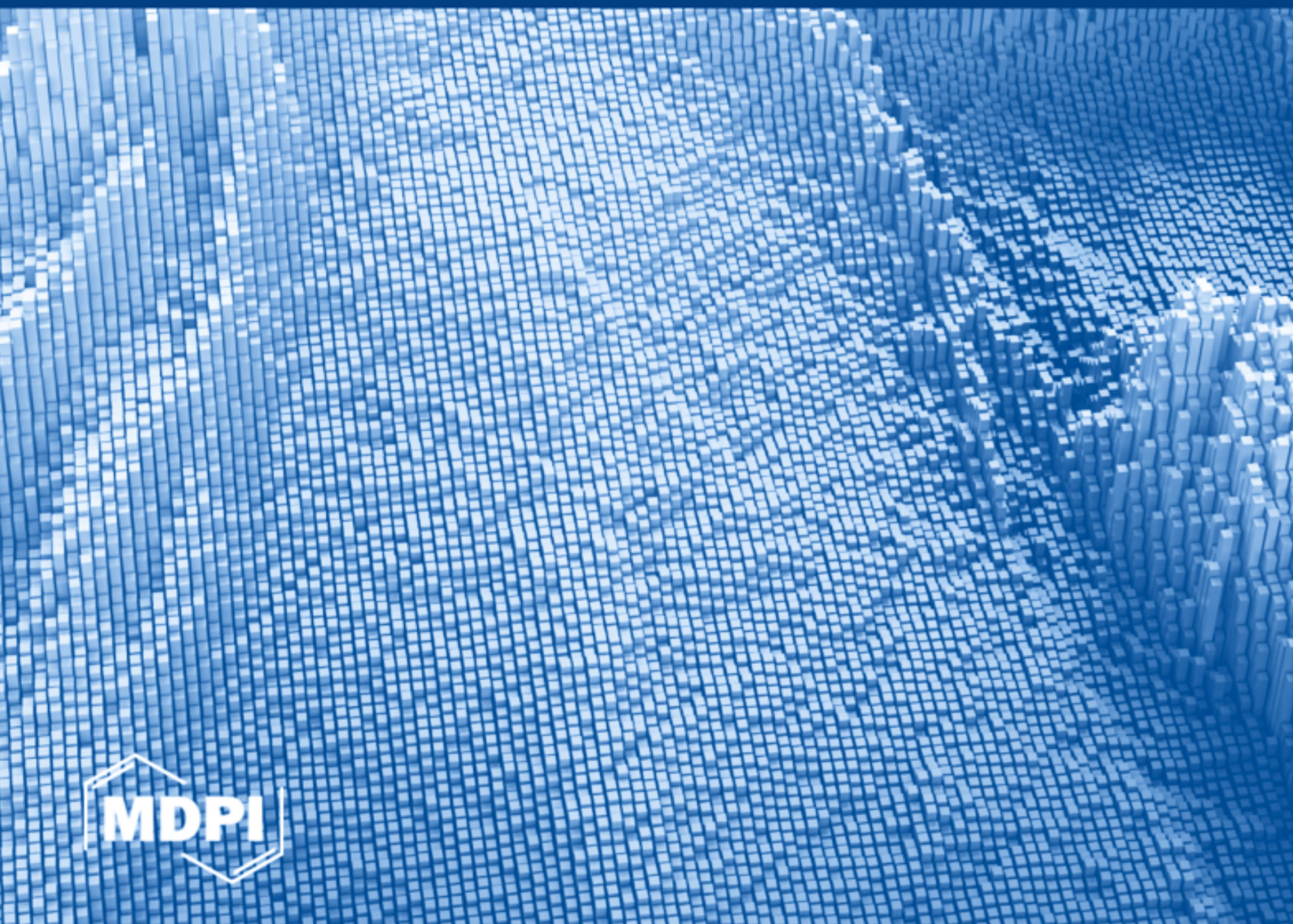
machine learning & knowledge extraction

Impact Factor 6.0
CiteScore 9.9

ISSN 2504-4990

Plug-and-Play LLM Knowledge Extraction for Robot Navigation: A Fine-Tuning-Free Edge Framework

Volume 8 · Issue 2 February 2026



MDPI

machine learning & knowledge extraction

IMPACT FACTOR 6.0

CITESCORE 9.9

Article

Plug-and-Play LLM Knowledge Extraction for Robot Navigation: A Fine-Tuning-Free Edge Framework

Sebastian Rojas-Ordoñez, Mikel Segura, Irune Yarza, Veronica Mendoza and Ekaitz Zulueta

MDPI

https://doi.org/10.3390/make8020049

machine learning & knowledge extraction

MDPI

Article

Plug-and-Play LLM Knowledge Extraction for Robot Navigation: A Fine-Tuning-Free Edge Framework

Sebastian Rojas-Ordoñez [1,2,*], Mikel Segura [1], Irune Yarza [1], Veronica Mendoza [2] and Ekaitz Zulueta [2]

[1] IKERLAN Technology Research Centre, Paseo José María Arizmendiarrieta 2, 20500 Arrasate/Mondragón, Gipuzkoa, Spain; msegura@ikerlan.es (M.S.); iyarza@ikerlan.es (I.Y.)
[2] Department of Systems Engineering and Automation, Faculty of Engineering—Vitoria-Gasteiz, University of the Basque Country (UPV/EHU), Nieves Cano 12, 01006 Vitoria-Gasteiz, Álava, Spain; ekaitz.zulueta@ehu.eus (E.Z.)
* Correspondence: jsrojas@ikerlan.es

Abstract

Large Language Models are increasingly used for high-level robotic reasoning, yet their latency and stochasticity complicate their direct use in low-level control. Moreover, extracting actionable navigation cues from multimodal context incurs inference costs that are challenging for embedded platforms. We present a plug-and-play framework that augments a finite-state machine with asynchronous velocity suggestions generated by a Large Language Model, using an off-the-shelf DistilGPT-2 model running on-device on a Jetson AGX Orin. The system extracts task-relevant cues from the current context and integrates them only if they satisfy deadline, schema, and kinematic validation, thereby preserving a deterministic 50 Hz control loop with a <5 ms fallback path. We compare multiple Large Language Models for embedded robot control and quantify trade-offs among model size, inference time, and output validity. To assess whether the Large Language Models add value beyond signal processing, we include an ablation against a standard smoothing baseline; the results indicate that the Large Language Models contribute anticipatory, context-dependent adjustments that are not captured by filtering alone. Experiments in Gazebo and on a real TurtleBot3 reduce the final position error from 0.246 m to 0.159 m and improve trajectory efficiency from 0.821 to 0.901 without increasing control-loop latency. Approximately 80% of the Large Language Models' outputs pass validation and are applied. Overall, the framework reduces developer effort by enabling behavioral changes at the prompt level while maintaining interpretable, robust edge-based navigation.

Keywords: knowledge extraction; mobile robotics; edge AI; Large Language Models; robot navigation; prompt engineering; developer accessibility

Check for updates

Academic Editor: Mehmed Kantardzic

Received: 20 January 2026
Revised: 15 February 2026
Accepted: 19 February 2026
Published: 21 February 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Developing autonomous mobile robots that can navigate and make decisions has traditionally required substantial expertise in control theory, state machines, perception, and real-time software [1]. Even relatively simple tasks, such as following traffic signs, often entail extensive integration work: designing vision pipelines, tuning parameters, and performing iterative testing in simulation and hardware. These requirements create a high barrier to entry and slow down prototyping and deployment.

Large Language Models (LLMs) offer a complementary mechanism for specifying and adapting behavior. Rather than encoding intent exclusively through hand-crafted logic, developers can express task goals in natural language and leverage the model to infer

context-dependent suggestions [2]. Prior work has used LLMs for high-level planning, task decomposition, and manipulation [3]. However, many existing approaches rely on cloud inference, specialized hardware, or model fine-tuning, which reduces portability and is often incompatible with embedded mobile platforms operating under strict power and latency budgets.

Mobile robot navigation imposes a non-negotiable constraint: control must meet millisecond-scale deadlines to remain safe under sensing uncertainty and dynamic environments [4]. While Vision Language Action (VLA) models [5] demonstrate end-to-end coupling between perception and action, they typically require substantial computation and can be difficult to interpret and constrain.

Directly integrating an LLM into the low-level control loop is therefore problematic due to high inference latency and stochastic outputs [6]; missed deadlines and malformed outputs can compromise safety and robustness. A practical alternative is a modular integration paradigm that separates perception, language-based reasoning, and deterministic control, preserving real-time guarantees while leveraging language-driven guidance only where it demonstrably adds value.

In this work, we treat language as a source of high-level guidance that complements rather than replaces a deterministic controller. We introduce a modular framework that augments a conventional Finite-State Machine (FSM) with asynchronous LLM suggestions, accepted only when timely and safe. Concretely, the FSM maintains a strict 50 Hz control loop with a <5 ms fallback path. In parallel, an off-the-shelf LLM (without fine-tuning) proposes structured adjustments to (v_x, ω_z) via prompt templates. The model is instructed to return a lightweight JavaScript Object Notation (JSON) object (e.g., `{"linear_x": 0.05, "angular_z": −0.10}`), which serves as the exchange format for knowledge extraction. A validator enforces JSON well-formedness, deadline compliance, and explicit kinematic bounds before merging an accepted suggestion with the baseline command. This design preserves the FSM's deterministic safety guarantees while allowing the LLM to provide context-aware refinements during approach phases.

We evaluate the framework in a 5 × 5 m Maze Arena, both in Gazebo (a robotics simulation environment) simulation and on a physical TurtleBot3 Burger (a differential-drive mobile robot platform). Across $N = 60$ trials per condition, the LLM-augmented controller reduces the final position error from 0.246 m to 0.159 m and improves trajectory efficiency from 0.821 to 0.901, without affecting control-loop latency. Approximately 80% of LLM outputs pass validation and are applied. Finally, an ablation against a smoothing baseline indicates that the observed gains are not explained by filtering alone, but arise from anticipatory, context-dependent adjustments.

The main contributions of this work are as follows:

- **Plug-and-play knowledge extraction:** A modular framework that augments existing Robot Operating System 2 (ROS 2) navigation with off-the-shelf LLM guidance, without model fine-tuning or additional data collection.
- **Latency-aware edge architecture:** A validation layer and deterministic fallback mechanism that preserves a stable 50 Hz control loop while accommodating asynchronous LLM inference on embedded hardware (Jetson Orin).
- **Verified performance gains:** In both simulation and real-world experiments, the augmented controller improves positioning accuracy (0.246 → 0.159 m) and trajectory efficiency (0.821 → 0.901) relative to the baseline.
- **Edge ML efficiency analysis:** An ablation study shows that a distilled Small Language Model (SLM) (DistilGPT-2 (a distilled GPT-2 language model)) extracts sufficient navigational context to outperform a classical smoothing baseline, supporting the necessity of the LLM component.

https://doi.org/10.3390/make8020049

- **Accessibility focus:** The framework reduces developer effort by enabling behavioral tuning through prompt modifications rather than extensive code changes.

Overall, the paper provides a practical and interpretable pathway for integrating LLM guidance into embedded robot navigation. The remainder of the paper is structured as follows: Section 2 reviews related work, Section 3 presents the system architecture, Section 4 details the implementation, Section 5 describes the experimental setup, Section 6 examines the results, including those of the ablation study, Section 7 discusses limitations, and Section 8 concludes the paper.

2. Related Work

This section reviews prior work relevant to integrating language models into robotic navigation. We discuss conventional engineering approaches (state machines, perception stacks, and Robot Operating System (ROS) tooling), LLMs and multimodal policies for decision-making, navigation-oriented language and vision–language methods, and the safety and real-time constraints that dominate embedded deployment in ROS 2. We also highlight a frequently under-reported dimension, developer accessibility, encompassing implementation effort, debugging overhead, and prompt-iteration cost. The discussion emphasizes trade-offs among training requirements, deployment modality (cloud vs. on-device), control-loop guarantees, and the practicality of knowledge-extraction mechanisms for embedded robotics.

2.1. Conventional Robot Programming Challenges

Mobile robot navigation has traditionally been implemented using explicit state machines, classical controllers (e.g., PID), and hand-engineered perception pipelines [7]. Building such systems requires expertise in control, computer vision, and real-time software engineering, and even relatively simple behaviors often involve substantial integration and tuning effort. Middleware frameworks such as ROSs [8] improve modularity but introduce their own learning curve, including message interfaces, launch systems, and debugging tools. As a result, robotics codebases can become difficult to maintain and extend, especially when perception and control components must be co-tuned across simulation and hardware [9].

2.2. LLMs and Multimodal Models in Robotics

A growing body of work leverages LLMs and vision–language models for high-level reasoning, planning, and manipulation. CLIPort [10] combines CLIP-based representations with Transporter-style networks for language-conditioned tabletop manipulation, while SayCan [11] grounds LLM outputs in skill primitives and value estimates to execute long-horizon instructions. The RT family (RT-1 [12], RT-2 [5], RT-X [13]) illustrates policy generalization through large-scale data collection and training. PaLM-E [14] further integrates multimodal perception with language-based reasoning. More recently, VLA models such as π_0 and $\pi_{0.5}$ [15,16] report strong generalization across heterogeneous robot data. Despite their capabilities, these systems often require substantial computation and curated datasets, which can limit portability and make on-device deployment challenging.

2.3. Language-Grounded Navigation

Vision Language Models (VLMs) have also been applied to navigation [17]. LM-Nav [18] composes pretrained vision and language modules to execute outdoor navigation tasks. ViNT [19] trains a generalist navigation transformer across diverse trajectories and reports transfer across environments. VLMaps [20] and LERF [21] incorporate language into 3D scene representations for flexible goal specification. More direct LLM-based navigation

has been explored by NavGPT [22] using large proprietary models for zero-shot reasoning and through long-horizon instruction-following on legged robots [23]. While promising, many of these approaches face practical limitations for embedded control, including high inference latency, sensitivity to prompt design, and limited real-time safety guarantees.

Several studies pursue "plug-and-play" integration to reduce training requirements. DriveMLM, for example, applies LLMs to autonomous driving and zero-shot robotic control [24]. However, prompt design can become a bottleneck: reasoning-style prompting (e.g., chain-of-thought) can affect performance substantially [25], yet discovering effective prompts often requires iterative experimentation and domain knowledge [26]. This motivates methods that constrain outputs, reduce prompt fragility, and provide robust validation mechanisms.

2.4. Deployment Challenges: Safety, Real-Time Constraints, and Resource Limitations

Safety and determinism are central challenges when deploying LLM-driven components on physical robots [27]. Prior analyses report that LLM/VLM-based policies can degrade under perturbations and distribution shift [28].

Fundamental deployment barriers. Beyond average inference latency, several factors hinder direct deployment of recent LLM/VLM/VLA methods on embedded mobile robots. First, token-level decoding is often variable under common sampling strategies, which complicates reproducibility, regression testing, and certification-oriented workflows when the model is placed in a closed-loop system. Second, hallucinations and structured-output violations (e.g., malformed fields, missing keys, or numerically inconsistent values) can produce unusable or unsafe actions unless strict output constraints and runtime validation are enforced [27,28]. Third, memory and computation requirements can exceed embedded budgets or induce thermal throttling, especially when language inference must co-execute with perception workloads on shared accelerators. Finally, systems-level effects in ROS 2 (message queuing, executor scheduling, clock alignment, and jitter under heterogeneous load) can amplify variability and complicate timing guarantees [29–31].

Real-time constraints further complicate integration: LLM inference latencies can reach hundreds of milliseconds, which conflicts with millisecond-scale control deadlines [32,33]. Existing reports on LLM–ROS 2 integration [34] remain fragmented and often provide limited guidance on how to combine slow, stochastic reasoning with fast, deterministic control in a principled and reproducible manner. These constraints motivate architectures in which language-based reasoning is treated as opportunistic guidance, gated by deterministic mechanisms (schema checks, bounded actions, freshness policies, and a fast fallback path), rather than embedded directly into the low-level loop.

Small language models in robotics. Despite potential advantages for resource-constrained platforms, small language models (sub-billion to a few billion parameters) remain comparatively understudied in robotics. Community benchmarks and flagship demonstrations prioritize cloud-scale models and long-horizon reasoning, while embedded-centric metrics (peak memory usage, energy per query, latency percentile distributions under concurrent load) are infrequently reported. Tooling for constrained decoding and structured output validation has matured primarily in large-model ecosystems, whereas lightweight on-device deployments often rely on generic generation without strong guarantees. Additionally, evaluation protocols that combine real-time control requirements with language-model reliability dimensions (format correctness, action boundedness, temporal freshness) remain unstandardized. These gaps motivate systematic investigation of small models under embedded constraints with explicit safety mechanisms. In this work, we specifically study whether a lightweight generative model can provide bounded, structured refinements under explicit validation, rather than acting as a standalone policy.

2.5. Developer-Centered Perspectives

Most robotics studies prioritize robot-level metrics (e.g., success rate, trajectory error, or completion time), whereas developer-centered metrics are rarely reported. Measures such as implementation time, debugging effort, and the number of prompt iterations are uncommon in empirical evaluations [35]. Surveys and perspective articles highlight this gap and call for methods that reduce programming complexity and support practical adoption [36,37]. From this viewpoint, a method may improve navigation accuracy yet remain impractical if it requires specialized training pipelines, extensive dataset collection, or large-scale computation resources. Accordingly, we treat prompt editing as the primary tuning interface and report developer-centered indicators alongside robot-level outcomes in our evaluation.

In contrast to prior studies, our work targets the interface between semantic guidance and real-time control. We propose a modular, plug-and-play framework that integrates LLM-based knowledge extraction into ROS 2 navigation without fine-tuning while preserving a deterministic 50 Hz control loop via a sub-5 ms fallback path. By employing a lightweight model (DistilGPT-2) on-device, we avoid cloud dependencies and mitigate latency bottlenecks. Finally, by reporting both robot-level outcomes and developer-centered indicators, we complement performance-driven evaluations with an accessibility-oriented perspective.

To clarify these differences, Table 1 summarizes representative approaches by domain, training requirements, control considerations, and accessibility-related aspects.

Table 1. Comparison of representative language- or multimodal-model robotics approaches with this work.

Work	Domain	Training Required	Deployment	Real-Time Safety Mechanism	Developer Effort Emphasis
CLIPort [10]	Manipulation	Supervised training on task data	On-device/server	Not a focus	Not reported
SayCan [11]	Mobile manipulation	Skill/value grounding + pretrained components	Typically server-class	Heuristic/partial (grounding)	Not reported
RT-1/2/X [5,12,13]	Manipulation	Large-scale dataset training	Server-class	Not a focus	Not reported
PaLM-E [14]	Multimodal reasoning	Large multimodal model	Typically server-class	Not a focus	Not reported
LM-Nav [18]	Outdoor navigation	Composition of pretrained modules	Often server-class	Limited (latency-bound)	Not reported
NavGPT [22]	Indoor navigation	None (zero-shot with proprietary LLM)	Cloud/API	No hard guarantees reported	Not reported
DriveMLM [24]	Autonomous driving	No fine-tuning (prompting)	Often server-class	Simulated/heuristic safety layer	Not reported
This Work	**Mobile navigation**	**None (off-the-shelf SLM)**	**On-device (Jetson)**	**Deadline + schema + kinematic validation with fallback**	**Reported qualitatively and via prompt-based tuning**

3. System Architecture

This section presents the overall architecture of our proposal, which integrates LLM-based knowledge extraction into mobile robot navigation while preserving the safety guarantees of a conventional deterministic controller. The architecture follows a modular, neuro-symbolic paradigm, layering logic-based control with data-driven reasoning. This layered approach ensures seamless integration with ROS2 and real-world execution on

edge devices. Here we provide a high-level overview; all technical components and timing mechanisms are explained in detail in Section 4.

3.1. Overall Framework

The framework transforms classical robot navigation into a modular, language-augmented pipeline. Instead of replacing deterministic controllers with end-to-end neural policies, the LLM acts as an asynchronous reasoning layer on top of a reliable Finite-State Machine (FSM). This hybrid design ensures that the robot always maintains stable, high-frequency control (symbolic layer) while gaining adaptive, context-aware suggestions from the LLM (neural layer).

As shown in Figure 1, the framework is organized into three main layers:

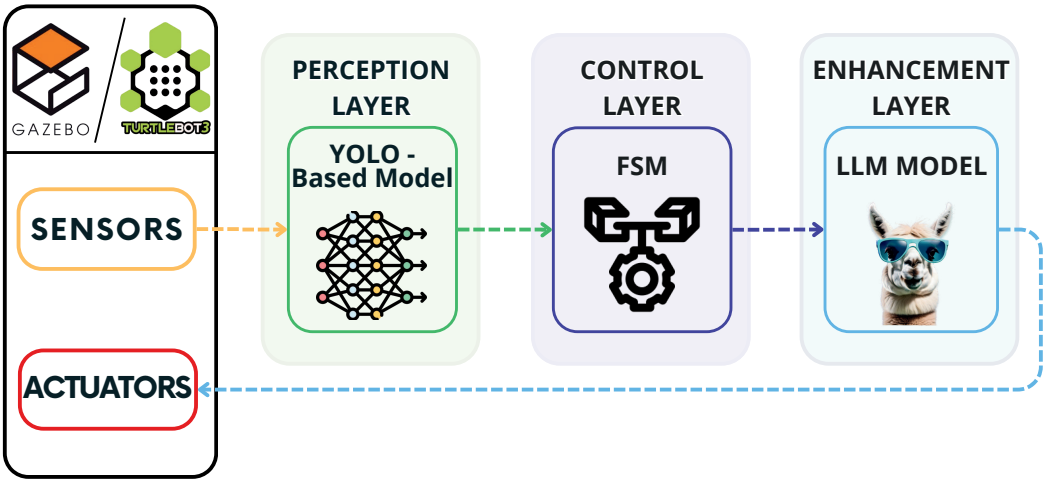


Figure 1. Overall system architecture illustrating the separation between the deterministic control loop (FSM) and the asynchronous knowledge-extraction loop (LLM).

3.2. Perception Layer

The Perception Layer captures the environmental context through a vision-based detector. A lightweight Convolutional Neural Network (CNN) model processes camera input to identify relevant traffic signs in real time. The detector provides three key outputs: class, estimated distance, and lateral offset. This design is modular, allowing the framework to incorporate different sensors or perception modules without requiring changes to the reasoning or control logic.

3.3. Control Layer

The Control Layer is centered on a deterministic FSM. Each state encodes a specific navigation behavior, and transitions are triggered strictly by perception events or safety timers. This provides a robust rule-based structure that guarantees the robot can operate safely at 50 Hz, even if the upper reasoning layer experiences latency or failure.

3.4. Knowledge-Extraction Layer

The Knowledge-Extraction Layer (formerly Enhancement Layer) introduces LLM-based reasoning as a plug-and-play module. Structured prompts summarize the current FSM state, sign detection, and baseline command to form a semantic context. The LLM processes this context to extract actionable velocity refinements. Crucially, this layer operates asynchronously: the high-latency LLM inference never blocks the high-frequency FSM loop. Suggestions are validated and fused only when available; otherwise, the system seamlessly defaults to the baseline control.

3.5. Prompt Engineering Strategy

The prompt is dynamically adapted to the robot's state to maximize relevance. The template includes the current FSM phase, the detected sign class, spatial metrics (distance, offset), and the baseline velocity command. The LLM is instructed to act as a "velocity optimizer," outputting only valid velocity pairs (v_x, ω_z) within a strict JSON schema. This strategy simplifies parsing and mitigates the risk of hallucinated commands. Prompts are requested only during the `APPROACH` phase, where fine-grained velocity tuning provides the largest efficiency benefits.

4. Implementation Details

This section describes the technical implementation of our framework, with emphasis on deploying knowledge-extractio components on embedded hardware. We detail the hardware constraints, the asynchronous software architecture, and the ROS 2 integration used to validate the system in both Gazebo simulation and on a physical TurtleBot3 platform.

4.1. Hardware Setup

The robotic platform is a TurtleBot3 Burger equipped with an Intel RealSense D435i Red–Green–Blue plus Depth (RGB-D) camera. The complete stack runs on an NVIDIA Jetson AGX Orin (an embedded AI computing platform) (64 GB RAM, Ampere GPU), enabling on-device execution of both the You Only Look Once (YOLO)-based perception module (vision) and the DistilGPT-2 inference module (language). Running locally avoids cloud dependencies, reduces network-induced delays, and supports privacy-preserving deployment.

4.2. Software Framework

The system is implemented in ROS 2 Humble and decomposed into modular nodes that separate deterministic control from stochastic language-based suggestions.

4.2.1. Perception Node

The perception node captures synchronized RGB-D streams from the Intel RealSense D435i (Figure 2). RGB frames are processed by a YOLOv8-based detector trained on the three traffic signs used in the Maze Arena (*left arrow*, *right arrow*, *stop*). The trained model is exported to Open Neural Network Exchange (ONNX) and deployed as an independent ROS 2 node. It publishes structured outputs: (i) `/sign_class` (detected label, `std_msgs/msg/String`); (ii) `/sign_distance` (estimated range; see Section 3); (iii) `/sign_offset` (normalized lateral displacement w.r.t. the image center); (iv) `/sign_detection/annotated_image` (overlay image, `sensor_msgs/msg/Image`).

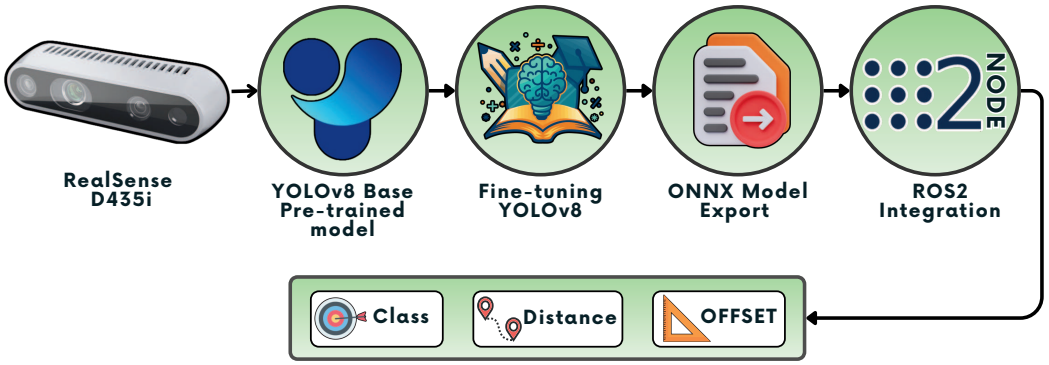


Figure 2. Perception node pipeline.

4.2.2. FSM Node (Deterministic Controller)

The FSM (Figure 3) publishes commands at a fixed 50 Hz (20 ms period). The LLM runs asynchronously and produces a new suggestion whenever inference completes (average ≈186 ms in our setup). Crucially, suggestions are not required to arrive within the next 20 ms tick. Instead, when a valid suggestion arrives, it is applied at the next available FSM tick and then held constant (sample-and-hold) until either (i) a newer valid suggestion becomes available, or (ii) a time-to-live (TTL) expires. This policy ensures that stale suggestions are never applied and also preserves deterministic 50 Hz actuation. The FSM enforces speed limits, collision-aware slowdown, and validated state transitions.

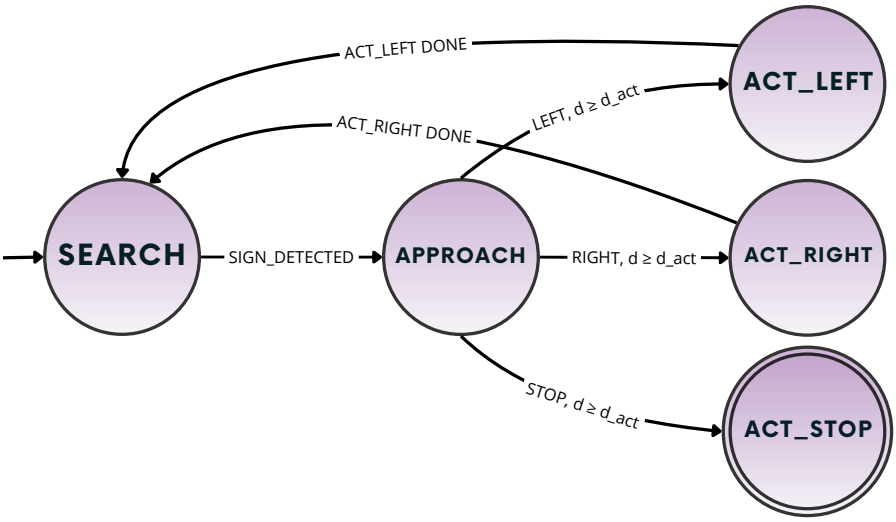


Figure 3. Finite-State Machine (FSM) logic for deterministic control.

Five navigation states are defined: `SEARCH`, `APPROACH`, `ACT_LEFT`, `ACT_RIGHT`, and `ACT_STOP`. During `APPROACH`, the FSM publishes a compact JSON summary on `/fsm_state` containing {`stamp`, `phase`, `class`, `distance`, `offset`, `base_v`, `base_w`, `deadline_ms`}. The LLM module consumes this summary only in `APPROACH`; in all other states, the baseline command is applied without language intervention.

At each control tick, the FSM computes a baseline velocity pair

$$\mathbf{v}_{\text{base}} = (v_x, \omega_z), \tag{1}$$

derived from distance and alignment errors and saturated within bounds:

$$v_x = \text{sat}\Big(k_v(d - d_{\text{goal}}),\ v_{\min},\ v_{\max}\Big), \qquad \omega_z = \text{sat}\big(k_\omega e_\theta,\ \omega_{\min},\ \omega_{\max}\big), \tag{2}$$

where d is the estimated range to the target sign, d_{goal} is the desired stopping distance, and e_θ is the alignment error (e.g., normalized horizontal offset of the sign centroid). The saturation operator is

$$\text{sat}(x, a, b) = \begin{cases} a & \text{if } x < a, \\ x & \text{if } a \le x \le b, \\ b & \text{if } x > b. \end{cases} \tag{3}$$

Finally, first-order rate limiters are applied to Δv_x and $\Delta \omega_z$ to reduce abrupt changes. The baseline command is always available as a deterministic fallback and provides the reference used when validating LLM suggestions during `APPROACH`.

4.2.3. LLM Node (Knowledge-Extraction/Refinement Layer)

The LLM node runs asynchronously in a separate ROS 2 process so that inference never blocks the 50 Hz control loop (Figure 4).

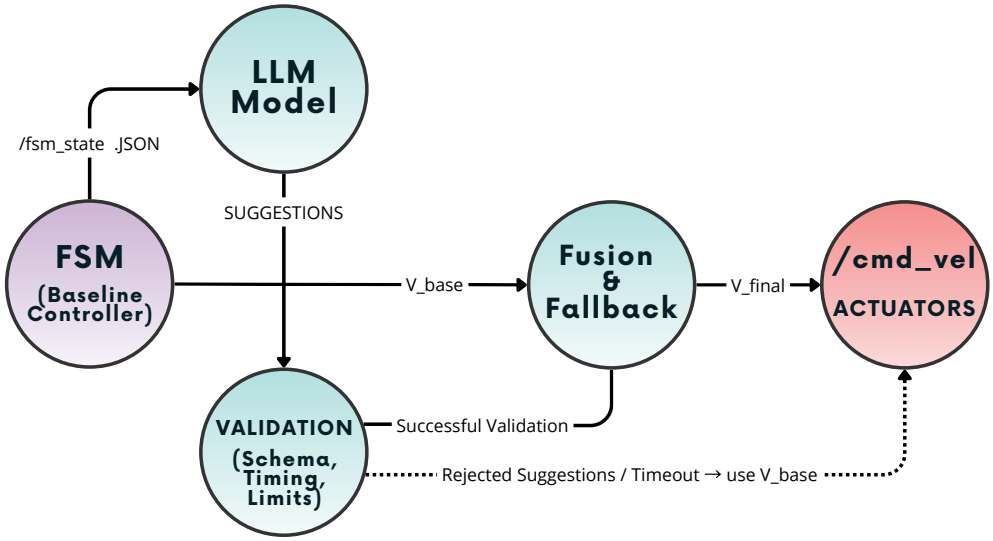


Figure 4. LLM node internal architecture: prompting, validation, and fusion.

It subscribes to `/fsm_state` and outputs *candidate* velocity refinements as a strict JSON object with fields `linear_x` and `angular_z` on `/llm_suggestion` (`std_msgs/msg/String`). The actuation topic `/cmd_vel` is published *only* by the FSM after validation and fusion, keeping the safety-critical path deterministic and auditable.

Interfaces. The LLM node uses the following:

- Input: `/fsm_state` (`std_msgs/msg/String`, JSON payload), e.g.,:

```
{
  "stamp": <unix>,
  "phase": "APPROACH",
  "class": "LEFT",
  "distance": 0.92,
  "offset": -0.12,
  "base_v": 0.10,
  "base_w": 0.05,
  "deadline_ms": 20
}
```

- Output: `/llm_suggestion` (`std_msgs/msg/String`, JSON payload); optional diagnostics on `/llm/metrics`.

Dual-rate timing and freshness (sample-and-hold with TTL). Because language-model inference is slower than the 50 Hz control loop, the FSM never waits for the LLM. The LLM produces suggestions asynchronously; once a suggestion is available, it can be applied at subsequent control ticks under a sample-and-hold policy until a newer suggestion arrives or a time-to-live (TTL) expires. Each suggestion carries the originating state timestamp `stamp`. A suggestion is considered *fresh* if its age satisfies

$$t_{\text{recv}} - t_{\text{stamp}} \leq T_{\text{TTL}}; \tag{4}$$

otherwise it is discarded as stale. T_{TTL} is selected from on-device latency profiling (and should be re-tuned when porting to different platforms or workloads). Figure 5 illustrates the resulting dual-rate behavior.

Validation. Before fusion, each suggestion is checked for:

- Schema correctness: well-formed JSON with numeric `linear_x` and `angular_z`;
- Freshness (TTL): suggestion age satisfies Equation (4);
- Kinematic admissibility: $|v_x| \leq v_{\max}$ and $|\omega_z| \leq \omega_{\max}$ (optional rate-limit checks).

Suggestions are requested only in the `APPROACH` state, where velocity refinement is most relevant.

Fusion and fallback. Let $\mathbf{v}_{\text{base}} = (v_x^{\text{base}}, \omega_z^{\text{base}})$ and $\mathbf{v}_{\text{llm}} = (v_x^{\text{llm}}, \omega_z^{\text{llm}})$ (mapped from JSON fields `linear_x` and `angular_z`). Validated suggestions are projected onto the admissible set $\mathcal{S} = [v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}]$ and merged as:

$$\mathbf{v}_{\text{final}} = (1 - \alpha)\, \mathbf{v}_{\text{base}} + \alpha\, \Pi_{\mathcal{S}}(\mathbf{v}_{\text{llm}}), \qquad \alpha \in [0, 1], \tag{5}$$

with $\alpha = 0.5$ in our experiments. If validation fails or no fresh suggestion is available, the FSM applies $\mathbf{v}_{\text{base}}$ immediately (deterministic fallback), preserving strict 50 Hz actuation.

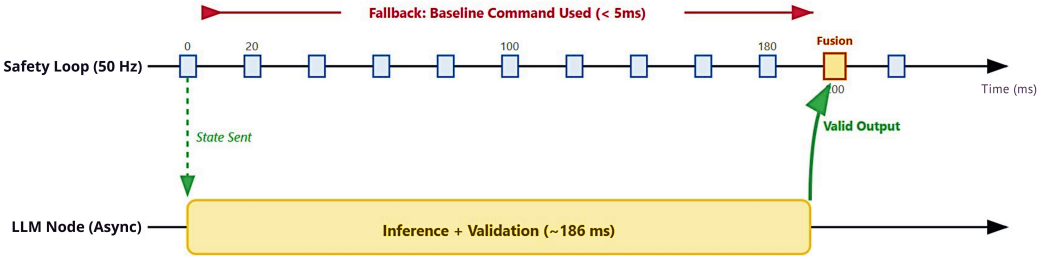


Figure 5. Asynchronous dual-rate architecture. The FSM maintains a 50 Hz safety loop (**top**), while the LLM processes context at a lower rate (**bottom**). Fusion occurs only when a new validated suggestion is available; otherwise, the baseline is applied.

This separation prevents actuation from depending on non-deterministic node scheduling and ensures that all executed commands are attributable to a single deterministic controller.

4.3. LLM Configuration

We prioritize inference speed and memory footprint over raw reasoning capability. Based on the protocol in Table 2, we select DistilGPT-2 (82 M parameters) as an edge-oriented candidate that can produce structured suggestions while remaining feasible on the Jetson Orin. Larger models (e.g., LLaMA-2 7B) exceed practical memory/latency budgets for our on-device setting, whereas smaller models (e.g., TinyGPT-2) often fail to reliably produce parsable structured outputs. In our configuration, DistilGPT-2 generates one suggestion approximately every ∼200 ms (about 5 Hz) under our decoding settings.

Table 2. Comparison of candidate LLMs for robotics integration. DistilGPT-2 was selected for its balance of efficiency and reliability on embedded hardware.

Model	Params	Arch.	Generation	RAM	Speed	Advantages	Disadvantages	Suitability
GPT-2	117 M	Decoder-only	High-quality, coherent long sequences	∼500 MB	Medium	Mature, structured code/commands	Higher memory, not optimal latency	Moderate
TinyGPT-2	28 M	Decoder-only	Limited (basic commands only)	∼120 MB	Very high	Very fast; minimal footprint	Poor expressiveness, brittle syntax	Limited
GPT-2 Small	117 M	Decoder-only	Similar to GPT-2 base	∼500 MB	Medium	Reliable, well-documented	No latency/memory gains vs. GPT-2	Moderate
GPT-Neo-125M	125 M	Decoder-only	Advanced, contextual generation	∼520 MB	Medium	Open-source, modern arch.	Slightly heavier, variable compat.	Moderate

Table 2. *Cont.*

Model	Params	Arch.	Generation	RAM	Speed	Advantages	Disadvantages	Suitability
GPT-Neo 1.3B	1.3 B	Decoder-only	State-of-the-art generation	∼5.2 GB	Low	Very strong reasoning	Massive computation, high latency	Impractical
FLAN-T5 Small	77 M	Enc–Dec	Good instruction following	∼320 MB	High	Precise, efficient	Limited free-form generation	Specialized
LLaMA 2–7B	7 B	Decoder-only	Human-level reasoning (long)	∼14 GB	Very low	Excellent text quality	Prohibitive RAM, slow on edge	Impractical
GPT-NeoX 20B	20 B	Decoder-only	Exceptional, ultra-complex	∼80 GB	Very low	Supreme capability	Prohibitive resources	Impossible
DistilGPT-2	82 M	Decoder-only	Excellent (close to GPT-2)	∼340 MB	High	Fast, compact, reliable	Slight quality loss vs. GPT-2	Chosen (optimal)

Prompt Engineering

To enable robust parsing and validation, we enforce a strict JSON output format. The prompt encodes the robot state and kinematic limits and instructs the model to return only a JSON object with numeric fields `linear_x` and `angular_z`. The template used is:

```
CURRENT STATE:
- Phase: {state['phase']}
- Distance: {state['current_distance']:.3f} m (target: {self.DIST_GOAL} m)
- Offset: {state['offset_horizontal']:.3f}
- Current velocities: linear_x={state['base_v']:.3f}, angular_z={state['base_w']:.3f}
- Deadline: {state['deadline_ms']} ms

VELOCITY LIMITS:
- linear_x: [{-self.V_MAX}, {self.V_MAX}]
- angular_z: [{-self.W_MAX}, {self.W_MAX}]

OBJECTIVE:
- APPROACH: move toward the target while maintaining alignment.

RESPONSE (JSON only; no extra text):
{"linear_x": <float>, "angular_z": <float>}
```

This design provides two benefits. First, it constrains outputs to a machine-parsable schema compatible with automatic validation and fusion. Second, it supports prompt-level behavioral tuning: developers can refine the `APPROACH` strategy (e.g., "move slower within 0.5 m of the target") by editing natural-language instructions rather than modifying the FSM control logic.

5. Experimental Setup

This section describes the experimental methodology used to evaluate the proposed framework in terms of navigation performance, real-time behavior, and developer-oriented usability aspects. The protocol is designed to be reproducible in both simulation and real-world settings and includes an ablation study to isolate the contribution of LLM-based semantic guidance from pure signal smoothing.

5.1. Test Environments

Experiments were conducted in two environments to assess sim-to-real consistency:

- **Simulation (Gazebo):** A TurtleBot3 Burger model was equipped with a simulated Intel RealSense D435i. Simulation enables controlled and repeatable trials under identical initial conditions.
- **Real robot (TurtleBot3 Burger):** The same ROS 2 stack was deployed without code changes on a physical TurtleBot3 Burger with a RealSense D435i and an NVIDIA Jetson

AGX Orin. This setting captures real-world effects such as sensor noise, illumination changes, wheel slip, and ground friction.

Figure 6 shows the simulation and physical setups.

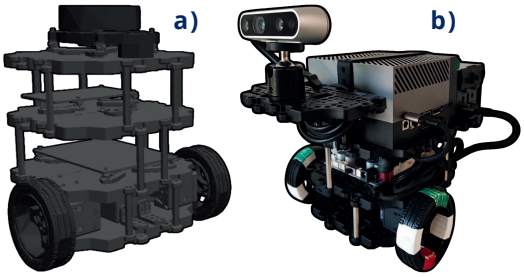


Figure 6. (**a**) TurtleBot3 in Gazebo; (**b**) physical TurtleBot3 Burger setup.

5.2. Navigation Scenario

A 5 × 5 m Maze Arena (Figure 7) was used in both Gazebo and the physical setup. The layout and placement of traffic signs (left, right, stop) were replicated as closely as possible across domains to ensure comparability. Each trial starts from a fixed initial pose and ends when the robot reaches the action threshold d_{act} and completes the corresponding maneuver (or when a timeout criterion is triggered; see Section 4).

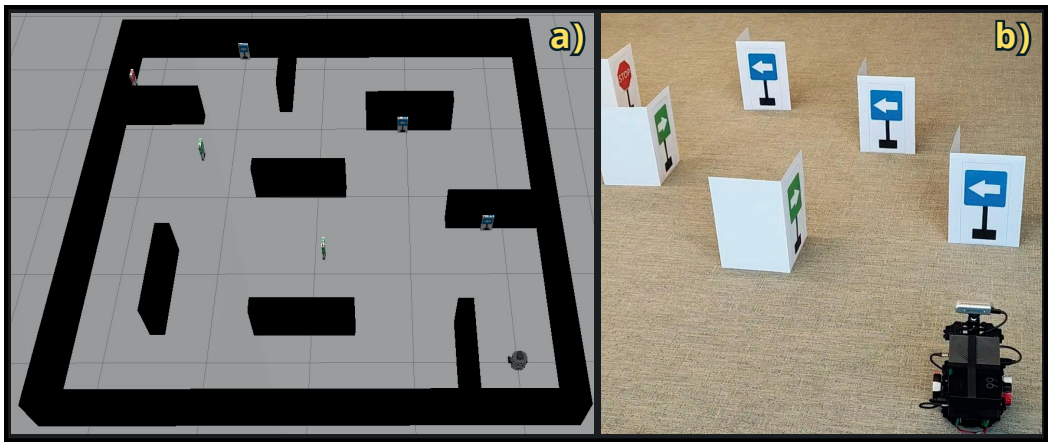


Figure 7. (**a**) Gazebo arena; (**b**) real-world arena replication.

In both simulation and real-robot experiments, each sign has a fixed, pre-measured pose in the arena coordinate frame. In the simulation, the target point $(x_{\text{target}}, y_{\text{target}})$ is obtained directly from the Gazebo world model. In the real setup, sign poses are measured once in the arena frame, and the robot pose is estimated in the same frame via the localization stack used in our TurtleBot3 deployment (logged at 50 Hz). This ensures that e_{pos} compares positions expressed in a consistent coordinate frame across domains.

5.3. Experimental Conditions (Ablation Design)

To assess whether improvements arise from semantic guidance rather than smoothing alone, we compare three controllers:

1. FSM-only (Baseline): The deterministic Finite-State Machine (FSM) without language-based refinement.
2. FSM + smoothing (Control): A purely signal-processing baseline that applies an Exponential Moving Average (EMA) to the FSM command to mimic smoothing effects without LLM input. Specifically, we filter the baseline command as

$$\mathbf{v}_t^{\mathrm{EMA}} = (1-\beta)\,\mathbf{v}_{t-1}^{\mathrm{EMA}} + \beta\,\mathbf{v}_t^{\mathrm{base}}, \qquad \beta \in (0,1], \tag{6}$$

with $\beta = 0.5$ in our experiments. For the smoothing-only baseline, we set the EMA parameter to $\beta = 0.5$ to match the fusion weight used in our proposed controller ($\alpha = 0.5$). This yields a fair ablation; both methods apply comparable smoothing strength, but only the proposed method receives semantic guidance from the language model. In general, a smaller β reduces smoothing and approaches the raw FSM behavior, while a larger β increases smoothing but may introduce lag and degrade responsiveness near transitions.

3. FSM + LLM (Proposed): The full framework in which the FSM produces $\mathbf{v}_{\mathrm{base}}$ and the asynchronous LLM suggestions are validated and fused as described in Equation (5).

An end-to-end "LLM-only" controller was not included because it cannot meet the real-time and safety requirements of the platform under our on-device inference constraints (in particular, inference latency and the lack of deterministic fallback).

For each environment (simulation and real robot), we executed $N = 30$ trials per condition (total $3 \times 2 \times 30 = 180$ trials). During each trial, all relevant topics were logged at 50 Hz.

5.4. Evaluation Metrics

We evaluate the following Key Performance Indicator (KPIs). All metrics are computed per trial and then aggregated across trials.

- Positioning accuracy. The final position error is defined as

$$e_{\mathrm{pos}} = \left\|\mathbf{p}_{\mathrm{final}} - \mathbf{p}_{\mathrm{target}}\right\|_2, \tag{7}$$

where $\mathbf{p}_{\mathrm{final}} = [x_{\mathrm{final}}, y_{\mathrm{final}}]^\top$ is the robot position at the stop event and $\mathbf{p}_{\mathrm{target}}$ is the desired stop location.

- Trajectory efficiency. The trajectory efficiency is

$$\eta = \frac{L_{\mathrm{opt}}}{L_{\mathrm{real}}}, \qquad 0 < \eta \leq 1, \tag{8}$$

where L_{real} is the traveled path length (arc-length) obtained from the logged robot pose at 50 Hz, and L_{opt} L_{opt} is computed offline from the known Maze Arena layout as the shortest collision-free path length between the start pose and the target sign pose (using the same map constraints for both simulation and real trials).

- Control-loop latency. Let t_k^{tick} denote the start time of the control tick k and t_k^{pub} the time at which the final velocity command is published. The control-loop latency is

$$\ell_k = t_k^{\mathrm{pub}} - t_k^{\mathrm{tick}}, \tag{9}$$

and we report summary statistics of $\{\ell_k\}$ over all ticks in a trial.

- Language-model inference latency. Let t_i^{stamp} be the timestamp embedded in the input state for suggestion i, and t_i^{ready} the time at which the suggestion becomes available after generation and parsing. The inference latency is

$$\ell_i^{\mathrm{llm}} = t_i^{\mathrm{ready}} - t_i^{\mathrm{stamp}}. \tag{10}$$

- Integration rates. We report the suggestion-level acceptance rate,

$$r_{\text{acc}} = \frac{N_{\text{accepted}}}{N_{\text{total_suggestions}}}, \tag{11}$$

and tick-level utilization rate (the fraction of control ticks in which a validated suggestion is actually applied),

$$r_{\text{use}} = \frac{N_{\text{ticks_using_llm}}}{N_{\text{total_ticks}}}. \tag{12}$$

For data collection and reproducibility, during each trial, we recorded:

- FSM states, transitions, detections, and baseline velocity outputs;
- LLM prompts, raw outputs, validation outcomes, and fused commands;
- Final executed velocities and timestamps for latency analysis.

Post-processing scripts compute trial-level metrics and export the results to CSV. To support reproducibility, all datasets and scripts are available on request.

6. Results and Discussion

This section reports the outcomes of the Maze Arena evaluation conducted in both Gazebo and on the physical TurtleBot3. We compare the deterministic baseline FSM against the proposed framework (FSM+LLM). Unless otherwise stated, results are reported as a pooled summary across simulation and real-robot trials ($N = 60$ per condition) to increase statistical power for inferential testing.

6.1. Final Positioning Accuracy

We report the terminal positioning error e_{pos} as defined in Equation (13) (Euclidean distance between the final robot pose at the stopping event and the trial-specific target point). Figure 8 summarizes e_{pos} at the stopping event. For each trial, we record the robot's final planar position $(x_{\text{final}}, y_{\text{final}})$ at the instant the FSM transitions into the terminal action state (e.g., `ACT_STOP`) and compare it to the target point $(x_{\text{target}}, y_{\text{target}})$ defined by the Maze Arena layout (fixed sign pose; see Section 5).

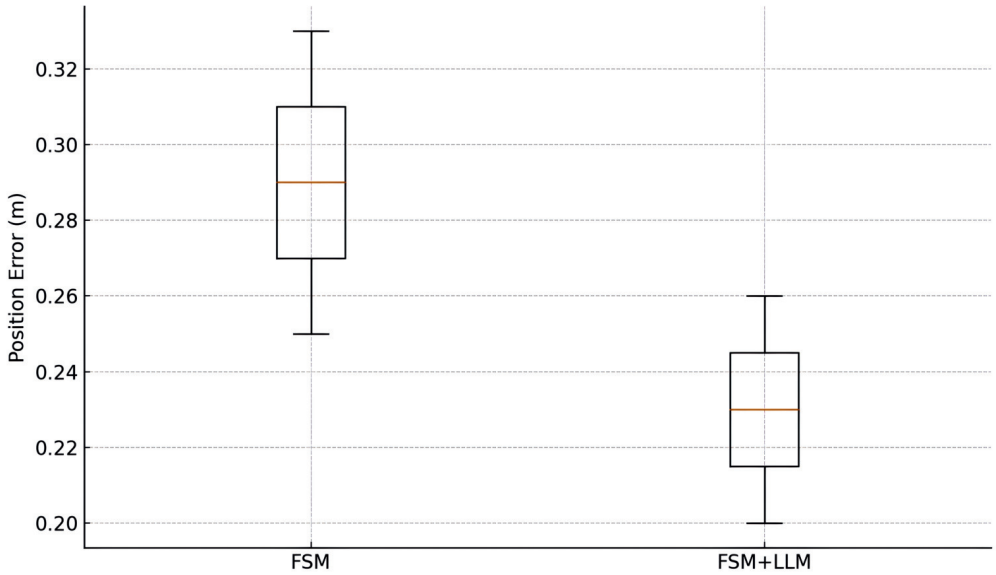


Figure 8. Distribution of final position error e_{pos} at the stopping event (pooled across simulation and real trials; $N = 60$ per condition). Lower values indicate more accurate stopping.

$$e_{\text{pos}} = \sqrt{(x_{\text{final}} - x_{\text{target}})^2 + (y_{\text{final}} - y_{\text{target}})^2}. \quad (13)$$

Aggregated over $N = 60$ pooled trials per condition, the FSM baseline yields $e_{\text{pos}} = 0.246 \pm 0.078$ m, whereas FSM+LLM reduces the error to 0.159 ± 0.069 m, corresponding to an approximate 35% reduction.

These results indicate that validated LLM suggestions improve approach-phase velocity modulation, yielding more consistent terminal alignment and stopping behavior. Reduced terminal error is particularly valuable for repeatable interaction with landmarks and for downstream tasks that require reliable spatial alignment.

6.2. Trajectory Efficiency

We report trajectory efficiency η as defined in Equation (8), and Figure 9 compares η across conditions. In both simulation and real-robot experiments, each sign has a fixed, pre-measured pose in the arena coordinate frame. In simulation, the target point $(x_{\text{target}}, y_{\text{target}})$ is obtained directly from the Gazebo world model. In the real setup, sign poses are measured once in the arena frame and the robot pose is estimated in the same frame via the localization stack used in our TurtleBot3 deployment (logged at 50 Hz). This ensures that e_{pos} compares positions expressed in a consistent coordinate frame across domains.

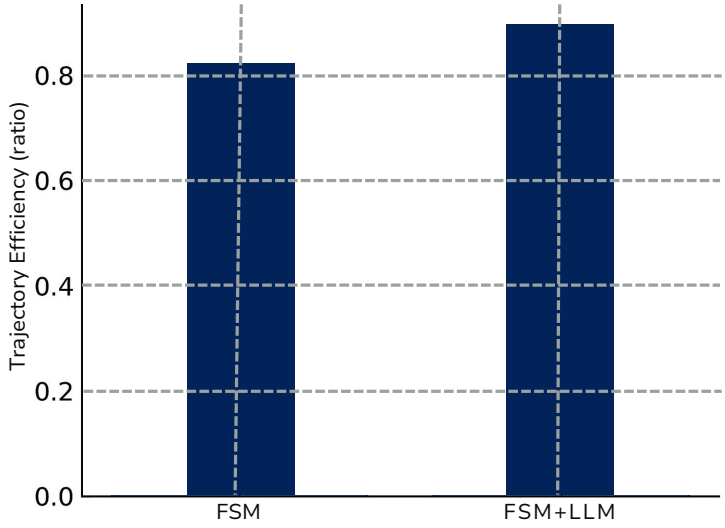


Figure 9. Trajectory efficiency η (pooled across simulation and real trials; $N = 60$ per condition). Higher is better.

Pooled across trials, the FSM-only baseline achieves $\eta = 0.821 \pm 0.043$, while FSM+LLM reaches $\eta = 0.901 \pm 0.038$. This improvement is consistent with fewer detours and reduced oscillations during approach and alignment phases. Higher efficiency generally implies shorter travel time and lower energy consumption, which are relevant for embedded deployments.

The fusion mechanism in Equation (5) introduces a smoothing effect because the FSM updates at 50 Hz, while LLM suggestions arrive at a lower rate (approximately every $\sim$200 ms). Consequently, $\mathbf{v}_{\text{llm}}$ is held constant across multiple FSM ticks, which can reduce abrupt command changes. However, the ablation in Section 6.4 indicates that smoothing alone does not account for the full improvement: the LLM-guided controller outperforms an EMA-based smoothing control that applies comparable low-pass behavior without language-derived context.

6.3. Real-Time Behavior and LLM Integration

As defined in Equations (9) and (10), we report the 50 Hz control-loop latency ℓ_k and the LLM inference latency ℓ_i^{llm}. Figure 10 reports the control-loop command publication latency for the deterministic 50 Hz loop. Because suggestions are applied asynchronously under a sample-and-hold policy, the LLM inference latency (approximately 186 ms) does not affect the 50 Hz control-loop timing. The FSM continues to publish at every 20 ms tick; LLM outputs, when they arrive, update the held suggestion for subsequent ticks as long as they remain within the TTL.

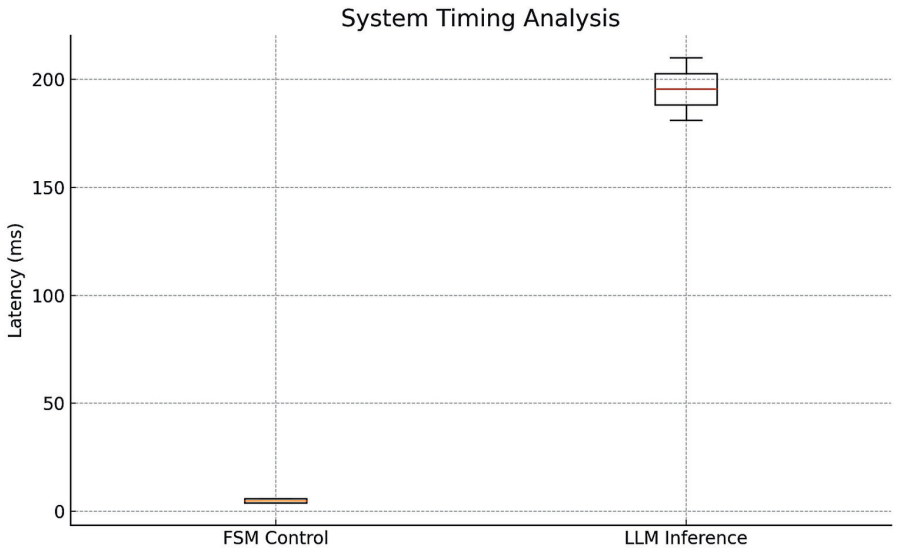


Figure 10. Distribution of control-loop command publication latency (50 Hz). Values are pooled across simulation and real trials.

The additional computation for parsing, validation, and fusion introduces only a small overhead relative to the FSM tick budget (approximately ∼0.2 ms in our measurements), and does not change the deterministic scheduling of the baseline controller. To contextualize where delays arise, we distinguish (i) perception-side latency (camera capture and detector inference), (ii) ROS 2 transport and executor jitter (queuing/scheduling), (iii) LLM inference time, and (iv) validation/fusion overhead. In our measurements, the reported control-loop latency ℓ_k isolates the deterministic actuation path (tick-to-publication), whereas ℓ_i^{llm} captures the end-to-end time from stamped context to a ready-to-consume suggestion.

6.3.1. Acceptance and Rejection of LLM Suggestions

Figure 11 summarizes the fate of LLM outputs. Here we report suggestion-level statistics (one output per LLM inference) rather than per-tick control-cycle fractions. On average, 81% of LLM outputs pass validation and are fused into the control stream; the remaining outputs are rejected due to missed freshness/deadline constraints (12%), kinematic violations (5%), or schema/parse failures (2%).

While Figure 11 reports the frequency of validation outcomes, the categories would differ substantially in terms of potential safety impact if a validator were absent. Late/stale suggestions (12%) primarily pose a performance risk; applying a command based on outdated perception can induce overshoot or oscillatory corrections, especially near the terminal stopping region. Kinematic violations (5%) are potentially safety-critical because unbounded v_x or ω_z may exceed platform limits and cause collisions or loss of stability. Schema/parse failures (2%) are fail-safe in our implementation because malformed outputs cannot be mapped into an actuation command and therefore deterministically trigger fallback. Finally, we observed occasional semantic inconsistencies (rare, included in the

kinematic/stale rejections when detected), such as suggesting acceleration while distance decreases; these are not necessarily kinematically unsafe but can degrade approach smoothness. Table 3 summarizes severity, risk without validation, and the mitigation enforced by the proposed architecture.

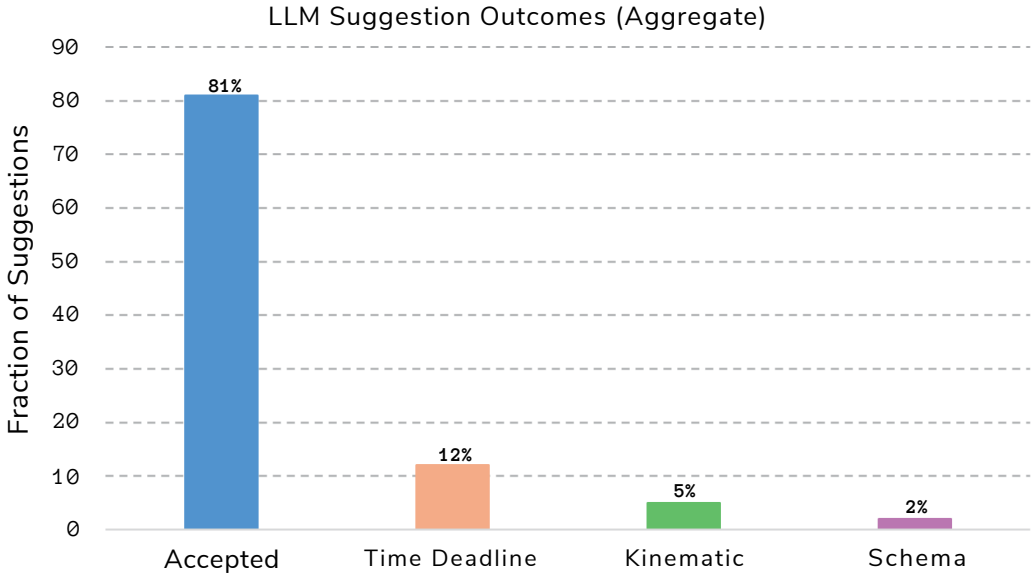


Figure 11. LLM output validation outcomes (pooled across simulation and real trials). Accepted: 81%; rejected due to missed freshness/deadline: 12%; kinematic bounds: 5%; schema/parse failure: 2%.

Table 3. LLM output failure modes: frequency, potential risk without a validator, and mitigation in the proposed architecture.

Failure Mode	Freq.	Severity	Risk Without Validation	Current Mitigation/Future Strengthening
Late/stale (missed TTL or deadline)	12%	Medium	Outdated action conditioned on old perception; can increase overshoot or induce oscillations near the stop region.	TTL freshness gating + deterministic fallback; future: adaptive TTL based on latency percentiles and distance-to-goal.
Kinematic out-of-bounds ($\|v_x\|, \|\omega_z\|$)	5%	High	Commands may exceed platform limits, increasing collision risk or destabilizing turns.	Projection to admissible set + bounds check + fallback; future: tighter bounds near obstacles and rate-limit enforcement.
Schema/parse failure (malformed JSON)	2%	Low	Uninterpretable command; could become unsafe if mapped incorrectly.	Strict JSON parsing rejects; fail-safe fallback; future: constrained decoding/grammar-based JSON.
Semantic inconsistency (context-unsafe but bounded)	(rare)	Medium	Bounded but undesirable actions (e.g., accelerating while approaching) degrade smoothness and efficiency.	Indirectly mitigated via TTL, fusion, and bounds; future: invariants (monotonic slowdown vs. distance), confidence gating, perception-consistency checks.

6.3.2. Velocity Profiles

Figure 12 compares the baseline linear velocity, raw LLM outputs, and fused velocities over a representative trajectory. The FSM baseline exhibits a staircase pattern driven by discrete setpoints (0.06, 0.03, and 0.00 m/s), whereas raw LLM outputs show higher variance and occasional infeasible values. Fusion with $\alpha = 0.5$ reduces dispersion while incorporating context-dependent corrections.

Table 4 summarizes descriptive statistics for v_x. While accepted LLM outputs have a higher spread than the baseline, the fused command reduces variance and remains bounded by the admissible limits.

Figures 13 and 14 illustrate representative trajectory overlays. The fused trajectories show smoother re-centering near junctions and smaller terminal deviation relative to the baseline, consistent with the quantitative improvements in e_{pos} and η.

Overall, these findings reinforce that the LLM component is not suitable as a standalone controller due to occasional infeasible outputs, but it can provide beneficial refinements when combined with strict validation and deterministic fallback.

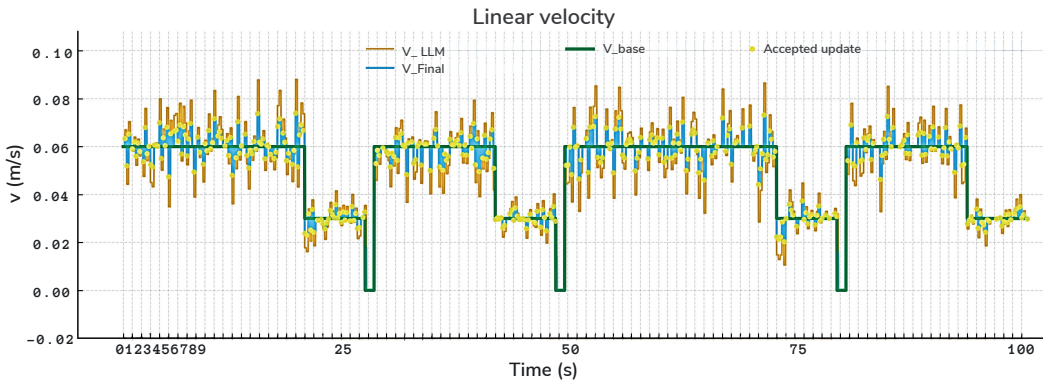


Figure 12. Linear velocity profiles: baseline, raw LLM outputs, and fused command over a representative trajectory.

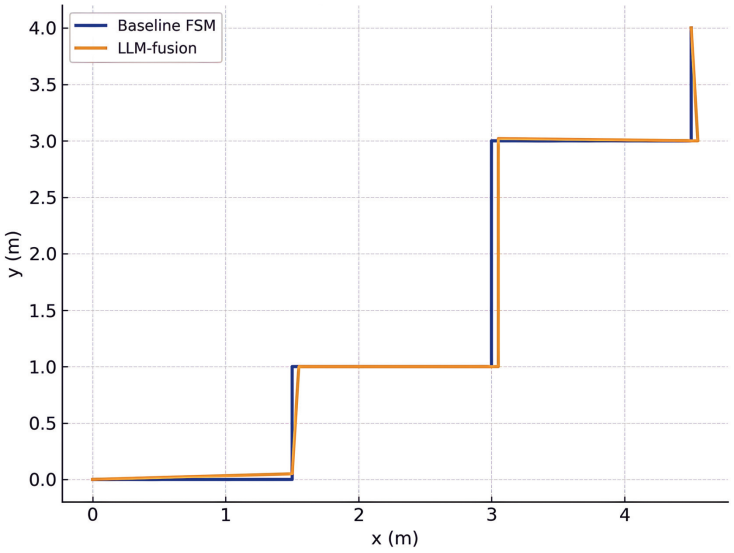


Figure 13. Representative trajectories in simulation (Maze Arena).

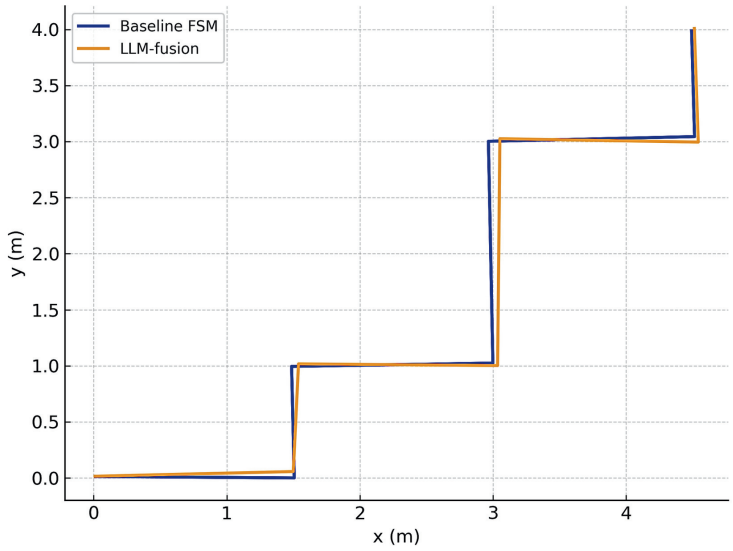


Figure 14. Representative trajectories on the real TurtleBot3 (Maze Arena).

Table 4. Descriptive statistics of linear velocity v_x. The FSM baseline uses discrete setpoints; raw LLM outputs introduce variance; and fusion reduces dispersion while preserving bounds.

Signal	Condition	Range [min, max]	Mean	Std. Dev.
v_x (m/s)	Baseline (0.06)	[0.06, 0.06]	0.060	≈0.000
v_x (m/s)	Baseline (0.03)	[0.03, 0.03]	0.030	≈0.000
v_x (m/s)	Baseline (stop)	[0.00, 0.00]	0.000	≈0.000
v_x (m/s)	LLM (accepted)	[0.00, 0.095]	0.047	0.021
v_x (m/s)	Final fusion	[0.00, 0.075]	0.044	0.019

6.4. Ablation Study: Semantic Guidance vs. Signal Smoothing

To test whether improvements are explained solely by low-pass smoothing, we compare FSM+LLM against a "blind" smoothing control (FSM+EMA) that applies Equation (6) to the baseline command with $\beta = 0.5$, without any LLM input.

Table 5 summarizes the pooled results. The smoothing control reduces positioning error relative to the raw FSM (from 0.246 m to 0.212 m), consistent with damping oscillations. However, FSM+LLM achieves substantially better performance (0.159 m error and $\eta = 0.901$), indicating that language-conditioned, context-dependent refinements contribute beyond smoothing alone.

Table 5. Ablation study results (pooled across simulation and real trials). The smoothing control improves stability over the raw FSM but does not match the LLM-augmented controller.

Metric	FSM (Baseline)	FSM + EMA (Control)	FSM + LLM (Ours)
Logic source	Hard-coded rules	Rules + smoothing	Rules + language-conditioned refinement
Final position error (m)	0.246 ± 0.07	0.212 ± 0.05	0.159 ± 0.06
Trajectory efficiency	0.821	0.855	0.901
Guidance type	Reactive	Reactive (damped)	Context-dependent (anticipatory)

6.5. Statistical Analysis of Outcomes

Table 6 reports the pooled outcomes with 95% confidence intervals (CIs). Continuous measures are reported as mean ± SD and 95% t-CIs. For inferential testing, we compare

conditions using Welch's two-sample t-tests (robust to unequal variances) and report Hedges' g with small-sample correction.

The results support the descriptive trends. The final position error is reduced by approximately 0.087 m on average ($p = 1.86 \times 10^{-9}$, $g = -1.18$), and the trajectory efficiency increases substantially ($p = 2.12 \times 10^{-18}$, $g = 1.91$), both reflecting large practical effects. Control-loop latency does not differ significantly ($p = 0.495$), indicating that LLM integration does not compromise deterministic responsiveness.

Table 6. Statistical summary pooled across simulation and real-robot trials ($N = 60$ per condition). Continuous outcomes report mean ± SD and 95% t-CIs. Two-sided tests.

Measure	Baseline (FSM)	LLM-Augmented	Test / Effect Size
Final position error (m)	0.246 ± 0.078 [0.226, 0.266]	0.159 ± 0.069 [0.141, 0.177]	Welch $t = 6.52$, $p = 1.86 \times 10^{-9}$; Hedges' $g = -1.18$
Trajectory efficiency	0.821 ± 0.046 [0.809, 0.833]	0.901 ± 0.037 [0.892, 0.911]	Welch $t = -10.51$, $p = 2.12 \times 10^{-18}$; Hedges' $g = 1.91$
Control-loop latency (ms)	3.37 ± 0.52 [3.24, 3.51]	3.44 ± 0.62 [3.28, 3.60]	Welch $t = -0.68$, $p = 0.495$ (n.s.)

For Welch's t-tests, we report the test statistic using the difference (defined as Baseline minus LLM-augmented), so negative t indicates higher values under the LLM-augmented condition.

7. Limitations and Generalizability

This study validates the proposed architecture in a controlled maze scenario with three traffic-sign classes and mostly static conditions. While the deterministic FSM baseline and fallback preserve safety through construction (the control loop never blocks on language inference and all suggestions are bounded and validated), several limitations affect generalizability and performance.

Scenario complexity. The current setup does not include moving obstacles or dense dynamic interactions. In highly dynamic environments, performance gains may diminish when the environment changes faster than the language-model update rate.

Perception uncertainty. The framework assumes reasonably reliable sign detection and range estimation. Under incomplete or contradictory sensor data (false positives/negatives, intermittent detections, or biased depth), the validator preserves feasibility, but semantic refinements may become less consistent. Coupling the approach with uncertainty-aware gating and explicit confidence fields is a necessary extension.

Action space and semantic scalability. Scaling beyond a small sign vocabulary requires an explicit command ontology and state-conditioned prompt templates to preserve auditability as the semantic space grows. Stronger structured interfaces (e.g., schema-constrained decoding or tool-like actions) can further reduce format violations.

Platform dependence. The selection of freshness TTL and overall feasibility depend on measured on-device latency distributions and concurrent workloads (e.g., perception). Porting to different hardware requires re-profiling and re-tuning of timing and bounds.

8. Conclusions and Future Work

This paper presents a plug-and-play architecture that augments a deterministic Finite-State Machine with asynchronous language-model suggestions, subject to strict validation and freshness constraints. The safety-critical controller maintains a 50 Hz loop and never blocks on language inference; instead, candidate suggestions are accepted only when they satisfy a strict schema, remain fresh, and respect explicit kinematic bounds, and are then fused with the baseline command.

Across Gazebo simulation and real-TurtleBot3 experiments, the proposed controller improved navigation outcomes without increasing control-loop latency. The final position error decreased from 0.246 m to 0.159 m and the trajectory efficiency increased from 0.821 to 0.901. Approximately 80% of generated suggestions passed validation and were integrated, while late, malformed, or out-of-bounds outputs were safely rejected and replaced by the deterministic fallback. The ablation against a smoothing-only baseline further indicates that improvements are not explained solely by low-pass filtering, but by opportunistic semantic modulation during approach phases.

The method's main practical implication is an interpretable integration pattern for edge robotics: language-based guidance can be incorporated without compromising deterministic actuation, and behavior tuning can be performed at the prompt level rather than by rewriting control code. At the same time, broader generalization depends on scenario complexity, perception reliability, and platform-dependent timing constraints, as discussed in Section 7.

Future work will extend evaluation to more diverse and dynamic environments (moving obstacles, occlusions, lighting changes), expand semantic command sets through an explicit action ontology, and study sensitivity to fusion weight, prompt paraphrasing, and freshness policies. Additionally, we will explore constrained decoding and lightweight instruction-tuned models to further improve structured-output reliability under embedded constraints. To isolate the effect of the proposed asynchronous architecture (validation, fusion, and deterministic fallback), we kept the prompt template fixed and therefore do not claim robustness to prompt paraphrasing. Minor rewordings may change the quality of generated suggestions and, consequently, schema validity and validator acceptance rates, as well as downstream navigation metrics. Importantly, safety does not rely on prompt wording, as all suggestions are gated by strict JSON parsing, kinematic admissibility checks, and temporal freshness/TTL constraints; thus, prompt changes may affect performance but cannot bypass the safety envelope or fallback behavior. A systematic prompt-robustness evaluation is left for future work, which will aim to generate semantically equivalent paraphrases of the prompt (minor syntactic rewordings preserving the same control intent) and report variability across paraphrases in (i) schema-valid rate, (ii) validator acceptance/rejection breakdown, and (iii) navigation outcomes (e.g., e_{pos} and η) with confidence intervals.

Author Contributions: Conceptualization, S.R.-O. and E.Z.; methodology, S.R.-O. and E.Z.; software, S.R.-O.; validation, S.R.-O. and E.Z.; formal analysis, S.R.-O.; investigation, S.R.-O.; data curation, S.R.-O.; writing—original draft, S.R.-O.; writing—review and editing, S.R.-O., E.Z., V.M., I.Y. and M.S.; visualization, S.R.-O.; supervision, E.Z.; project administration, E.Z.; resources, M.S.; review, V.M. and I.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the SENDOA project (ID: KK-2025/00102) and the DBaskIN KK-2025/00012 project, financed by the Council of the Basque Country.

Data Availability Statement: The data and code used in this study are available on request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LLM	Large Language Model
SLM	Small Language Model
FSM	Finite-State Machine
ROS	Robot Operating System

RGB-D	Red–Green–Blue plus Depth
EMA	Exponential Moving Average
VLA	Vision–Language–Action
VLM	Vision–Language Model
CNN	Convolutional Neural Network
YOLO	You Only Look Once
ONNX	Open Neural Network Exchange
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MSE	Mean Squared Error
IAE	Integral of Absolute Error
MPC	Model Predictive Control
PID	Proportional–Integral–Derivative

References

1. Escobar-Naranjo, J.; Caiza, G.; Ayala, P.; Jordan, E.; Garcia, C.A.; Garcia, M.V. Autonomous Navigation of Robots: Optimization with DQN. *Appl. Sci.* **2023**, *13*, 7202. [CrossRef]
2. Tabarsi, B.; Reichert, H.; Limke, A.; Kuttal, S.; Barnes, T. LLMs' Reshaping of People, Processes, Products, and Society in Software Development: A Comprehensive Exploration with Early Adopters. *arXiv* **2025**, arXiv:2503.05012. [CrossRef]
3. Chu, K.; Zhao, X.; Weber, C.; Wermter, S. LLM+MAP: Bimanual Robot Task Planning using Large Language Models and Planning Domain Definition Language. *arXiv* **2025**, arXiv:2503.17309. [CrossRef]
4. Zhu, Y.; Wan Hasan, W.Z.; Harun Ramli, H.R.; Norsahperi, N.M.H.; Mohd Kassim, M.S.; Yao, Y. Deep Reinforcement Learning of Mobile Robot Navigation in Dynamic Environment: A Review. *Sensors* **2025**, *25*, 3394. [CrossRef] [PubMed]
5. Zitkovich, B.; Yu, T.; Xu, S.; Xu, P.; Xiao, T.; Xia, F.; Wu, J.; Wohlhart, P.; Welker, S.; Wahid, A.; et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In Proceedings of the Conference on Robot Learning (CoRL), Atlanta, GA, USA, 6–9 November 2023; pp. 2165–2183. Available online: https://proceedings.mlr.press/v229/zitkovich23a/zitkovich23a.pdf (accessed on 15 January 2026).
6. Shentu, Y.; Wu, P.; Rajeswaran, A.; Abbeel, P. From LLMs to Actions: Latent Codes as Bridges in Hierarchical Robot Control. *arXiv* **2024**, arXiv:2405.04798. [CrossRef]
7. Tran, T.T.H. PID Application in Mobile Robot Control. *Int. J. Adv. Eng. Manag. (IJAEM)* **2022**, *4*, 2181–2185. Available online: https://ijaem.net/issue_dcp/PID%20application%20in%20mobile%20robot%20control.pdf (accessed on 15 January 2026).
8. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An Open-Source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009.
9. Coleman, D.; Sucan, I.A.; Chitta, S.; Correll, N. Reducing the Barrier to Entry of Complex Robotic Software: A MoveIt! Case Study. *arXiv* **2014**, arXiv:1404.3785. [CrossRef]
10. Shridhar, M.; Manuelli, L.; Fox, D. CLIPort: What and Where Pathways for Robotic Manipulation. In Proceedings of the Conference on Robot Learning, Auckland, New Zealand, 14–18 December 2022.
11. Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv* **2022**, arXiv:2204.01691. [CrossRef]
12. Brohan, A.; Brown, N.; Carbajal, J.; Chebotar, Y.; Dabis, J.; Finn, C.; Gopalakrishnan, K.; Hausman, K.; Herzog, A.; Hsu, J.; Ibarz, J.; et al. RT-1: Robotics Transformer for Real-World Control at Scale. *arXiv* **2023**, arXiv:2212.06817. [CrossRef]
13. Padalkar, A.; Pooley, A.; Mandlekar, A.; Jain, A.; Tung, A.; Bewley, A.; Herzog, A.; Irpan, A.; Khazatsky, A.; Rai, A.; et al. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. *arXiv* **2023**, arXiv:2310.08864.
14. Driess, D.; Xia, F.; Sajjadi, M.S.M.; Lynch, C.; Chowdhery, A.; Ichter, B.; Wahid, A.; Tompson, J.; Vuong, Q.; Yu, T.; et al. PaLM-E: An Embodied Multimodal Language Model. *arXiv* **2023**, arXiv:2303.03378.
15. Black, K.; Brown, N.; Driess, D.; Esmail, A.; Equi, M.; Finn, C.; Fusai, N.; Groom, L.; Hausman, K.; Ichter, B.; et al. π_0: A Vision–Language–Action Flow Model for General Robot Control. *arXiv* **2024**, arXiv:2410.24164.
16. Intelligence, P.; Black, K.; Brown, N.; Darpinian, J.; Dhabalia, K.; Driess, D.; Esmail, A.; Equi, M.; Finn, C.; Fusai, N.; et al. $\pi_{0.5}$: A Vision–Language–Action Model with Open-World Generalization. *arXiv* **2025**, arXiv:2504.16054.
17. Goetting, D.; Singh, H.G.; Loquercio, A. End-to-End Navigation with Vision Language Models: Transforming Spatial Reasoning into Question-Answering. *arXiv* **2024**, arXiv:2411.05755. [CrossRef]
18. Shah, D.; Osinski, B.; Levine, S. LM-Nav: Robotic Navigation with Large Pretrained Models of Language, Vision, and Action. In Proceedings of the 6th Conference on Robot Learning, Auckland, New Zealand, 14–18 December 2022.
19. Shah, D.; Levine, S. ViNT: A Foundation Model for Visual Navigation. *arXiv* **2023**, arXiv:2306.14896. [CrossRef]

20. Huang, C.; Mees, O.; Zeng, A.; Burgard, W. Visual Language Maps for Robot Navigation. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023.
21. Kerr, J.; Kim, C.M.; Goldberg, K.; Kanazawa, A.; Tancik, M. LERF: Language Embedded Radiance Fields. *arXiv* **2023**, arXiv:2303.07324. [CrossRef]
22. Zhou, G.; Hong, Y.; Wu, Q. NavGPT: Explicit Reasoning in Vision-and-Language Navigation with Large Language Models. *arXiv*, **2023**, arXiv:2305.16986. [CrossRef]
23. Ouyang, Y.; Li, J.; Li, Y.; Li, Z.; Yu, C.; Sreenath, K.; Wu, Y. Long-Horizon Locomotion and Manipulation on a Quadrupedal Robot with Large Language Models. *arXiv*, **2024**, arXiv:2404.05291. [CrossRef]
24. Wang, W.; Xie, J.; Hu, C.; Zou, H.; Fan, J.; Tong, W.; Wen, Y.; Wu, S.; Deng, H.; Li, Z.; et al. DriveMLM: Aligning Multi-Modal Large Language Models with Behavioral Planning States for Autonomous Driving. *arXiv* **2023**, arXiv:2312.09245.
25. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv* **2022**, arXiv:2201.11903. [CrossRef]
26. Zhou, D.; Schärli, N.; Hou, L.; Wei, J.; Scales, N.; Wang, X.; Schuurmans, D.; Cui, C.; Bousquet, O.; Le, Q.; et al. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *arXiv* **2022**, arXiv:2205.10625. [CrossRef]
27. Robey, A.; Ravichandran, Z.; Kumar, V.; Hassani, H.; Pappas, G.J. Jailbreaking LLM-Controlled Robots. 2024. Available online: https://robopair.org/files/research/robopair.pdf (accessed on 15 January 2026).
28. Cui, E.; Wang, W.; Li, Z.; Xie, J.; Zou, H.; Deng, H.; Luo, G.; Lu, L.; Zhu, X.; Dai, J. Highlighting the Safety Concerns of Deploying LLMs/VLMs in Robotics. *arXiv* **2024**, arXiv:2402.10340.
29. Ye, Y.; Nie, Z.; Liu, X.; Xie, F.; Li, Z.; Li, P. ROS2 Real-Time Performance Optimization and Evaluation. *Chin. J. Mech. Eng.* **2023**, *36*, 144. [CrossRef]
30. Park, J.; Delgado, R.; Choi, B.W. Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems: An Empirical Study. *IEEE Access* **2020**, *8*, 152840–152851. [CrossRef]
31. Puck, L.; Keller, P.; Schnell, T.; Plasberg, C.; Tanev, A.; Heppner, G.; Roennau, A.; Dillmann, R. Performance Evaluation of Real-Time ROS2 Robotic Control in a Time-Synchronized Distributed Network. In Proceedings of the 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), Lyon, France, 23–27 August 2021; pp. 1664–1671.
32. Lee, S.; Kang, W.; Bertogna, M.; Chwa, H.S.; Lee, J. Timing Guarantees for Inference of AI Models in Embedded Systems. *Real-Time Syst.* **2025**, *61*, 259–267. [CrossRef]
33. Waseem, M.; Bhatta, K.; Li, C.; Chang, Q. Pretrained LLMs as Real-Time Controllers for Robot Operated Serial Production Line. *arXiv* **2025**, arXiv:2503.03889. [CrossRef]
34. Mower, C.E.; Wan, Y.; Yu, H.; Grosnit, A.; Gonzalez-Bill, on J.; Zimmer, M.; Wang, J.; Zhang, X.; Zhao, Y.; Zhai, A.; et al. ROS-LLM: A ROS Framework for Embodied AI with Task Feedback and Structured Reasoning. *arXiv* **2024**. [CrossRef]
35. Bode, J.; Pätzold, B.; Memmesheimer, R.; Behnke, S. A Comparison of Prompt Engineering Techniques for Task Planning and Execution in Service Robotics. *arXiv* **2024**, arXiv:2410.22997. [CrossRef]
36. Kim, Y.; Kim, D.; Choi, J.; Park, J.; Oh, N.; Park, D. A Survey on Integration of Large Language Models with Intelligent Robots. *Intell. Serv. Robot.* **2024**, *17*, 1091–1107. [CrossRef]
37. Wang, J.; Shi, E.; Hu, H.; Ma, C.; Liu, Y.; Wang, X.; Yao, Y.; Liu, X.; Ge, B.; Zhang, S. Large Language Models for Robotics: Opportunities, Challenges, and Perspectives. *J. Autom. Intell.* **2025**, *4*, 52–64. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.