Adding Determinism to a Dialogue Agent for a Robotic Environment

Oihana Garcia, Riccardo Cocola, Cristina Aceta
TEKNIKER, Iñaki Goenaga 5, Eibar, 20600, Basque Country, Spain
oihana.garcia@tekniker.es, riccardo.cocola@tekniker.es, cristina.aceta@tekniker.es

Abstract

Large Language Models (LLMs) have strong capabilities in natural dialogue, but their inherent indeterminacy presents challenges in robotic environments where safety and reliability are critical. In this study, we propose a dialogue agent that has been developed to guide and support human operators during robot demonstrations, following the Learning from Demonstration (LfD) paradigm, where the robot learns tasks from the operator's actions. The agent presented in this work extends the standard prompt-based LLM setup by integrating state graphs that explicitly encode dialogue states and transitions. This structure ensures that user interactions follow the intended path, while still allowing users to communicate in a flexible and natural manner. The state graph agent is benchmarked against a monolithic prompt baseline in challenging dialogue scenarios involving ambiguity, incomplete actions, or operator errors. Despite the LLM prompt achieving good standalone performance, the state-controlled agent shows greater contextual understanding, reasoning capability, and advisory performance, leading to more intelligent and reliable interactions.

1 Introduction

Dialogue systems are increasingly adopted in industrial robotics to assist human operators during demonstrations and complex tasks. Large Language Models (LLMs) enable flexible natural language understanding and generation, but their inherent non-determinism can lead to inconsistent or unsafe responses—an unacceptable risk in safety-critical and highly structured environments.

In this study, we focus on the context of Learning from Demonstration (LfD), where machines are taught through examples rather than explicit programming. Within this framework, we propose a dialogue agent that guides and supports human operators during robot demonstrations. We compare two strategies for mitigating LLM non-determinism: a carefully engineered prompt-approach and an AI agent with a state machine, which integrates LLM within a deterministic execution framework to ensure consistent, safe, and reliable interactions. In addition to comparing these two architectures, the evaluation includes models of various sizes to assess the impact of model scale on performance and reliability.

This paper is structured as follows: Section 2 provides related work, Section 3 describes the system presented in this paper and Sections 4 and 5 report the experimental setup for evaluation (including the use case description) and the results obtained. Finally, Section 6 outlines the conclusions and related work of this paper.

2 Related Work

While LLMs enable flexible language understanding and generation, their unpredictability can be a liability in human–robot collaboration, where deterministic behaviour is essential to avoid real-world mistakes (Kim et al., 2024). To mitigate this, researchers are exploring hybrid approaches. In these, LLMs are used for high-level language interpretation or plan suggestion, but their outputs are funnelled through deterministic layers such as symbolic planners, logical checkers, or formal representations like linear temporal logic (Mendoza et al., 2024) and planning domain definition language (Huang et al., 2025). These systems ensure that generated actions are valid, safe, and repeatable before execution.

Another approach to mitigating non-determinism involves agent-based architectures that combine LLM reasoning with stateful execution. By explicitly tracking environment state, managing contextual information, and iteratively refining plans through closed-loop interaction, such agents reduce the influence of stochastic language model outputs on downstream behaviour. A representative example is SayCan (Ichter et al., 2023), which employs an LLM solely to assign semantic preference scores to candidate actions, while deterministic affordance functions, grounded in the robot's current state, determine which actions are physically executable. Consequently, LLM uncertainty is confined to soft preference ranking rather than direct action selection, improving execution reliability. Another notable application of AI agents in robotics include the work of Yang et al. (2024), which presents an AI agent that combines LLM-based reasoning with formal safety constraint enforcement to prune unsafe actions, assess compliance, and provide interpretable explanations.

3 System Design

This section focuses on the implementation of the dialogue agent in the context of robot demonstration support, where the operator demonstrates tasks that the robot then imitates. Both the baseline monolithic prompt agent and the agentic dialogue with states are presented, showing how determinism can be integrated into an LLM-driven agent.

3.1 System Architecture

The system is composed of several modules that together enable spoken human-robot interaction. The operator communicates through a headset, and spoken input is automatically transcribed into text using a speech-to-text module. This text is sent to the dialogue agent, which returns a JSON object containing an action command for the robot and a natural language response. The dialogue agent is deployed on a server and is accessed via an API. The command is executed by the robot, while the response is converted to speech and played back to the operator. Figure 1 illustrates the overall flow.

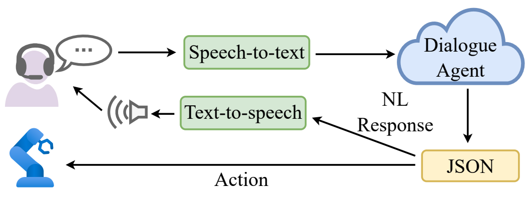

Figure 1: Dialogue API and operator interaction flow

The intention was to create a fully standalone dialogue agent that did not require an internet connection. For local inference, the Ollama (https://ollama.com/) engine was selected due to its lightweight runner, container-like model management, and its ability to run privately on the local machine. This ensures efficiency, data privacy, independence from other services and avoid external dependencies. Several LLMs were evaluated and compared in terms of performance. To enable perception of the workspace, the vision module used LLaVA to interpret visual input. Finally, LangChain (https://www.langchain.com/) was employed to integrate the system, managing chat history, memory and connectivity with Ollama endpoints.

3.2 Monolithic Prompt Agent

This agent has been developed to include the essential components required for the dialogue system: a short-term memory to track conversations and maintain continuity with the user and an Ollama LLM guided by a prompt containing all the necessary instructions for the use case.

Messages are handled using LangChain's *BaseMessage*, while memory relies on the *BaseChatMessageHistory* abstract base class, where some methods are overridden to append new messages to the history. The conversation flow is structured through a *ChatPromptTemplate*, which incorporates a *MessagesPlaceholder* for the history, a system prompt, and the user input string. To manage token limits, LangChain's *trim_messages* function is applied, keeping only the most recent exchanges. The LLM module is defined as a *RunnableWithMessageHistory* and is responsible for reading, updating and maintaining the consistency of the chat memory across sessions. When invoked, the module automatically manages and updates the memory based on the session ID to ensure continuity, as illustrated in Figure 2.

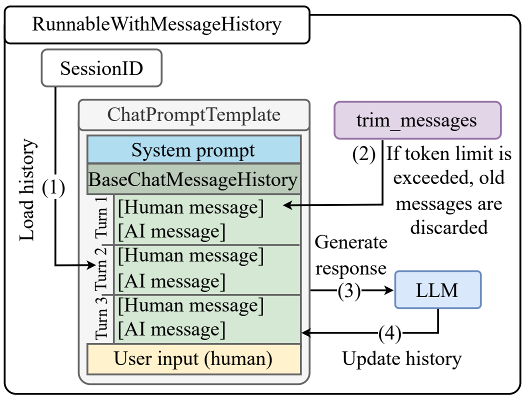

Figure 2: Workflow of the message handling and memory update process in LangChain

The prompt was carefully designed using prompt engineering techniques to constrain the model's behaviour. The prompt was carefully designed using prompt engineering techniques to constrain the model's behavior. Specifically, (1) the assistant's role and the robot's operational context were clearly defined, (2) few-shot examples were incorporated to guide responses, (3) action sequencing was enforced through explicit prerequisite constraints, (4) permitted actions, execution conditions, and system responses to unmet prerequisites were specified to enable structured error handling, and (5) stylistic guidance and turn-by-turn objectives were included to support consistent, task-oriented interactions. However, as there is no explicit state management, the system relies entirely on the LLM and its prompt restrictions to guide the dialogue. The absence of an explicit state machine to track the position of the dialogue within the demonstration process means that the system retains a degree of indeterminacy, which can result in inconsistent behaviour by the LLM.

In order to make the agent contextually aware of its workspace, a multimodal perception module was added. Its purpose is to analyse workspace images and determine the presence of objects in key areas, such as the picking region. Importantly, this perception step is performed a priori: the MLLM processes the images separately and produces a structured "yes/no" output. This result is then inserted into the user input within the prompt, allowing the dialogue agent to reason about the workspace in real time.

3.3 Deterministic State-based Agent

In order to achieve a more controlled and deterministic flow within the robotic environment, a second interpretation of the agent was developed. Building on the agent described in the previous section, explicit states were incorporated to ensure that the LLM could follow the state graph used by the robot. To this end, a class named *DemonstrationState* was created, containing attributes used to control the transitions within the demonstration state. This class provides a structured representation of the current context, enabling the flow of dialogue to be managed programmatically, thus reducing the dependence on the internal behaviour of the LLM.

When using the *DemonstrationState* class, additional steps are taken before generating the final response, compared to the monolithic agent. In this approach, the LLM is invoked twice, and the resulting process flow is as follows:

1. **Action identification:** The first LLM identifies the operator's intended action based on the dialogue history and the user's latest input.
2. **Action validation and state update:** The system then proceeds to evaluate the validity of the identified action, considering the parameters of the current state. The *DemonstrationState* object is consulted, and logical conditions determine whether the action can proceed. The state transitions and action conditions are manually specified and explicitly encoded and validated in accordance with the principles of finite-state machines (FSMs). In this configuration, the robot can be in one and only one of a finite number of clearly defined states at any given time. Each state corresponds to a specific phase of the dialogue or demonstration, and transitions between states are only triggered by validated user actions. If the action involves verifying a workspace region, the system invokes a dedicated function to check the area. This function leverages a MLLM and issues a targeted prompt to the model, asking whether the provided image contains relevant objects. If the model confirms their presence, the area is considered ready for the corresponding task.
3. **Response generator:** The second LLM module generates the final response. The system is designed to receive user input, the intended action, the result of the validity check and the current state of the demonstration. Using this information, the system generates a natural language output for the operator.

The two LLM modules introduced previously differ in their roles and memory access. The action identifier is designed only to detect the intended action, and it uses read-only access to the chat history to avoid unnecessary information that could introduce noise. In contrast, the response generator produces a natural output for the user and updates the memory. To implement this, both modules share the same session ID but use different custom classes that inherit from the *BaseChatMessageHistory* abstract base class. Read-only access was achieved by overriding the *add_message* function so that incoming messages are not appended.

As a result, the agent explicitly encodes state transitions, rather than relying solely on the LLM's internal memory or implicit instruction-following. Figure 3 shows the difference between the monolithic and deterministic implementations.

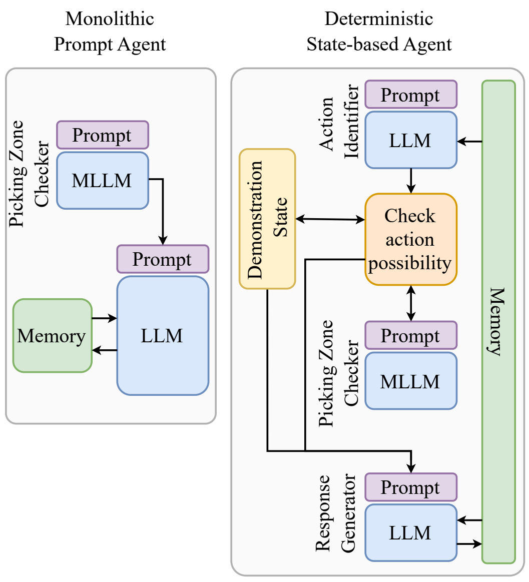

Figure 3: Comparison of the monolithic and the deterministic agent architectures

4 Experimental Setup

A set of controlled experiments was designed to evaluate the two dialogue agent designs in terms of dialogue accuracy, robustness and consistency under different operating conditions. This section describes the experimental setup. First, the use case is described, followed by the test scenarios, the models selected for text evaluation and the assessment metrics.

4.1 Use Case

The use case of this study is based on a Learning from Demonstration (LfD) framework, in which the robot acquires new skills by imitating tasks demonstrated by a human operator. For the robot to learn a complete task, the operator must perform a sequence of demonstrations, each consisting of one or more individual runs. The dialogue agent's role is to guide the operator through the demonstration recording process, ensuring that each step is carried out in a structured and consistent manner.

Figure 4 shows the robotic environment, where a FANUC arm, equipped with a suction gripper, is used to pick and place objects. The workspace is a flat surface divided into two distinct zones: a picking area and a placing area. A Photoneo camera is employed to capture 3D images of the workspace, enabling the verification of the objects' positions.

The operator conducts demonstrations by manually moving the objects from the picking zone to the placing zone. This creates a set of examples that the robot later generalises from and learns, achieving learning from demonstration.

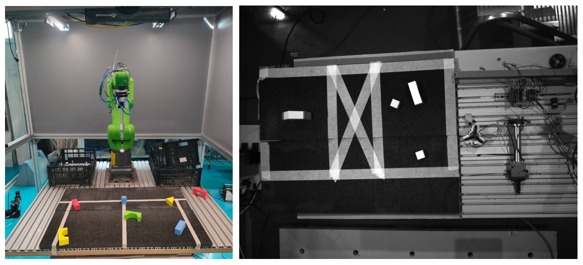

Figure 4: Picture of the FANUC robotic arm and an example image captured by the Photoneo camera

The objective of the dialogue agent is to support the operator throughout the series of demonstrations. The agent issues the action commands required for the robot to progress through the states, while also assisting the operator during the demonstrations and responding to questions when needed. For example, it can inform the operator how many demonstrations are currently being executed or what the next steps are.

To manage the interaction flow, the dialogue agent sends a predefined set of actions that communicate with the robot. The operator carries out the demonstrations, but the robot must be informed of when each step occurs. The predefined actions are the following:

- *start_demonstration_series*: Prepares the robot to begin recording a new series.
- *end_demonstration_series*: Signals that the series is finished.
- *start_demonstration*: Indicates that the operator is beginning a demonstration.
- *end_demonstration*: Marks the end of a demonstration.
- *picking_zone_prepared*: Confirms that the picking zone has been prepared (the placing zone is empty and all pieces are in the picking zone).
- *no_action*: Indicates that no action is being taken, typically when the operator asks a question that the agent should answer.

These actions must be performed in a specific order, with the agent guiding the operator to ensure that the correct steps are taken. Figure 5 illustrates the general structure of these state transitions by presenting the dialogue flow as a state graph. In summary, the process begins with the start of a series. Then, the picking zone is prepared and demonstrations are started and finished in a loop until the operator decides to stop. Once no further demonstrations are required, the series is finalised.

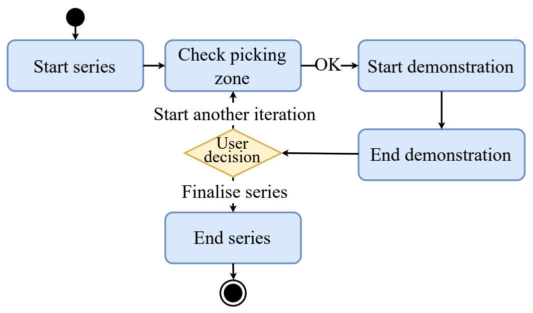

Figure 5: State Diagram

Each agent response follows a JSON format, encoding the identified action in a machine-readable structure for the robot, followed by natural language feedback in plain text for the operator.

4.2 Evaluation Scenarios

The experiments were divided into five scenarios:

1. **Normal flow:** The agent is given a complete and correctly-ordered sequence of actions: starting a series, preparing the picking zone, beginning and finishing a demonstration, and ending the series. This case verifies whether the agent can follow the intended operational path.
2. **Operator uncertainty:** Beyond executing valid actions, the agent must sustain a coherent dialogue with the operator. This scenario evaluates the agent's ability to guide uncertain operators by providing contextual assistance and directing them toward the appropriate next steps in the demonstration process.
3. **Invalid actions and order errors:** This scenario assesses the agent's ability to detect operator mistakes, issue corrective feedback, and recover to a valid interaction state when actions are missing or incorrectly ordered, such as starting a demonstration before initiating a series or attempting to end a series before ending a demonstration.
4. **Consistency after several iterations:** Multiple series and demonstrations are executed consecutively, followed by an invalid command. This scenario evaluates whether the agent maintains an accurate internal state across repeated operations and can provide correct, contextually relevant information when queried, such as the number of completed demonstrations.
5. **Vague responses and prompt injection attempts:** To reflect realistic industrial usage, the agent is exposed to short or informal inputs (e.g., "yes", "picking ok", "I'm done") to evaluate its handling of ambiguous or underspecified commands. Additionally, adversarial prompts are introduced to assess robustness against prompt injection and role manipulation, including the system's ability to preserve the required JSON structure under misleading inputs.

4.3 Models Evaluated

The evaluation process included the execution of experiments across a total of six language models, with parameter sizes ranging from 3 to 20 billion: **LLaMA 3.2 3B**, **LLaMA 3.1 8B**, **Mistral 7B**, **Qwen3 8B**, **Qwen3 14B** and **GPT-OSS 20B**, each Qwen model evaluated in both *thinking* and *no-thinking* inference modes. All models were evaluated using the same execution scripts and identical prompting conditions, with the temperature set to 0 to minimize randomness.

4.4 Evaluation Metrics

Three performance metrics were used to evaluate the models:

- **Action Selection Accuracy (ASA)**: the proportion of correctly selected next actions.
- **Message Relevance (MR)**: the proportion of responses that are consistent with the current system state and selected action, provide information that is pertinent to the user query, and do not introduce hallucinated or unsupported content.
- **Latency**: the average execution time per interaction turn, measured from receiving the user input to producing the system response.

Action Selection Accuracy and Message Relevance were evaluated by a human reviewer, while latency was measured automatically. All experiments were conducted using Ollama v0.12.9 on a workstation equipped with an AMD EPYC 74F3 CPU (9 cores, 18 threads) and an NVIDIA A10-12Q vGPU with 12 GB VRAM, running Ubuntu 22.04.5 LTS.

5 Results & Discussion

A total of 166 iterations were performed for each model, distributed across five different scenarios to evaluate performance. These experimental results

Table 1: Scenario 1 – Normal flow

	Monolithic Prompt		State-based Agent	
Model	ASA	MR	ASA	MR
LLaMA 3.2 3B	0.632	0.789	**1.000**	**1.000**
LLaMA 3.1 8B	0.579	0.632	**0.947**	**1.000**
Mistral 7B	0.737	0.789	**0.842**	**0.947**
Qwen3 8B (thinking)	**0.947**	**0.947**	**0.947**	0.895
Qwen3 8B (no-thinking)	**0.947**	**0.947**	**0.947**	**0.947**
Qwen3 14B (thinking)	0.842	0.947	**0.947**	**1.000**
Qwen3 14B (no-thinking)	0.842	0.947	**0.947**	**1.000**
GPT-OSS 20B	0.789	0.895	**0.947**	**0.947**

Table 2: Scenario 2 - Operator uncertainty

	Monolithic Prompt		State-based Agent	
Model	ASA	MR	ASA	MR
LLaMA 3.2 3B	**0.812**	0.875	0.625	**0.938**
LLaMA 3.1 8B	0.812	0.875	**0.938**	**1.000**
Mistral 7B	0.750	0.875	**0.875**	**0.938**
Qwen3 8B (thinking)	0.750	0.875	**0.875**	**0.938**
Qwen3 8B (no-thinking)	0.562	0.938	**0.875**	**1.000**
Qwen3 14B (thinking)	**0.938**	**1.000**	0.812	0.938
Qwen3 14B (no-thinking)	**0.875**	**0.938**	0.812	**0.938**
GPT-OSS 20B	0.875	**0.938**	**0.938**	**0.938**

Table 3: Scenario 3 - Invalid actions and order errors

	Monolithic Prompt		State-based Agent	
Model	ASA	MR	ASA	MR
LLaMA 3.2 3B	**0.857**	**0.857**	0.643	0.821
LLaMA 3.1 8B	0.571	0.714	**0.893**	**1.000**
Mistral 7B	0.750	0.821	**0.821**	**0.964**
Qwen3 8B (thinking)	**0.929**	**0.964**	0.857	0.893
Qwen3 8B (no-thinking)	0.643	0.643	**0.786**	**0.893**
Qwen3 14B (thinking)	0.857	0.929	**0.893**	**1.000**
Qwen3 14B (no-thinking)	0.750	0.821	**0.821**	**1.000**
GPT-OSS 20B	**0.929**	0.929	**0.929**	**1.000**

Table 4: Scenario 4 Consistency after several iterations

	Monolithic Prompt		State-based Agent	
Model	ASA	MR	ASA	MR
LLaMA 3.2 3B	0.172	**0.914**	**0.207**	0.897
LLaMA 3.1 8B	0.655	0.810	**0.966**	**1.000**
Mistral 7B	**0.897**	0.931	**0.897**	**0.948**
Qwen3 8B (thinking)	0.879	0.914	**0.966**	**0.983**
Qwen3 8B (no-thinking)	0.931	0.931	**0.966**	**0.983**
Qwen3 14B (thinking)	0.897	0.931	**0.966**	**0.983**
Qwen3 14B (no-thinking)	0.759	0.931	**0.966**	**1.000**
GPT-OSS 20B	**0.931**	**0.931**	0.828	0.914

Table 5: Scenario 5 - Vague responses and prompt injection attempts

	Monolithic Prompt		State-based Agent	
Model	ASA	MR	ASA	MR
LLaMA 3.2 3B	0.533	0.689	**0.756**	**0.911**
LLaMA 3.1 8B	0.511	0.711	**0.889**	**1.000**
Mistral 7B	0.533	0.600	**0.756**	**0.889**
Qwen3 8B (thinking)	0.800	0.911	**0.889**	**0.978**
Qwen3 8B (no-thinking)	0.778	0.844	**0.867**	**0.956**
Qwen3 14B (thinking)	**0.933**	**0.956**	0.889	**0.956**
Qwen3 14B (no-thinking)	0.578	0.689	**0.867**	**0.956**
GPT-OSS 20B	0.756	0.867	**0.867**	**0.911**

are summarised in Tables 1–5, with cells in **bold** indicating that an architecture achieved equal or superior performance compared to the other. These results are discussed in the following sections.

5.1 Monolithic Prompt Approach

In the monolithic prompt configuration, reasoning, action selection, and state tracking are handled implicitly within a single prompt, relying solely on the conversational history. Under normal flow conditions (Table 1), large-capacity models (Qwen3 14B and GPT-OSS 20B) and Qwen3 8B achieve high Action Selection Accuracy (ASA) and Message Relevance (MR), indicating that sufficient capacity can partially offset the lack of explicit structure. By contrast, smaller models, including Mistral 7B and LLaMA 3.1 and 3.2, show noticeably lower ASA and MR even in nominal workflows, suggesting difficulties in maintaining coherent task progression.

In scenarios involving operator uncertainty (Table 2) and invalid or out-of-order actions (Table 3), monolithic prompts show inconsistent behaviour: some models benefit from the flexibility of unconstrained reasoning, while others suffer sharp drops in action accuracy due to error propagation and misinterpretation of intent.

In Table 4, performance degradation is particularly evident for LLaMA 3.2 3B, highlighting that robustness in extended interactions depends largely on model scale rather than architectural guarantees.

Finally, in scenarios involving vague inputs and prompt injection attempts (Table 5), monolithic prompting exhibits its most pronounced weaknesses. While Qwen3 14B operating in thinking mode shows strong resistance to adversarial inputs and informal expressions commonly encountered in practice (e.g., "I'm done"), smaller models frequently select incorrect actions. In general, both versions of Qwen3 operating in thinking mode, achieve strong overall performance across all scenarios, sometimes outperforming the agent.

5.2 State-Agent Approach

In the state-based agent configuration, interactions are mediated by an external controller that enforces a deterministic finite-state machine, separating action validation from language generation. This architectural separation generally improves ASA and MR across the evaluated models.

Under normal operation (Table 1), the state-based agent achieves near-perfect ASA and MR for all models, including smaller architectures. This shows that explicit state tracking effectively compensates for weaker instruction-following capabilities, allowing models with limited capacity to follow the intended operational flow reliably.

When operator uncertainty is introduced (Table 2), the state-based agent maintains high MR while consistently improving ASA relative to the monolithic prompt. An exception occurs with Qwen3 14B, where ASA under the state-based agent is lower than under the monolithic prompt, despite MR remaining high.

In scenarios involving invalid actions and ordering errors (Table 3), the state-based agent significantly improves recovery performance. Notably, smaller models such as LLaMA 3.1 8B and Mistral 7B achieve performance comparable to larger models when coupled with the state-based controller, a result not observed in the monolithic prompt configuration, where performance varies strongly with model scale.

In long-horizon interactions (Table 4), the state-based agent preserves consistent performance over multiple demonstration cycles. While the smaller LLaMA 3.2 3B shows some performance degradation, all other models sustain high ASA and MR.

Finally, in the presence of vague commands and prompt injection attempts (Table 5), the state-based agent exhibits markedly higher robustness than the monolithic prompt. By constraining permissible actions through explicit state validation, the system limits the effect of adversarial or underspecified input, reducing hallucinated or unsupported behaviour even in smaller models.

An exception is observed for Qwen3 14B in thinking mode, where the state-based agent does not consistently improve action selection. This reflects a broader pattern evident across all scenarios. Models in thinking mode tend to exhibit more ambiguous and variable behavior because their internal deliberation can both enhance reasoning and planning while occasionally misaligning with task constraints, making action selection less predictable.

5.3 Comparative Analysis

Figure 6 compares the behavioural performance of the monolithic prompt and state-based agent, showing the ASA and MR averaged across all evaluated scenarios. For all considered models, the state-based agent consistently outperforms the monolithic prompt configuration on both metrics, indicating more reliable action selection and more state-consistent responses. Improvements in ASA are particularly pronounced for models with weaker instruction-following capabilities, such as LLaMA and Mistral, where explicit state tracking substantially reduces incorrect or out-of-order action selection. GPT-OSS 20B, the largest evaluated model, as well as Qwen3 operating in thinking mode, already achieve relatively high ASA under the monolithic prompting approach; however, the state-based agent still provides consistent gains. A similar trend is observed for MR, where the state-based agent produces responses that are more consistently aligned with the current system state and selected actions, indicating a reduction in hallucinated or unsupported content.

Figure 7 illustrates the corresponding latency trade-offs. It is worth noting that comparing latency across models is most informative for highlighting relative differences rather than absolute values, as these measurements are strongly influenced by computational hardware and system-level factors. Consequently, the use of higher-performance equipment may yield substantially improved results. The state-based agent incurs higher average execution times across all models due to additional reasoning steps and explicit state validation. This overhead remains modest for smaller models (approximately 2–5 seconds) but becomes more pronounced for larger architectures, especially GPT-OSS 20B and Qwen3 14B in thinking mode. Using the no-thinking configurations substantially reduces latency while preserving most of the gains in ASA and MR. On the other hand, using the state-based agent with smaller models helps to achieve comparable or slightly higher ASA and MR scores while maintaining substantially low in-

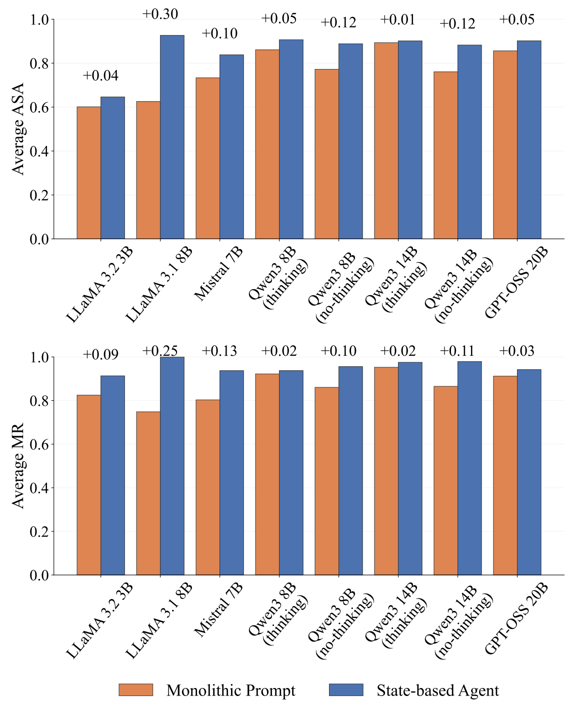

Figure 6: Comparative evaluation of State-based Agent vs Monolithic Prompt across Action Selection Accuracy (top) and Message Relevance (bottom).

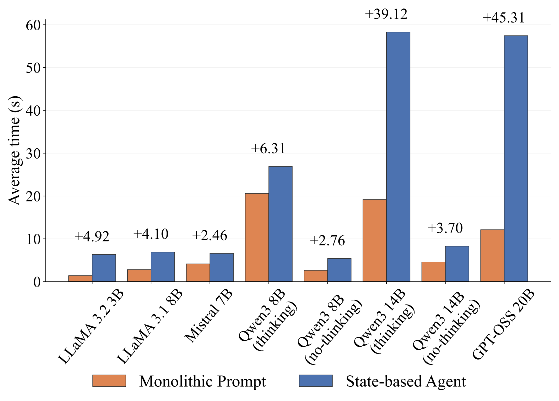

Figure 7: Average execution time per interaction for each model.

ference times. For instance, the state-based agent with LLaMA 3.1 8B matches or exceeds the ASA and MR of the monolithic prompt using Qwen3 in thinking mode, while exhibiting significantly low latency due to the smaller underlying model. This shows that explicit state management can compensate for reduced model scale, enabling efficient yet high-quality performance.

Overall, these results demonstrate that the state-based agent architecture improves both action correctness and response quality across models at the cost of increased latency. Notably, smaller models, which exhibit the lowest inference times under both approaches, benefit most from the state-based agent architecture, achieving substantial performance gains while maintaining practical response times. Among these, Qwen3 8B operating without thinking mode represents a particularly effective compromise, combining strong performance with low latency, and thus offering a favorable trade-off for real-time interactive settings.

6 Conclusions and Future Work

In this work, we compared monolithic prompt-based agents and deterministic state-based agents for guiding human operators in industrial robotic scenarios. The results show that state-based agents consistently outperform monolithic prompt-based approaches in terms of action correctness and response relevance across all evaluated models. By explicitly encoding task state and action constraints, the proposed architecture enables smaller language models to achieve performance comparable to, or exceeding, that of larger models operating under monolithic prompting, while maintaining reduced inference latency. However, an important exception arises for configurations operating in thinking mode or relying on large-capacity models. Although such configurations achieve strong performance, their elevated inference latency limits their suitability for real-time robot–human interaction, where timely responses are essential.

Excluding these cases, the state-based agent demonstrates that appropriate architectural design choices can effectively compensate for reduced model capacity, offering a practical alternative to reliance on increasingly large and computationally expensive models.

To further extend these results, future work could focus on enhancing the state-agent's robustness and usability. One potential direction is the integration of mechanisms to cancel ongoing demonstrations and perform rollbacks, thereby giving users greater control over the interaction. In addition, user studies with human operators are needed to assess usability and interaction effectiveness in real-world settings.

Acknowledgments

This research was supported by the Elkartek Programme under grant number KK-2024/00050 (IKASPROD) sponsored by the Basque Government.

References

Sukai Huang, Nir Lipovetzky, and Trevor Cohn. 2025. Planning in the dark: Llm-symbolic planning pipeline without experts. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR.

Yeseung Kim, Dohyun Kim, Jieun Choi, Jisang Park, Nayoung Oh, and Daehyung Park. 2024. A survey on integration of large language models with intelligent robots. *Intelligent Service Robotics*, 17(5):1091–1107.

LangChain. Langchain (version 0.3.27). `https://www.langchain.com/`.

Daniel Mendoza, Christopher Hahn, and Caroline Trippel. 2024. Translating natural language to temporal logics with large language models and model checkers. In *2024 Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–11.

Ollama. Ollama (version 0.12.9). `https://ollama.com/`.

Ziyi Yang, Shreyas Raman, Ankit Shah, and Stefanie Tellex. 2024. Plug in the safety chip: Enforcing constraints for llm-driven robot agents. pages 14435–14442.