answer knowledge question
`set_emotion`	emotion	Set robot facial expression

⋆Invokes a vision-language model (VLM) for open-vocabulary reasoning.
•Uses an object detector without additional LLM calls.

Skill Layer Each tool registered with the planner is a Python function decorated with the `@tool` interface from *smolagents*, which automatically exposes the function's signature and docstring to the LLM as part of the system prompt. Internally, every tool constructs and executes a YASMIN state machine.

A skill may be atomic, wrapping a single ROS 2 action or service call, or composite, chaining multiple states into a longer behavioral sequence. For instance, the `set_emotion` skill simply receives a string, validates it among the possible options, and calls a ROS 2 service with the emotion name. The `detect_and_pick_object` skill, on the other hand, composes object search with multi-angle neck sweeping, approach navigation, optimal grasp planning, and gripper actuation through the Interbotix arm controller. Each state machine re-

turns a discrete outcome (SUCCEED, ABORT, CANCEL, or TIMEOUT) that the tool translates into a boolean value returned to the planner, enabling the LLM to branch on success or failure in subsequent code steps. Table 1 summarizes the 20 skills currently available to the planner, organized by functional domain.

4 Experiments

This section reports experiments across three axes: planning accuracy for code generation across foundation models on a GPSR benchmark (Section 4.1), the effect of code generation versus structured tool calling (Section 4.2), and real-world validation on the BORIS platform (Section 4.3).

4.1 Planning Evaluation

We evaluate the proposed architecture on a benchmark of 100 natural-language commands derived from the RoboCup@Home GPSR task[2]. Commands are organized into four tiers of 25 commands each, designed to isolate progressively harder planning challenges. Tier 1 contains single-skill commands with no ambiguity (e.g., *"Navigate to the kitchen"*), testing basic tool selection and argument mapping. Tier 2 comprises multi-step sequential commands that require chaining three to five skills in the correct order (e.g., *"Give me a lemon from the refrigerator"*). Tier 3 introduces semantic ambiguity and commonsense reasoning, where the instruction is underspecified and the planner must infer a viable strategy (e.g., *"Bring me something to drink from the kitchen"*). Tier 4 requires conditional logic, fallback strategies, or recovery from anticipated failures (e.g., *"Take a drink from the kitchen to the person in the bedroom, but only if they are not sleeping"*). Tiers 1 and 2 were curated from the official GPSR command generator, Tier 1 being a simplified version of those commands, with structural duplicates removed to maximize diversity across skill categories; Tiers 3 and 4 were manually authored using the same environment vocabulary.

Since the planner operates as a closed-loop agent that generates code, observes tool feedback, and iterates with a new model request, the evaluation targets the final task plan. Two independent annotators review the complete execution trace of each task and assign an outcome score on a four-point scale: 3 (task fully accomplished), 2 (minor non-critical error), 1 (partially accomplished with a critical step wrong or missing), and 0 (task failed or agent loop exhausted without completion). Disagreements were resolved through discussion between the two annotators until consensus was reached. In addition to the outcome score, we extract two complementary metrics directly from the execution logs. *Steps* reports the average number of CodeAgent loop iterations per task, while *Tokens* reports the mean total token consumption, serving as a proxy for inference cost. All tasks are subject to a maximum of 20 steps, at which point execution is terminated regardless of completion status.

[2] The benchmark and system prompt will be made publicly available on the project's website upon publication.

Table 2: Evaluation of foundation models as iterative planners on the GPSR benchmark (100 commands total). Each task is judged on final outcome after the full CodeAgent loop using a four-point scale: 3 (correct), 2 (minor error), 1 (partial), 0 (incorrect). **Task Score** is the percentage of tasks scoring ≥ 2, i.e. tasks where the agent accomplished the goal with at most minor errors; **Steps** ($\bar{n}$) is the mean number of CodeAgent reasoning steps per task. **Tokens** ($\bar{k}$ [10^3]) is the mean total tokens consumed per task. All models received the same system prompt, grounding context, and tools' docstring; inference used greedy decoding.

Model	**Task Score (%)**					**Steps**	**In. T.**	**Out. T.**
	Tier 1	Tier 2	Tier 3	Tier 4	Overall	($\bar{n}$)	($\bar{k}$ [10^3])	($\bar{k}$ [10^3])
Cloud-hosted models								
Claude Sonnet 4.6	100.0	92.0	88.0	100.0	**95.0**	5.8 ± 2.8	30.7 ± 16.8	0.5 ± 0.3
Gemini 3 Flash	100.0	76.0	88.0	92.0	89.0	2.9 ± 1.1	13.3 ± 5.6	1.1 ± 0.9
Open-weight models								
Llama 3.3 70B	88.0	68.0	56.0	48.0	65.0	2.1 ± 1.3	8.8 ± 6.0	0.3 ± 0.2
Qwen3 32B	96.0	68.0	72.0	68.0	**76.0**	2.0 ± 1.3	8.8 ± 5.5	1.8 ± 1.4
Gemma 3 27B	80.0	28.0	16.0	24.0	37.0	9.9 ± 9.0	77.5 ± 81.4	1.3 ± 1.2
Qwen3 14B	84.0	56.0	60.0	56.0	64.0	2.4 ± 3.0	10.3 ± 12.6	2.2 ± 1.7
Llama 3.1 8B	64.0	20.0	16.0	4.0	26.0	10.8 ± 8.4	44.0 ± 34.4	1.8 ± 2.1
Llama 3.2 3B	44.0	8.0	4.0	0.0	14.0	11.5 ± 8.5	47.2 ± 34.8	3.6 ± 7.8

For cloud-hosted inference, we include as representative frontier models: Claude Sonnet 4.6, which is widely recognized for strong code generation and agentic reasoning, and Gemini 3 Flash, a recent high-efficiency release with competitive benchmark performance. For open-weight models, we select models with increasing parameter counts across two families, Llama and Qwen3, to assess how planning capability scales with model size. Gemma 3 27B was included as the only local multimodal model, offering a contrast between vision-capable and text-only architectures at a similar parameter count.

Table 2 summarizes the results across all eight models. Claude Sonnet 4.6 achieves the highest overall task score at 95.0%, followed by Gemini 3 Flash at 89.0%, while the best open-weight model, Qwen3-32B, reaches 76.0%, a 19 point difference relative to the top performer. A closer look at per-tier scores reveals that the primary source of difficulty is not semantic ambiguity or conditional logic, but rather the multi-step execution required from Tier 2 onward. For the stronger models, Tiers 2 through 4 show comparable performance: Gemini 3 Flash scores 76.0%, 88.0%, and 92.0% on Tiers 2–4 respectively, while Qwen3-32B scores 68.0%, 72.0%, and 68.0%. In both cases, the sharpest accuracy drop occurs between Tier 1 and Tier 2, after which scores plateau or even recover. This pattern suggests that models with sufficient reasoning capacity can handle underspecified instructions (Tier 3) and conditional branching (Tier 4) about as

well as straightforward multi-step sequences (Tier 2). The main challenge lies in correctly chaining and monitoring multiple skill invocations rather than in the nature of the reasoning required. The smaller models, by contrast, lack the baseline capability to compose multi-step plans at all, which makes any further distinction across tiers difficult: Llama 3.2 3B and Llama 3.1 8B struggle on every tier above Tier 1.

The step and token columns reveal two distinct patterns. Low-performing models (Gemma 3 27B at 9.9 steps, Llama 3.1 8B at 10.8, Llama 3.2 3B at 11.5) frequently exhaust the 20-step budget without completing the task, inflating input token counts as each step appends the full execution trace to the context. Gemma 3 27B is the most extreme case, consuming an average of 77.5k input tokens per task while achieving less than half the score of Qwen3-32B.

Among stronger models, step count reflects planning strategy rather than failure. Llama 3.3 70B and Qwen3-32B average roughly 2 steps, generating complete plans in a single code block; this one-shot approach is efficient when correct but fragile, since errors in early steps are uncorrected. Claude Sonnet 4.6, by contrast, averages 5.8 steps and decomposes tasks into one or two tool calls per step, conditioning subsequent actions on observed outcomes and achieving the highest score at the cost of higher token consumption. Qualitatively, Claude Sonnet 4.6 shows consistent awareness of the full skill set and well-structured VLM queries; its few errors were tool orchestration mistakes (e.g., invoking `follow_person` without first obtaining a visual description) rather than reasoning failures.

4.2 Code Generation vs. Structured Tool Calling

The results in Table 2 were obtained using a CodeAgent that generates executable Python at each reasoning step. An alternative formulation, adopted in prior work [9, 12], restricts the LLM to producing structured JSON action sequences rather than free-form code. To isolate the effect of this design choice, we repeat the benchmark evaluation using the ToolCallingAgent from the same *smolagents* framework. Both agents share the identical iterative loop, system prompt, tool docstrings, and grounding context; the only difference is the output format the LLM is instructed to produce.

We select three models that span the performance range observed in Experiment 1: a cloud model (Gemini 3 Flash), the strongest open-weight model (Qwen3-32B), and a low-tier model (Llama 3.1 8B). Running the ablation on models of different capability levels allows us to determine whether the advantage of code generation, if any, is consistent across model scales or emerges only above or below a certain capability threshold.

Table 3 presents the results. Code generation consistently outperforms structured tool calling across all three models. Gemini 3 Flash drops from 89.0% to 83.0%, Qwen3-32B from 76.0% to 65.0%, and Llama 3.1 8B from 26.0% to a complete failure at 0.0%. The gap widens as model capability decreases, suggesting that code generation extends the range of models viable as robotic planners. Moreover, the ToolCallingAgent requires roughly twice as many steps

Table 3: Code generation vs. structured tool calling. **T1-T4** (%) is the percentage of tasks scoring ≥ 2. **Steps** ($\bar{n}$) is the mean of reasoning steps per task. Bold marks the higher score.

Model	Plan	T1	T2	T3	T4	All	Steps
Gemini 3 Flash	Code	100.0	76.0	88.0	92.0	**89.0**	2.9 ± 1.1
	Tool	96.0	72.0	76.0	88.0	83.0	5.7 ± 2.4
Qwen3-32B	Code	96.0	68.0	72.0	68.0	**76.0**	2.0 ± 1.3
	Tool	88.0	40.0	60.0	72.0	65.0	7.5 ± 3.8
Llama 3.1 8B	Code	64.0	20.0	16.0	4.0	**26.0**	10.8 ± 8.4
	Tool	0.0	0.0	0.0	0.0	0.0	9.8 ± 7.9

as the CodeAgent, since it tends to issue fewer tool calls per iteration, stepping through actions sequentially rather than composing them in a single code block.

These results reflect an expressiveness gap between the two formulations. A CodeAgent can import libraries, perform arithmetic and string manipulation, define variables to track intermediate results across tool calls, and use conditional branching and loops natively within the generated code. A ToolCallingAgent, by contrast, is restricted to selecting a tool name and populating its arguments, any logic that falls outside a single tool invocation must be encoded in the model's reasoning rather than explicitly expressed in executable form.

These findings are consistent with Wang et al. [19], who show across 17 LLMs that executable code actions outperform JSON-based formats by up to 20%, with the gap widening as complexity increases. Our benchmark evaluates this effect in a more realistic robotic setting, with 20 tools, semantic disambiguation, and VLM-backed perception skills whose outputs must be interpreted within generated code.

4.3 Real World Deployment

Sections 4.1 and 4.2 evaluate planning in isolation. In this experiment, we deploy the full architecture on the BORIS platform and execute commands end-to-end, from speech input through physical action completion. We construct a reduced benchmark of 12 unique commands drawn from the same tier structure used in Section 4.1, with three commands per tier. Each command is executed three times per model. All trials are conducted in an arena simulating a domestic environment, and example execution videos are available on our website.

We compare Gemini 3 Flash against Qwen3-14B served on-device on the Jetson Orin AGX with 4-bit quantization. The 14B variant is chosen over the higher-scoring 32B since it fits within the Orin's 64GB unified memory while leaving headroom for concurrent perception and has acceptable latency. The 12 selected commands do not require object manipulation, isolating the other capabilities from the limitations of the current 5-DOF arm. We additionally

record total wall-clock time, and classify each trial scoring below 2 by its primary failure source: planning, navigation, perception, or interaction.

Table 4: Real-world evaluation on the BORIS platform. **Task Score** (%) is the percentage of trials scoring ≥ 2 across 36 trials per model. **Time** (s) is the per-trial mean. **Recov.** reports the number of trials in which at least one tool returned failure but the agent recovered to a final score ≥ 2. **P**/**N**/**V**/**I** report unrecovered failure counts attributed to planning, navigation, perception, or interaction.

Model	Task Score (%)	Time (s)	Recov.	P	N	V	I
Gemini 3 Flash	83.3	106.8 ± 81.5	4	1	1	4	0
Qwen3-14B	58.3	267.2 ± 179.4	5	11	0	3	1

Table 4 summarizes the results. Gemini 3 Flash achieves 83.3% task completion across 36 trials, while Qwen3-14B reaches 58.3%, a 25-point gap consistent with the planning-only evaluation in Section 4.1. The failure breakdown reveals distinct bottlenecks for each model: Qwen3-14B's failures are predominantly planning errors, while Gemini 3 Flash produces only 1 planning failure; its primary failure mode is perception, where VLM backed skills returned incorrect or incomplete scene interpretations that the planner could not recover from.

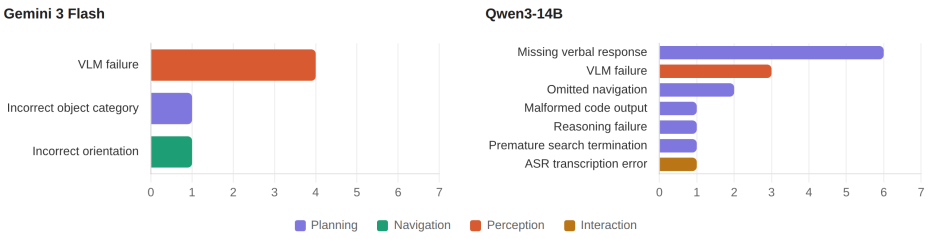


Fig. 2: Distribution of unrecovered failure causes across 36 real-world trials per model. Failures are classified by primary source (planning, navigation, perception, or interaction) and labeled by specific sub-reason.

Figure 2 provides a better view of these failures. Among Qwen3-14B's planning errors, missing verbal response, where the model completed reasoning but omitted the final `say_something` call, accounts for 6 of 15 total failures, pointing to a prompt-following deficit rather than diverse planning breakdowns. Gemini 3 Flash shows no such pattern: its failures come from VLM-backed skills returning incorrect scene interpretations, a failure mode outside the planner's control. Both models achieved successful recoveries, confirming that the architecture enables error recovery regardless of model capability. Wall-clock time differs substantially, reflecting both the higher latency of on-device inference and the additional steps triggered by more frequent errors.

The architecture was also validated at a competition on the GPSR task: at RoboCup@Home 2025, a prior planning system without the CodeAgent scored 65/1000, while at CBR 2025 the proposed architecture with Gemini 2.5 Flash scored 880/1000 on the same task format, though per-command logs were not recorded at either event.

5 Conclusion

We presented a modular architecture for domestic service robots that confines non-deterministic LLM reasoning to a code-generating planner while delegating execution to deterministic ROS 2 state machines. Evaluation on a 100-command GPSR benchmark showed that Claude Sonnet 4.6 achieves 95.0% task planning success rate, code generation consistently outperforms structured tool calling across model scales, and the architecture transfers to real-world deployment with 83.3% success on the BORIS platform using Gemini 3 Flash.

Three limitations suggest directions for future work. First, the dependence on manually programmed skills constrains manipulation capabilities. Integrating vision-language-action policies such as π_0 [2], $\pi_{0.5}$ [5], or GR00T N1 [3] as skills within the CodeAgent framework should enable more reliable object manipulation. Second, the lack of long-term spatial memory limits the robot to pre-mapped environments; incorporating a spatial retrieval-augmented generation framework would allow the planner to resolve open-vocabulary navigation commands in novel settings by querying a persistent map-frame memory. Third, our evaluation uses a single fixed prompt across all models; a systematic ablation of prompt structure, including role descriptions, context formatting, and tool docstring verbosity, would clarify how sensitive planning accuracy is to prompt design and guide practitioners in adapting the architecture to new domains.

Acknowledgments. This work was partly supported by FAURG and CNPQ.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. A. Brohan, N. Brown, J. Carbajal *et al.*, "Rt-1: Robotics transformer for real-world control at scale," in *Proceedings of Robotics: Science and Systems (RSS)*, Daegu, Republic of Korea, Jul. 2023.
2. K. Black, N. Brown, D. Driess, A. Esmail *et al.*, "π_0: A vision-language-action flow model for general robot control," 2024. [Online]. Available: https://arxiv.org/abs/2410.24164
3. NVIDIA, J. Bjorck, F. Castañeda, N. Cherniadev *et al.*, "GR00T N1: An open foundation model for generalist humanoid robots," in *ArXiv Preprint*, March 2025.
4. G. A. Dorneles, K. S. Kappel, R. S. Guerra, and P. L. J. Drews-Jr, "Foundations of vision language action models for generalist robotic manipulation," in *Proceedings of the 2nd Brazilian Conference on Robotics (CROS)*, 2026, to appear.

5. K. Black, N. Brown, Darpinian *et al.*, "$\pi_{0.5}$: a vision-language-action model with open-world generalization," in *Proceedings of The 9th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Lim, S. Song, and H.-W. Park, Eds., vol. 305. PMLR, 27–30 Sep 2025, pp. 17–40. [Online]. Available: https://proceedings.mlr.press/v305/black25a.html
6. C. Huyen, *AI Engineering*. USA: O'Reilly Media, 2025.
7. W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *ArXiv*, vol. abs/2201.07207, 2022.
8. M. Ahn, A. Brohan, N. Brown, Y. Chebotar *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*, 2022.
9. I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11 523–11 530, 2022.
10. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *ArXiv*, vol. abs/2210.03629, 2022.
11. D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery *et al.*, "Palm-e: An embodied multimodal language model," 2023. [Online]. Available: https://arxiv.org/abs/2303.03378
12. C. E. Mower, Y. Wan, H.-D. Yu, A. Grosnit, J. Gonzalez-Billandon *et al.*, "Ros-llm: A ros framework for embodied ai with task feedback and structured reasoning," *ArXiv*, vol. abs/2406.19741, 2024.
13. S. Nasrat, M. Jo, S. Lee, M. Kim, J. Lee, Y. Jang, and S.-J. Yi, "Rdmm: Enhancing household robotics with on-device contextual memory and decision making," in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025, pp. 19 944–19 949.
14. N. F. Dias, J. D. S. Dyonisio, L. F. M. Quadros, G. A. Dorneles, J. J. P. Prado, M. Z. Rocha, M. C. Bicho, J. F. S. S. Lemos, H. C. Bezerra, C. N. S. Teixeira, I. P. Maurell, R. S. Guerra, and P. L. J. Drews-Jr., "SHARK: Stable hoverboard-driven autonomous robot kit," in *Proceedings of the V Brazilian Humanoid Robot Workshop (BRAHUR) and VI Brazilian Workshop on Service Robotics (BRASERO)*, 2024.
15. M. González-Santamarta, F. J. Rodríguez-Lera, C. F. Llamas, F. M. Rico, and V. M. Olivera, "Yasmin: Yet another state machine library for ros 2," 2022. [Online]. Available: https://arxiv.org/abs/2205.13284
16. A. Roucher, A. V. del Moral, T. Wolf, L. von Werra, and E. Kaunismäki, "'smolagents': a smol library to build great agentic systems." https://github.com/huggingface/smolagents, 2025.
17. J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903
18. BerriAI, "Litellm: Call all llm apis using the openai format," 2024. [Online]. Available: https://github.com/BerriAI/litellm
19. X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better llm agents," 2024. [Online]. Available: https://arxiv.org/abs/2402.01030