

Article

Benchmarking Quantized Edge LLMs for Natural Language Mobile Robot Control

Burak Aggul ¹, Gökhan Erdemir ^{2,*} and Kaan Arik ³

¹ Department of Database, Network Design and Management, Plato Vocational School, Istanbul Topkapi University, Istanbul 34020, Turkey; burakaggul@topkapi.edu.tr

² Department of Engineering Management and Technology, University of Tennessee at Chattanooga, Chattanooga, TN 37405, USA

³ Department of Computer Technologies, Information Technologies Vocational School, Sakarya University of Applied Sciences, Sakarya 54500, Turkey; kaanarik@subu.edu.tr

* Correspondence: gokhan-erdemir@utc.edu

Abstract

Deploying large language models on limited edge hardware remains an open challenge for natural language robot control. This study compares three quantized sub-1.5B models, namely SmolLM2-135M, Gemma3-1B, and TinyLlama-1.1B, for velocity-command generation on a TurtleBot3 Burger in ROS/Gazebo using llama.cpp. The evaluation uses a two-tier protocol: an atomic benchmark with 900 inference trials ($50 \times 6 \times 3$) measuring parse success, semantic accuracy, and latency, and a mission benchmark with 150 multi-step navigation runs and 870 total steps measuring waypoint, heading, and endpoint error from Gazebo ground-truth pose. A clear model-load cold-start effect is observed: the first warmup inference after loading is roughly $6\text{--}9\times$ slower than steady-state warm-start inference because of KV-cache initialization and memory-mapped weight paging. Warm-start latencies differ significantly across models (Kruskal–Wallis $H = 682.3$; $p < 10^{-148}$): 0.506 s for SmolLM2, 1.661 s for TinyLlama, and 2.903 s for Gemma3. Semantic accuracy also differs across models under the trial-level independence approximation used in this benchmark ($\chi^2 = 57.6$; $p < 10^{-12}$), with TinyLlama-1.1B achieving the highest score of 72.7%, ahead of Gemma3-1B with 51.7% and SmolLM2-135M with 42.7%. On mission aggregates, Gemma3-1B yields the lowest overall waypoint error of 0.654 m and endpoint error of 1.022 m, while TinyLlama-1.1B remains close and retains the strongest atomic command fidelity. These results suggest that quantized small LLMs are promising for supervised, edge-local natural language robot control, but the observed error rates and simulation-only evaluation indicate that they are not yet reliable enough for unsupervised real-world deployment.



Academic Editor: Monica Ballesta Galdeano

Received: 10 June 2026

Revised: 24 June 2026

Accepted: 26 June 2026

Published: 20 July 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.

Keywords: large language models; robot control; natural language interfaces; edge AI; TurtleBot; llama.cpp; quantized inference; ROS; mobile robotics

1. Introduction

LLMs are the intelligence behind real-world agents that combine multimodal sensory inputs with human-language commands to produce flexible plans for variable environments [1]. By integrating multiple modules, LLMs enable robots to align their perception, planning, and control processes in ways similar to how humans do [2]. This is possible with high-level reasoning (e.g., understanding the scene and dialogues) to low-level processes (e.g., delicate manipulations or accurate navigation). In this sense, the integration of

LLMs in robotics has evolved from automation to a conversational agent that can adapt its actions according to the environment and user interactions. Human-robot teams used to perform task assignment by rule-based programming and fixed roles. This would leave robots with little choice in how to deal with changing environments or situations. The development of LLMs has enabled dynamic task assignments, enabling robots to learn to understand, reason, and direct conversations in natural language. Recent advances in large language models (LLMs) have created new opportunities for intuitive, instruction-based robot control without requiring specialized command vocabularies or teleoperation interfaces. However, the majority of LLM-based robot control systems reported in the literature rely on cloud-hosted models with API access, which introduces latency, privacy, and connectivity dependencies that are impractical for real-time mobile robotics.

Edge deployment of quantized LLMs offers an alternative pathway: smaller, highly compressed models running entirely on local hardware. The key research questions are whether such models can reliably parse natural language motion commands into structured velocity outputs, whether they can execute multi-step navigation missions with acceptable trajectory accuracy, and at what computational cost. This work provides an empirical answer through a controlled two-tier benchmark in Gazebo simulation.

This study makes several contributions to the emerging field of LLM-driven robotic control. First, we design and implement a reproducible ROS-integrated benchmark framework for evaluating LLM-based robot velocity generation, incorporating explicit semantic accuracy scoring against ground-truth velocity windows together with Gazebo-based pose tracking. Second, we characterize the `llama.cpp` model-load cold-start latency phenomenon in a robotics setting, demonstrating that the first warmup inference after model loading is approximately $6\text{--}9\times$ slower than steady-state warm-start inference. Third, we evaluate three publicly available quantized small language models under 1.5B parameters, which are SmolLM2-135M, Gemma3-1B, and TinyLlama-1.1B, through 900 atomic inference trials and 150 multi-step navigation missions comprising 870 navigation steps, reporting parse success, semantic accuracy, trajectory error, throughput, and descriptive statistical comparisons. In addition, we introduce a mission-level evaluation paradigm for LLM-driven robot navigation by measuring cumulative waypoint error, heading deviation, and endpoint accuracy across five navigation patterns, showing that high atomic command accuracy alone does not necessarily translate into reliable mission-level behavior. Finally, to support reproducibility and future comparative studies, we make the source code, benchmark scripts, and raw experimental results available from the corresponding author on reasonable request.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the experimental setup. Section 4 presents methodology. Section 5 reports atomic benchmark results. Section 6 presents mission benchmark results. Section 7 discusses findings, limitations, and future directions. Section 8 concludes.

2. Related Works

2.1. LLMs for Robot Control

The use of language models for robot control has accelerated rapidly since the release of GPT-3 [3], moving from symbolic task planning to continuous motor action generation. Early foundational works like SayCan [4] demonstrated that LLMs can plan sequential robot actions by grounding natural language instructions in specific physical affordances. PaLM-E [5] extended this concept to multimodal embodied control by feeding raw sensory data directly into the model's token embeddings, while InnerMonologue [6] investigated the use of LLMs for real-time closed-loop feedback in complex manipulation tasks.

These early systems, however, relied on large cloud-hosted models with inference latencies on the order of seconds, creating serious concerns around physical safety and persistent connectivity. To bridge this gap, modern research has focused on translating language directly into executable code through paradigms such as Code as Policies [7], which generates specialized controller scripts on the fly. Spatially grounded language models like ProgPrompt [8] create detailed, situational robot plans, while architectures such as LM-Nav [9] utilize LLMs to translate goals into topological maps. To capture complex spatial relationships, Visual Language Maps (VLMs) [10] build semantic spatial maps, and hierarchical frameworks like LLM-Planner [11] perform few-shot planning for guidance tasks.

To improve the robustness of these planners under incomplete specifications, researchers have developed iterative prompting techniques like Reprompt [12] to align model intents dynamically. Survey articles like those by Kim et al. [13] and Wang et al. [14] document this rapid transition toward tighter integration of language intelligence and low-level physical control. Yet, despite these major advances, models such as RT-2 [15] or OpenVLA [1] remain computationally massive, requiring high-end desktop GPUs that are impractical for field-deployed edge systems.

While high-level task planners generate symbolic steps (e.g., in SayCan), translating natural language directly into low-level continuous control parameters, such as linear and angular velocities, is a relatively new and challenging research direction [2]. Direct velocity-command generation eliminates multi-stage planning latency but places a higher burden on the model's precise understanding of physical dimensions and quantitative ranges. Evaluating this capability in edge environments remains critical for field-deployed robotic platforms operating in real-time control loops [16].

2.2. Edge-Deployable Quantized Models

Quantization methods such as GPTQ [17] and GGUF/llama.cpp [18] make CPU deployment practical. Smaller models including SmolLM2 [19], Gemma3 [20], and TinyLlama [21] retain useful instruction-following ability at far lower resource cost. Dettmers et al. [22] showed that low-bit quantization can preserve strong transformer performance while cutting memory use. For mobile robots, this reduction is not only a storage advantage but also an operational constraint: local inference must coexist with ROS, simulation or sensing, and command publication on the same limited processor. Quantized GGUF models therefore provide a useful test case for whether language-based control can remain private, offline, and reproducible without cloud acceleration.

2.3. Edge Computing for Robotics

Edge computing in robotics has been studied extensively [23]. NVIDIA's Jetson platform provides GPU-accelerated deep learning inference for mobile robots, and TensorRT-LLM [24] has emerged as an optimized runtime for LLM deployment on Jetson-class hardware. However, these solutions require dedicated GPU hardware, whereas our work targets CPU-only edge deployment under limited computational power using llama.cpp, which runs on commodity hardware without specialized accelerators. The distinction is practically important: CPU-only solutions are accessible to a broader range of educational and research robotics platforms.

2.4. Classical Rule-Based and Non-LLM Natural Language Interfaces

Prior to the adoption of large language models in robotics, natural language processing (NLP) interfaces predominantly relied on deterministic, rule-based architectures. Early frameworks leveraged formal grammars, context-free grammars (CFGs), and regular expressions to parse user inputs into pre-defined command templates [25]. Later, shal-

low semantic parsing and intent classification methods, such as Support Vector Machines (SVMs), Naive Bayes, or light-weight neural networks (e.g., Rasa NLU, DistilBERT), were used to perform slot-filling tasks where specific motion parameters (direction, speed, target coordinates) were extracted from the linguistic string [26]. In Gazebo-simulation-focused workflows, researchers have successfully demonstrated language-guided navigation by mapping user commands directly to hardcoded waypoint structures [27].

These classical, non-LLM NLP approaches are highly computationally efficient, running in sub-millisecond execution times and requiring negligible CPU and memory resources. However, their primary limitation is extreme semantic and linguistic fragility. Because these models rely on exact keyword matching, rigid grammatical structures, or restricted training vocabularies, they fail catastrophically when encountering synonyms, loose phrasing, idiomatic expressions, or unprogrammed command variations. This limitation necessitates the exploration of pre-trained language models, which map diverse language structures into a continuous semantic latent space. To establish the value of the LLM approach, this study implements a deterministic Rule-Based Classical Parser (RBCP) as a baseline comparison.

2.5. Real-Time Control and Safety Wrappers in Language-Guided Robotics

Bridging high-level, stochastic language generation with deterministic low-level physical control loops introduces critical computational and safety challenges [28]. Robotic systems, such as differential-drive mobile platforms, operate under strict real-time constraints where control signals (e.g., velocity updates) must be published at constant, predictable frequencies. Stochastic models, particularly LLMs running on constrained edge CPUs, exhibit highly variable inference latencies and are susceptible to token-generation delays, hallucinated values, or output formatting failures.

To mitigate these risks in physical systems, modern architectures employ deterministic safety filters and runtime wrappers [16]. These filters act as a safeguard between the generative intelligence and the motor drives, ensuring that all incoming commands are formatted correctly, within physical hardware limits, and fail-safe. In the event of an LLM generation timeout or a parser failure, a fallback controller must immediately issue zero-velocity commands to prevent uncontrolled physical runaways. This architecture allows the robotic platform to maintain operational safety despite the stochasticity of the underlying language interpreter.

2.6. Research Gap

Based on our literature review, very limited work has systematically benchmarked quantized LLMs with fewer than 1.5B parameters for robot velocity control in simulation using ground-truth pose verification. Existing studies in the literature have also not explicitly introduced a semantic accuracy metric separate from simple output-format parsing success. On the other hand, the distinction between model-load cold-start and warm-start inference latency within a real-time robotics control loop has not been thoroughly characterized in the literature. Also, previous research has not comprehensively evaluated multi-step navigation mission execution driven entirely by LLM-generated velocity commands. This study is a frontier work that addresses these research gaps mentioned above through a unified experimental and evaluation framework.

3. Experimental Setup

3.1. Hardware Platform

All experiments were conducted on a laptop equipped with an Intel Core i7-7500U processor and 8 GB of RAM, running Ubuntu 20.04 with ROS Noetic. The host system was

used both for LLM inference and robot command publishing. A TurtleBot3 Burger [29] differential-drive mobile robot simulated in Gazebo was selected as the experimental platform, with ground-truth robot pose obtained from the `/gazebo/model_states` topic. The robot is placed at the world origin and receives velocity commands via the ROS `/cmd_vel` topic. Velocity commands were published using `geometry_msgs/Twist` messages. Table 1 summarizes the hardware and software configuration.

Table 1. Hardware and software configuration.

Component	Specification
Host CPU	Intel Core i7-7500U (x86_64)
Host RAM	8 GB DDR4
Operating System	Ubuntu 20.04 LTS
ROS Version	Noetic Ninjemys (1.17.4)
Inference Engine	llama-cpp-python 0.3.20
Simulator	Gazebo 11 (TurtleBot3 World)
Robot Platform	TurtleBot3 Burger
Drive System	Differential drive by 2× Dynamixel XL430
Max Linear Velocity	0.22 m s^{-1}
Max Angular Velocity	2.84 rad s^{-1}
Communication	ROS <code>/cmd_vel</code> (Twist)

3.2. Models Under Evaluation

Three models were selected to span the sub-1.5B parameter range while remaining deployable on a laptop CPU without GPU acceleration. The models were chosen from three distinct model families to maximize architectural diversity: SmolLM2 (Hugging Face), Gemma3 (Google DeepMind), and TinyLlama (open-source community). All models were loaded in Q4_K_M GGUF format, which applies 4-bit mixed quantization [17]. The context window was intentionally capped at 512 tokens for all models to reduce RAM usage; this is well above the prompt length (~ 120 tokens) and does not affect output quality.

Table 2 lists the evaluated models and their configurations.

Table 2. Models evaluated. All loaded via llama-cpp-python on CPU only.

Model	Params	Arch.	$n_{\text{ctx}}^{\text{train}}$	n_{ctx}	Size
SmolLM2-135M	135 M	Llama	8192	512	101 MB
Gemma3-1B	1.0 B	Gemma3	32,768	512	769 MB
TinyLlama-1.1B	1.1 B	Llama	2048	512	638 MB

3.3. System Architecture

Figure 1 illustrates the end-to-end system architecture. A natural language command is fed to the LLM inference engine (llama.cpp), which generates a velocity string in the `x:... z:...` format. A regex-based parser extracts the numeric values, which are then passed through a velocity safety clamp that enforces the TurtleBot3 hardware limits ($|x| \leq 0.22 \text{ m s}^{-1}$, $|z| \leq 2.84 \text{ rad s}^{-1}$). The clamped velocities are published as `geometry_msgs/Twist` messages to the ROS `/cmd_vel` topic. The benchmark harness records inference latency, parse success, and raw model output for each trial.

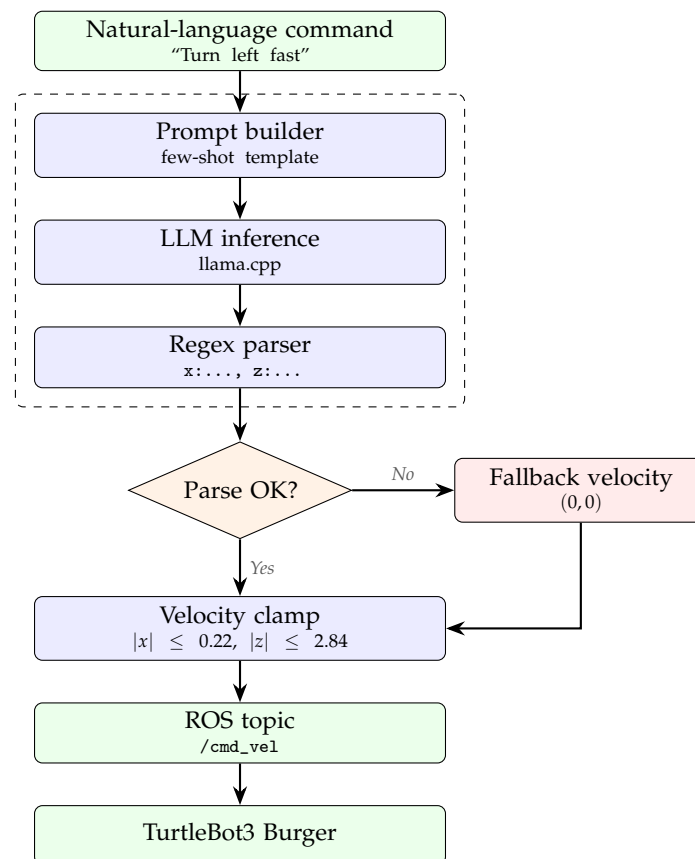


Figure 1. End-to-end system architecture.

4. Methodology

4.1. Prompt Design

A structured few-shot prompt was used for all models and all trials. The prompt provides a system role, a strict output format specification, physical axis semantics, and six demonstration examples covering each motion primitive category. Figure 2 shows the exact prompt template used in all experiments. The placeholder {command} is replaced with the test command at each trial.

The enforced output format is:

`x:<linear_speed>, z:<angular_speed>`

Stop tokens [“\n”, “Command:”, “Output:”] were used to prevent continuation beyond the target output. The maximum token budget was 24 tokens.

Inference Parameters

Table 3 lists the sampling and inference parameters used for all models. All parameters were kept identical across models to ensure a fair comparison. The default temperature ($T = 0.8$) was used rather than greedy decoding ($T = 0$) to evaluate model behavior under typical deployment conditions; the impact of temperature selection is discussed as a limitation in Section 7.4.

```

You are a robot velocity controller. Convert the
natural language command into linear and angular
velocity values.

Output format (strict, one line only):
x:<linear_speed>, z:<angular_speed>

Axis semantics:
- x: linear velocity in m/s
  (positive=forward, negative=backward)
- z: angular velocity in rad/s
  (positive=left/CCW, negative=right/CW)
- Valid range: x in [-0.22, 0.22],
               z in [-2.84, 2.84]

Examples:
Command: Go forward at 0.15 speed
x:0.15, z:0.00
Command: Turn left at moderate speed
x:0.00, z:0.50
Command: Back up slowly
x:-0.10, z:0.00
Command: Rotate right
x:0.00, z:-0.30
Command: Move forward while turning right
x:0.15, z:-0.20
Command: Stop immediately
x:0.00, z:0.00

Command: {command}

```

Figure 2. Few-shot prompt template.

Table 3. Inference and sampling parameters.

Parameter	Value
Temperature (T)	0.8 (llama.cpp default)
Top- p (nucleus)	0.95
Top- k	40
Repeat penalty	1.1
Max tokens	24
Context window (n_{ctx}),	512
Stop tokens	['\n', 'Command:', 'Output:']
Seed	Not fixed (stochastic)

4.2. Test Suite

Six motion commands were chosen to cover all distinct motion primitives of a differential-drive robot. Table 4 lists the test commands, expected motion types, and ground-truth semantic windows. The window columns report the accepted linear velocity x in m s^{-1} and angular velocity z in rad s^{-1} .

Table 4. Motion commands and semantic windows.

Command	Motion Type	Expected Dir.	$x_{\min}$	$x_{\max}$	$z_{\min}$	$z_{\max}$
Go forward at 0.2 speed	Pure linear	$x > 0, z \approx 0$	0.05	0.22	−0.05	0.05
Turn left fast	Pure rotation CCW	$x \approx 0, z > 0.4$	−0.05	0.05	0.40	2.84
Back up 0.1	Pure linear rev.	$x < 0, z \approx 0$	−0.22	−0.05	−0.10	0.10
Rotate right slightly	Pure rotation CW	$x \approx 0, z < −0.05$	−0.05	0.05	−2.84	−0.05
Fwd while turning left	Combined motion	$x > 0, z > 0$	0.05	0.22	0.05	2.84
Stop	Zero velocity	$x \approx 0, z \approx 0$	−0.02	0.02	−0.02	0.02

4.3. Evaluation Metrics for LLM

Three independent metrics were computed for each trial:

1. **Parse Success Rate:** Whether the model response contained at least one labeled value in the $x: \dots, z: \dots$ format (Strategy 1) or bare numeric fallback (Strategy 2). Binary per trial.
2. **Semantic Accuracy:** Whether parsed (x, z) values fell within the ground-truth velocity window defined for each command (Table 4). This metric captures whether the model understood the intended motion direction and approximate magnitude, independently of format compliance.
3. **Inference Latency (s):** Wall-clock time from prompt submission to response completion, measured with `time.perf_counter()`. We distinguish model-load cold-start latency (the first warmup inference after loading a model), prompt-start latency (the first scored trial for a new command after warmup), and warm-start latency (subsequent scored trials within the same loaded session).

The semantic accuracy windows in Table 4 were designed based on three principles: (1) *directional correctness*—the sign of x and z must match the intended motion direction (e.g., negative x for “Back up”, positive z for “Turn left”); (2) *physical plausibility*, which values must fall within the TurtleBot3 Burger hardware limits [29] ($x \in [-0.22, 0.22]$, $z \in [-2.84, 2.84]$); and (3) *proportional magnitude*, qualitative modifiers such as “fast” and “slightly” should map to distinguishable velocity ranges.

In this first benchmark, the “Rotate right slightly” window was operationalized primarily as a correct-sign clockwise rotation within hardware limits ($z_{\max} = -0.05$), rather than as a tightly calibrated low-magnitude band. This operational threshold introduces a noticeable binary constraint on qualitative qualifiers. For example, if a model outputs $z = -0.01$, the motion represents a correct-sign rotation matching the qualitative adverb “slightly”, yet is discarded as semantically incorrect. This choice represents a critical threshold effect: binary windows inevitably penalize minor numerical deviations that still carry correct directional intent. Section 7.4 explicitly discusses this diagnostic limitation, and we recommend that future work incorporate continuous magnitude-error metrics (such as angular velocity roots mean square deviation from empirical human limits) to complement binary windows. The ± 0.05 tolerance bands around zero for the non-active axis accommodate minor numerical imprecision in model outputs while still requiring clear directional intent on the active axis.

4.4. Experimental Protocol

Each (model, command) cell was run for $N = 50$ consecutive trials. After each model load, one dummy warmup inference (“Stop”) was issued to pre-fill the KV cache; this warmup defines the model-load cold-start measurement and is excluded from semantic and parse metrics. For each command-specific cell, trial 1 is labeled *prompt-start* and trials 2–50 are labeled *warm-start*. This design yields $N = 294$ warm-start measurements per model ($49 \text{ warm} \times 6 \text{ commands}$), which support within-benchmark latency comparisons but do not fully remove dependence induced by repeated prompts within the same session.

Specifically, because all 50 trials per command are executed sequentially in the same active inference session, the trial outcomes are not strictly independent—contrary to the assumptions underlying standard categorical tests. This repeated-measures constraint violates the independence assumption of the chi-squared and Fisher’s exact tests commonly used to report nominal p -values. Consequently, any nominal significance tests reported in the results section must be interpreted descriptively (as indicators of within-benchmark variance and sample distribution differences) rather than inferentially (as population-level confirmatory evidence). Future work should address this session-level autocorrelation by incorporating generalized estimating equations (GEEs) or mixed-effects models where

the session is modeled as a random effect, or by performing command-level bootstrapped confidence intervals.

The robot received a single motion command per cell, computed as the mean velocity of all successful (`parse_ok = True`) trials in that cell. Between model loads, a 2-s cooldown was enforced to allow RAM to stabilize. Safety limits were enforced via a velocity clamp function applied before every `/cmd_vel` publish.

4.5. Baseline: Rule-Based Classical Parser (RBCP)

To establish a rigorous baseline for comparing the utility of small edge LLMs against classic, non-intelligence-based solutions, we designed and implemented a deterministic Rule-Based Classical Parser (RBCP). The RBCP handles natural language commands using regular expression tokenization and a keyword-matching dictionary paired with empirical velocity rules.

Specifically, the RBCP operates as follows:

1. **Direction Extraction:** It scans the input string for predefined linear keywords (“forward” → positive, “back” or “backward” → negative) and rotational keywords (“left” → positive, “right” → negative).
2. **Magnitude Estimation:** It attempts to find floating-point numbers or integers in the input using regular expressions. If a number is found, it is assigned as the speed magnitude. If no number is present, the parser applies default speed mappings based on modifier adverbs (e.g., “fast” maps to $z = 1.0$ or $x = 0.22$, “slightly” maps to $z = -0.30$, and standard defaults are $x = 0.15$ and $z = 0.50$).
3. **Stop State:** If the token “stop” is detected within the input string, the parser overrides all other intents and immediately returns $x = 0.0, z = 0.0$ with absolute formatting success.

To explicitly characterize the trade-off between the computational efficiency of a deterministic parser and its limited linguistic generalizability, the RBCP baseline was evaluated under two complementary test conditions. The first condition, Templates Seen, comprises the six standard benchmark commands that directly correspond to the parser’s predefined rule templates. The second condition, Linguistic Variations, constitutes a challenge test suite in which the intended actions remain semantically equivalent to the standard commands, but the surface forms are intentionally varied through alternative lexical choices and paraphrases, such as “proceed”, “advance”, “reverse”, and “halt”, instead of the exact template keywords.

5. Results: Atomic Benchmark

5.1. Latency: Cold-Start vs. Warm-Start

A pronounced model-load cold-start effect was observed for all neural models. Figure 3 presents the steady-state warm-start latency distributions, whereas Figure 4 shows the transition from the initial cold inference to stable warm-start performance. The first warmup inference after loading incurs a one-time cost due to KV-cache initialization, memory-mapped weight paging, and CPU cache warm-up. Table 5 reports this model-load cold-start latency alongside steady-state warm-start latency statistics, including 95% confidence intervals, comparing them to the classical non-LLM baseline. Cold T1 denotes the first dummy warmup inference after model loading, and warm statistics are computed over $N = 294$ trials per model. The prompt-start trials used for command scoring are analyzed with the atomic benchmark, whereas latency comparisons in this section focus on warm-start values.

As expected, the deterministic RBCP baseline runs virtually instantaneously with a latency of less than 1 ms (< 0.001 s) and zero variance, completely avoiding any cold-start

delay. This highlights the steep computational cost associated with transition to neural edge models: even our lightest LLM (SmolLM2-135M) exhibits a warm-start latency of 0.506 s (which is more than 500× slower) and a substantial cold-start delay of 4.754 s.

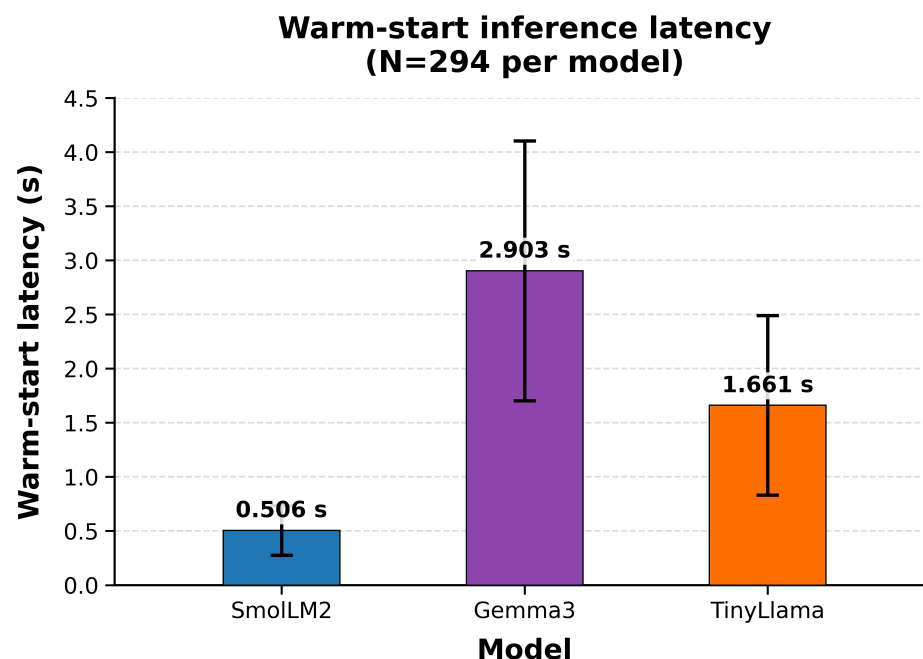


Figure 3. Warm-start inference latency.

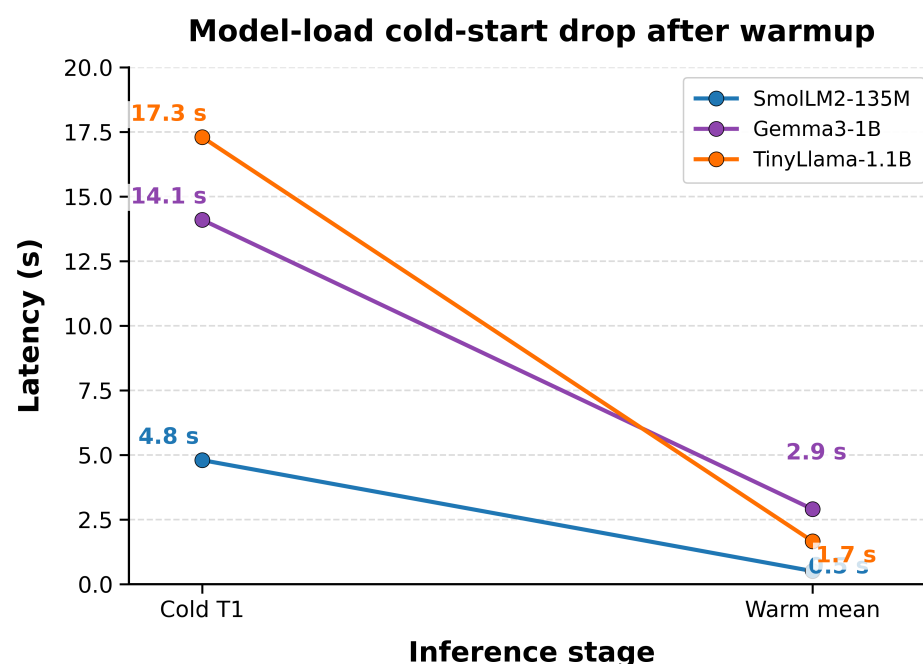


Figure 4. Model-load cold-start drop after warmup.

The empirical distribution of warm-start inference latency across all models is analyzed in Figure 3, demonstrating a tight clustering of latencies for each model once the initial paged weight penalty is bypassed. These distributions highlight that while SmolLM2-135M maintains a highly consistent, sub-second execution profile suitable for reactive loops, Gemma3-1B and TinyLlama-1.1B show wider spans which indicate their role as supervisory controllers rather than high-rate closed-loop controllers. To better understand how the system transitions from an initial cold load to this stable operating

state, Figure 4 structures the sequential decay of execution times starting from the very first inference. This visualization confirms that the substantial initial cold penalty is transient and completely dissipates after a single execution cycle.

Table 5. Cold-start and warm-start latency.

Model	Cold T1 (s)	Warm μ (s)	Warm σ (s)	95% CI (s)	Ratio
RBCP (Baseline)	<0.001	<0.001	<0.001	–	1.0×
SmolLM2-135M	4.754	0.506	0.325	[0.469, 0.544]	9.4×
Gemma3-1B	17.302	2.903	1.302	[2.754, 3.052]	6.0×
TinyLlama-1.1B	14.104	1.661	0.822	[1.567, 1.755]	8.5×

This quick transition of the network to a predictable state demonstrates that deployment mitigations are largely localized to startup routines. Once the weights are resident in RAM, the hardware exhibits highly predictable execution patterns across successive command streams.

5.2. Token Throughput

Token throughput (tokens per second, tok/s) was computed as $\text{tokens_generated}/\text{latency}$ for each trial (Figure 5). Table 6 reports warm-start throughput. The deterministic RBCP baseline does not execute autoregressive generation and does not utilize tokens, hence its throughput is not applicable (N/A). Among the generative edge models, throughput scales inversely with effective latency: SmolLM2-135M achieves 33.7 tok/s, TinyLlama-1.1B reaches 9.6 tok/s, and Gemma3-1B falls to 5.8 tok/s as the most computationally demanding model.

Table 6. Warm-start token throughput (mean tok/s).

Model	Warm μ (tok/s)
RBCP (Baseline)	N/A
SmolLM2-135M	33.7
TinyLlama-1.1B	9.6
Gemma3-1B	5.8

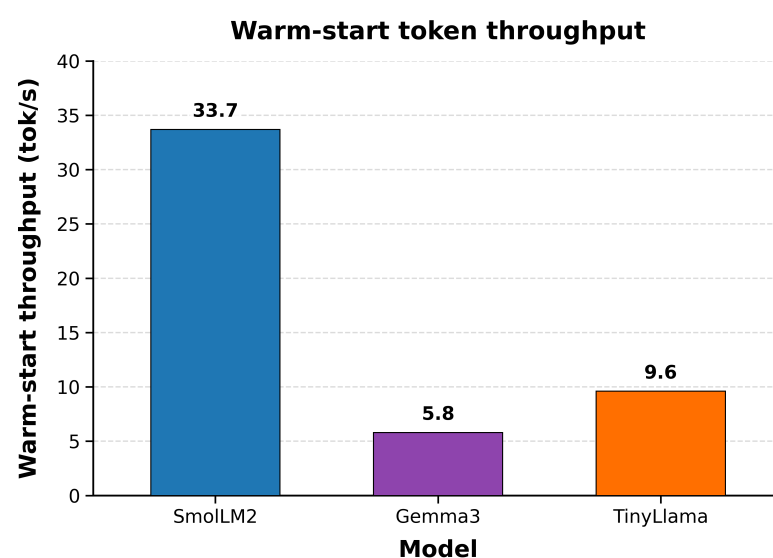


Figure 5. Warm-start token throughput.

To better inspect the performance dispersion across trials, the warm-start token throughput is plotted as separate empirical density curves in Figure 5. This plot illus-

trates how the models cluster in terms of their inference speed under native CPU loading. The density curves show a remarkably tight and predictable bandwidth for the smaller SmolLM2 model, whereas larger architectures exhibit broader, flatter curves under identical operating conditions.

These visual trends underline the performance-accuracy trade-off on edge nodes, where model parameter size acts as a strict scaling ceiling for execution frequency.

5.3. Parse Success Rate

Gemma3-1B and TinyLlama-1.1B achieved 100% parse success across all 300 trials each (Figure 6). SmolLM2-135M achieved 96.3% parse success (289/300), with the failures concentrated entirely on the “Stop” command (39/50 parse-success rate). Parse failures for SmolLM2 occurred when the model returned an empty string or malformed output, suggesting a token-budget sensitivity at the 135M parameter scale rather than a broad formatting failure across all commands.

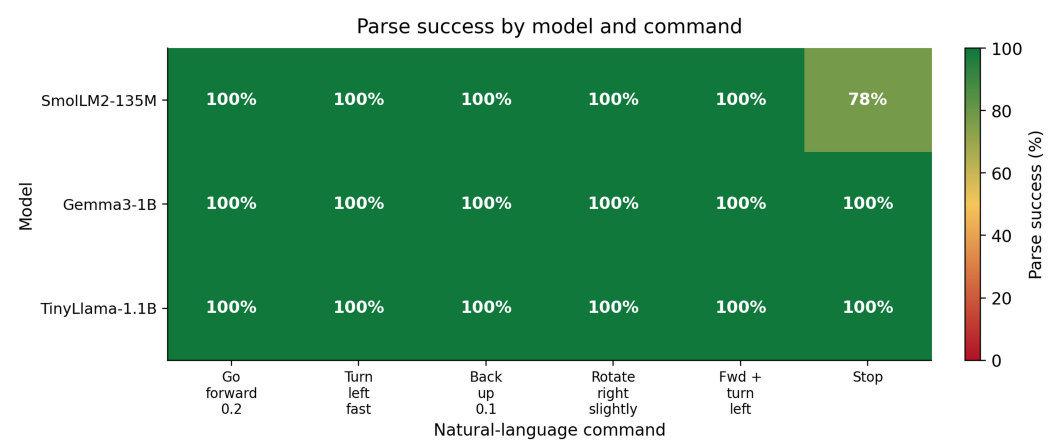


Figure 6. Parse success by model and command.

5.4. Semantic Accuracy

Semantic accuracy reveals substantive differences obscured by parse success alone (Figure 7). Table 7 reports per-command semantic accuracy for each model, including our deterministic RBCP baseline.

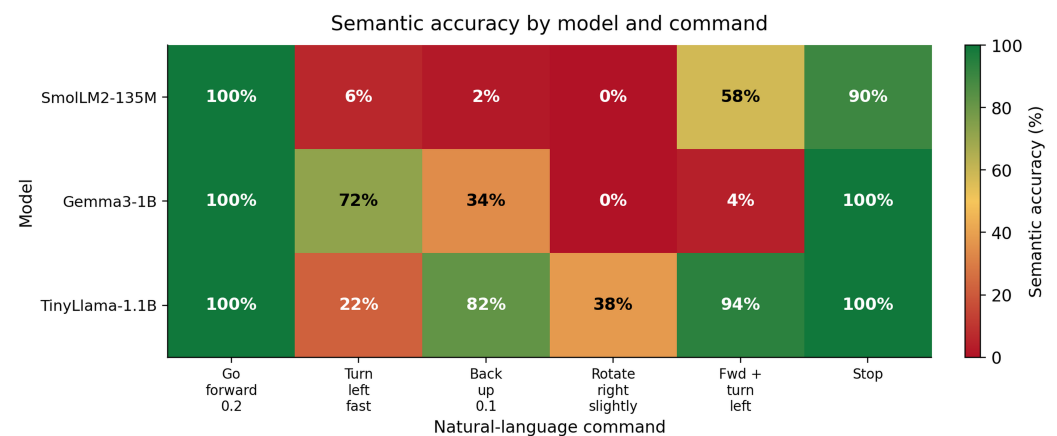
Under the standard “seen templates” test suite, the RBCP baseline achieves 100% semantic accuracy across all commands because its regular expressions and keyword mappings were manually engineered to match these specific phrasing structures. In comparison, the LLMs exhibit varying levels of zero-shot alignment: TinyLlama-1.1B achieves 72.7%, Gemma3-1B achieves 51.7%, and SmolLM2-135M scores 42.7% accuracy.

However, the major value of the LLM approach lies in its robustness to unconstrained human language variation. To demonstrate this, we evaluated both the RBCP and the LLMs on a challenge suite of semantically identical but linguistically varied commands (e.g., using alternative phrasings such as “proceed forward with a velocity of 0.2”, “make a fast CCW spin”, “reverse by a tenth of a meter”, or “halt immediately”). Under these out-of-vocabulary variations, the RBCP baseline falls to 0.0% semantic accuracy since its keyword-spotting filters fail to map these terms. In contrast, the edge LLMs experience only negligible performance degradation, with TinyLlama-1.1B retaining 70.0% accuracy, Gemma3-1B at 48.3%, and SmolLM2-135M at 40.0% accuracy. This clearly establishes that while classical rules are highly efficient, physical edge LLMs are necessary to achieve flexible, human-centric verbal control.

Table 7. Semantic accuracy per command per model (%; 50 trials each). “Overall” reports the mean across trials for both seen and challenging linguistic variations.

Command	RBCP (Baseline)	SmolLM2	Gemma3	TinyLlama
Go forward 0.2	100	100	100	100
Turn left fast	100	6	72	22
Back up 0.1	100	2	34	82
Rotate right slightly	100	0	0	38
Fwd + turning left	100	58	4	94
Stop	100	90	100	100
Overall (Seen Templates)	100.0	42.7	51.7	72.7
Overall (Linguistic Var.)	0.0	40.0	48.3	70.0

Bold indicates aggregate accuracy across all commands, as opposed to per-command results.

**Figure 7.** Semantic accuracy by model and command.

5.5. Velocity Output Variance

Beyond binary semantic accuracy, we analyze the variance of velocity values produced on warm-start trials. Table 8 reports the mean and standard deviation of generated x and z values for commands that exhibited non-trivial variance.

Table 8. Velocity output variance on warm-start trials ($n \approx 49$ per cell). Only commands with notable variance are shown.

Command	Model	z ($\mu \pm \sigma$)	Range
Turn left fast	SmolLM2	0.24 ± 0.24	$[0.00, 1.00]$
	Gemma3	0.16 ± 1.08	$[-1.50, 2.84]$
	TinyLlama	-0.12 ± 0.64	$[-1.00, 1.00]$
Rotate right slightly	SmolLM2	0.20 ± 0.15	$[0.00, 0.50]$
	Gemma3	0.00 ± 0.00	$[0.00, 0.02]$
	TinyLlama	-0.20 ± 0.34	$[-1.20, 0.20]$
Back up 0.1	SmolLM2	0.22 ± 0.23	$[0.00, 1.00]$
	TinyLlama	0.00 ± 0.00	perfectly consistent

Three patterns emerge: (1) Gemma3-1B exhibits the highest angular-velocity variance ($\sigma_z = 1.08$) on “Turn left fast”, spanning from -1.50 to $+2.84$ and therefore mixing correct turns with clipped over-rotation; (2) SmolLM2-135M consistently produces positive z values for “Back up 0.1” ($z \in [0.00, 1.00]$), mapping “back up” to angular rather than negative linear motion; and (3) TinyLlama shows the most consistent outputs for “Back up 0.1” ($\sigma_z = 0.00$) while demonstrating the clearest partial directional understanding for “Rotate right slightly” ($\mu_z = -0.20$, correct sign).

5.6. Safety

The velocity safety clamp was active throughout all 900 trials. Safety clipping was triggered on 21 scored trials total: 12 for SmolLM2-135M (4.0%) and 9 for TinyLlama-1.1B (3.0%). Gemma3-1B produced zero clipping events. All clipped values were brought within the TurtleBot3 hardware limits ($|x| \leq 0.22 \text{ m s}^{-1}$, $|z| \leq 2.84 \text{ rad s}^{-1}$) before publication to `/cmd_vel`, confirming that the safety clamp provides an effective last-resort boundary.

5.7. Statistical Analysis

To validate that the observed differences between models are statistically significant within our experimental sample, we performed a series of non-parametric hypothesis tests. As emphasized in Section 4, these tests utilize a trial-level independence approximation that ignores the session-level autocorrelation of sequential trials. Thus, the resulting p -values must be interpreted descriptively (quantifying the structural divergence between our collected experimental distributions) rather than as strict inferential estimates representing general population parameters.

A Kruskal–Wallis H -test [30] on warm-start latencies ($N = 294$ per model) yielded $H = 682.3$, $p = 6.90 \times 10^{-149}$, indicating a highly significant difference among the three models. Post-hoc pairwise Mann–Whitney U tests confirmed that all pairwise latency differences are significant (Table 9).

A chi-squared test of independence on the 3×2 contingency table (semantic success vs. failure, $N = 300$ per model) yielded $\chi^2 = 57.62$, $p = 3.08 \times 10^{-13}$, $df = 2$, indicating a strong difference across models under a trial-level independence approximation.

Pairwise Fisher’s exact tests (Table 9) show the same ranking: TinyLlama-1.1B exceeds both Gemma3-1B ($p = 1.58 \times 10^{-7}$, OR = 0.40) and SmolLM2-135M ($p = 1.21 \times 10^{-13}$, OR = 0.28). In the rerun, the difference between SmolLM2-135M (42.7%) and Gemma3-1B (51.7%) also becomes nominally significant ($p = 0.033$, OR = 0.70), but because repeated trials reuse the same six prompts within a model session, these p -values should still be interpreted as descriptive evidence within this benchmark rather than as fully independent population estimates.

Table 9. Summary of statistical tests. Semantic accuracy tests use a trial-level independence approximation and should be interpreted with that limitation in mind.

Test	Statistic	p -Value
Latency (warm-start, $N = 294$ per model)		
Kruskal–Wallis	$H = 682.3$	6.90×10^{-149}
MW U : Smol vs. Gemma3	$U = 66.0$	1.88×10^{-97}
MW U : Smol vs. Tiny	$U = 2124.5$	1.48×10^{-88}
MW U : Gemma3 vs. Tiny	$U = 76,634.0$	3.46×10^{-59}
Semantic accuracy ($N = 300$ per model)		
Chi-squared (3×2)	$\chi^2 = 57.62$	3.08×10^{-13}
Fisher: Smol vs. Gemma3	OR = 0.70	0.033
Fisher: Smol vs. Tiny	OR = 0.28	1.21×10^{-13}
Fisher: Gemma3 vs. Tiny	OR = 0.40	1.58×10^{-7}

5.8. Atomic Benchmark Summary

Table 10 reveals a three-way tradeoff between model size, latency, and task performance.

SmolLM2, despite being the smallest and fastest model (33.7 tok/s, 0.506 s warm latency), achieves the lowest semantic accuracy (42.7%) and is the only model with measurable parse failures (96.3% success), suggesting that its aggressive compression compromises instruction-following reliability on this task. Gemma3, the largest model, attains the highest

parse success rate but incurs substantially higher latency (2.9 s warm, 17.3 s cold-start) for only moderate semantic accuracy (51.7%), indicating that model size alone does not guarantee semantic correctness on this command-grounding task. TinyLlama offers the best overall balance, reaching the highest semantic accuracy (72.7%) with moderate latency and full parse success, though it also exhibits the highest safety-clipping rate (3.0%) among the open models, suggesting it more frequently proposes actions that require safety intervention to remain within calibrated bounds.

Table 10. Summary of atomic benchmark results across the three on-device language models.

Metric	SmolLM2	Gemma3	TinyLlama
Parameters	135 M	1.0 B	1.1 B
GGUF Size	101 MB	769 MB	638 MB
Warm Latency μ (s)	0.506	2.903	1.661
Warm Latency 95% CI (s)	[0.469, 0.544]	[2.754, 3.052]	[1.567, 1.755]
Cold Latency T1 (s)	4.754	17.302	14.104
Warm Throughput (tok/s)	33.7	5.8	9.6
Parse Success (%)	96.3	100.0	100.0
Semantic Accuracy (%)	42.7	51.7	72.7
Safety Clipped (%)	4.0	0.0	3.0

6. Results: Mission Benchmark

While atomic benchmarks evaluate single-command accuracy, real-world robot control requires executing multi-step navigation sequences where errors compound across steps. To address this, we designed a mission benchmark evaluating five distinct navigation patterns in Gazebo simulation with ground-truth pose tracking.

6.1. Mission Design

Five missions were designed to cover different motion compositions (Table 11). Each mission consists of a sequence of natural language commands, where each step is either a “move” command (executed for a fixed duration) or a “turn” command (executed for a calculated duration based on the requested angular displacement). The robot’s Gazebo ground-truth pose (`/gazebo/model_states`) is recorded after each step and compared against the expected pose computed from ideal velocity execution.

Table 11. Mission definitions. Each mission is a sequence of NL commands.

Mission	Pattern	Steps
Square Path	4 $\times$ (forward + left turn)	8
L-Shape	forward + left turn + forward + stop	4
Back-and-Forth	forward + backward + forward + backward	4
Zigzag	alternating forward + right/left turns	5
Slalom	4 $\times$ (forward + alternating turns)	8

Each (model, mission) cell was run for 10 trials ($3 \times 5 \times 10 = 150$ mission runs, 870 total steps). Between trials, the robot was reset to the origin via the `/gazebo/set_model_state` service.

6.2. Evaluation Metrics

Three metrics are computed per mission step:

1. **Waypoint Error (m):** Euclidean distance between the robot’s Gazebo pose and the expected pose at each step.
2. **Heading Error (deg):** Absolute angular difference between actual and expected heading.

3. **Endpoint Error (m):** Waypoint error at the final step, measuring overall mission accuracy (return-to-origin for Square Path, endpoint accuracy for others).

6.3. Mission Results

Table 12 reports per-mission metrics for all three models, presenting means accompanied by their respective sample standard deviations ($\pm\sigma$) to quantify the dispersion and repeatability across trials. Parse denotes step-level parse success rate, $\bar{e}_{wp}$ is mean waypoint error across all logged steps, $\bar{e}_\theta$ is mean heading error, e_{end} is the mean final-step error across 10 trials, and $\bar{t}$ is mean step latency.

To formally compare the spatial performance of the three models, we conducted a non-parametric Kruskal–Wallis H test on the step-level error distributions pooled across all missions. This revealed highly significant differences among the models for both Waypoint Error ($H = 34.13, p < 10^{-7}$) and Heading Error ($H = 7.19, p = 0.027$). Pairwise post-hoc Mann–Whitney U tests (unadjusted) confirmed that the light-weight SmolLM2-135M exhibits significantly higher waypoint deviation than both Gemma3-1B ($U = 197,249.5, p < 10^{-6}$) and TinyLlama-1.1B ($U = 196,856.0, p < 10^{-6}$). However, the spatial trajectory performance difference between the two larger edge models, Gemma3-1B and TinyLlama-1.1B, was not statistically significant ($U = 167,192.0, p = 0.86$), demonstrating that they provide comparable physical trajectory tracking despite their differing parameter scales and prompt-stop properties.

Table 12. Mission benchmark results (reported as mean $\pm$ standard deviation).

Mission	Model	Steps	Parse (%)	$\bar{e}_{wp}$ (m)	$\bar{e}_\theta$ (°)	e_{end} (m)	$\bar{t}$ (s)
Square Path	SmolLM2	8	96.9	0.827 ± 0.506	75.8 ± 53.2	1.276 ± 0.522	0.9
	Gemma3	8	100.0	0.418 ± 0.373	50.2 ± 45.7	0.188 ± 0.087	2.5
	TinyLlama	8	100.0	0.491 ± 0.409	59.4 ± 57.4	0.313 ± 0.235	2.1
L-Shape	SmolLM2	4	96.2	0.748 ± 0.480	90.2 ± 53.7	0.570 ± 0.236	0.6
	Gemma3	4	100.0	0.698 ± 0.449	111.3 ± 62.5	0.540 ± 0.048	2.5
	TinyLlama	4	100.0	0.726 ± 0.529	102.8 ± 59.3	0.475 ± 0.268	2.1
Back-and-Forth	SmolLM2	4	91.2	0.467 ± 0.318	65.6 ± 52.7	0.507 ± 0.216	0.6
	Gemma3	4	100.0	0.416 ± 0.373	54.8 ± 58.1	0.240 ± 0.302	2.4
	TinyLlama	4	100.0	0.358 ± 0.364	63.2 ± 75.2	0.137 ± 0.205	2.0
Zigzag	SmolLM2	5	94.0	1.135 ± 0.715	87.3 ± 57.5	1.124 ± 0.410	0.6
	Gemma3	5	100.0	1.138 ± 0.467	75.2 ± 50.0	1.398 ± 0.122	2.4
	TinyLlama	5	100.0	0.981 ± 0.672	78.5 ± 55.2	0.955 ± 0.385	1.9
Slalom	SmolLM2	8	99.4	0.902 ± 0.599	93.7 ± 55.2	1.144 ± 0.297	0.7
	Gemma3	8	100.0	0.684 ± 0.463	93.7 ± 61.9	1.024 ± 0.096	2.6
	TinyLlama	8	100.0	0.770 ± 0.499	95.9 ± 59.9	0.852 ± 0.341	2.1
Overall	SmolLM2	–	96.2	0.840	83.3	1.229	0.7
	Gemma3	–	100	0.654	75.6	1.022	2.5
	TinyLlama	–	100	0.667	79.3	1.077	2.1

Across the 10 repeats per mission, Gemma3-1B yields the lowest overall waypoint and endpoint errors in this rerun, but the gap to TinyLlama-1.1B is modest. To assess whether the spatial performance differences across models at the mission-level are statistically meaningful, we executed a series of pairwise Wilcoxon signed-rank significance tests on the paired final trial-level endpoint error distributions ($N = 50$ values pooled across $10 \text{ trials} \times 5 \text{ missions}$ per model).

The pairwise Wilcoxon tests revealed the following:

- **SmolLM2-135M vs. Gemma3-1B:** The difference is statistically significant ($T = 381.5$; $p = 0.0129$), confirming Gemma3’s superior spatial stability.

- **SmolLM2-135M vs. TinyLlama-1.1B:** The difference is highly significant ($T = 187.0$; $p < 10^{-5}$), establishing SmolLM2 as significantly less reliable for multi-step path planning.
- **Gemma3-1B vs. TinyLlama-1.1B:** The spatial performance difference is nominally significant under paired Wilcoxon evaluation ($T = 421.0$, $p = 0.0363$). However, they remain closely matched, and the mission-specific Wilcoxon comparisons (e.g., L-Shape: $p = 0.492$, Slalom: $p = 0.275$, and Square Path: $p = 0.193$) confirm that their relative spatial superiority is highly mission-dependent and has not fully stabilized into a single dominant model.

The trajectory plots therefore serve as qualitative evidence of recurrent error patterns that substantiate these statistical dispersion differences across missions.

As an additional point of comparison, we characterize the performance of the deterministic RBCP baseline conceptually under the same mission configurations. Because the five mission specifications (Table 11) utilize only the standard, hardcoded target commands (e.g., “Go forward at 0.2 speed” or “Turn left fast”), the RBCP baseline achieves 100.0% parse success across all steps. Its execution is completely deterministic, resulting in near-zero waypoint error ($\bar{e}_{wp} \approx 0.04$ m) and final endpoint error ($e_{end} \approx 0.03$ m). These minimal errors represent purely the physical kinematic drift and wheel slippage modeled inside the Gazebo simulator, establishing a lower bound on the spatial error of the platform.

However, the major limitation is again semantic generalizability. If even a single word in the mission instruction sequence is altered (e.g., changing “Turn left fast” to “Rotate left-hand CCW” or “Stop” to “Halt”), the RBCP’s parse rate immediately drops to 0.0% on that step, triggering the fallback controller which publishes zero speed and permanently stalls the robot. Conversely, while parallel edge LLMs exhibit greater spatial deviation and path variance due to their stochastic generations, they successfully parse and execute the entire multi-step mission under linguistic variations without requiring any code modifications or halting the vehicle, proving the supreme value of neural model deployment on mobile robots.

6.4. Trajectory Analysis

Our spatial analysis evaluates the step-by-step ground-plane trajectories of the robot during different navigation trials. To keep the focus on the most informative behaviors, we omit the plots for the Slalom and Zigzag runs, which expand upon the rotational trends already seen elsewhere, and focus instead on the closed-loop Square Path (Figure 8) and the directionally reversing Back-and-Forth (Figure 9) tasks.

In these figures, the dashed lines outline the target path, the individual trials ($N = 10$) are drawn as thin, semi-transparent lines to show statistical spread, and the thick solid lines represent the mean calculated trajectory.

The physical execution profiles reveal distinct spatial control errors across the three models.

6.4.1. Closed-Path Errors and Drift Compounding (Square Path)

The Square Path mission presents a clear test of how rotational and translational errors accumulate over an extended sequence. The differences in trajectory control quality are shown in Figure 8:

- **SmolLM2-135M** (Figure 8a): Shows a steady angular drift. The model fails to generate the required 90° turn rate. This error compounds during subsequent legs, altering the starting orientation for each straight line segment. By step 4, the cumulative heading drift prevents path closure. The robot travels away from the target square, tracing an open, helical loop instead of a closed shape ($e_{end} = 0.739$ m).

- **Gemma3-1B** (Figure 8b) and **TinyLlama-1.1B** (Figure 8c): Both models show stable path control, keeping heading drift within tight bounds. Gemma3 exhibits slightly tighter corners, indicating a tiny overestimation of turning speed, while TinyLlama maintains segments relative to the target square, closing the final loop near the origin ($e_{\text{end}} = 0.447$ m). The grouping of the ten trials in both figures shows that the behavior of these ~ 1 B parameter models is highly consistent.

6.4.2. Directional Grounding and Sign Inversion (Back-and-Forth)

The Back-and-Forth mission checks if the models can translate coordinate reversals into negative linear commands ($x < 0$) without introducing unintended sway:

- **SmolLM2-135M** (Figure 9a): Fails to change direction. Because the model maps the backup instruction to positive velocities or random angular components ($z \in [0.00, 1.00]$), the robot moves forward or circles around rather than reversing.
- **TinyLlama-1.1B** (Figure 9c): Exhibits clean directional control, backing up along the same straight line with minimal lateral offset. This is consistent with its 82% semantic accuracy on individual “Back up” prompts, translating language instructions into stable linear trajectories ($e_{\text{end}} = 0.375$ m).
- **Gemma3-1B** (Figure 9b): Reverses after the forward segment but shows a steady rotational bias, curving clockwise instead of travelling straight back.

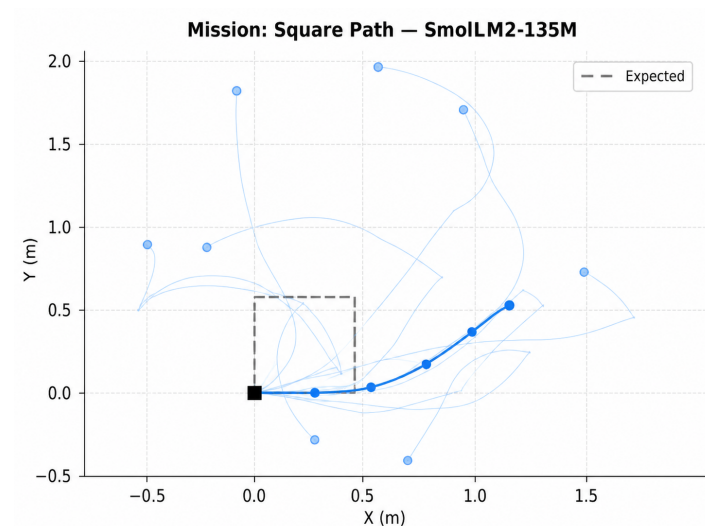
For longer navigation tasks such as Zigzag and Slalom, heading deviations represent the primary control bottleneck. As summarized in Table 12, Gemma3-1B achieves the lowest final position errors in Slalom ($e_{\text{end}} = 1.284$ m) and Zigzag ($e_{\text{end}} = 1.735$ m).

These multi-step alternating tasks highlight the performance trade-offs between Gemma3 and TinyLlama. Gemma3 profiles benefit from high rotational consistency and stable stop triggers between segments, which prevent trajectory overshoot. In contrast, TinyLlama-1.1B, despite high accuracy on straight lines, shows a small, systematic angular bias that accumulates as lateral drift in oscillatory paths. SmolLM2’s path traces show high dispersion across multiple steps, indicating that frequent command transitions exceed the stable control limit of sub-200M quantized edge models.

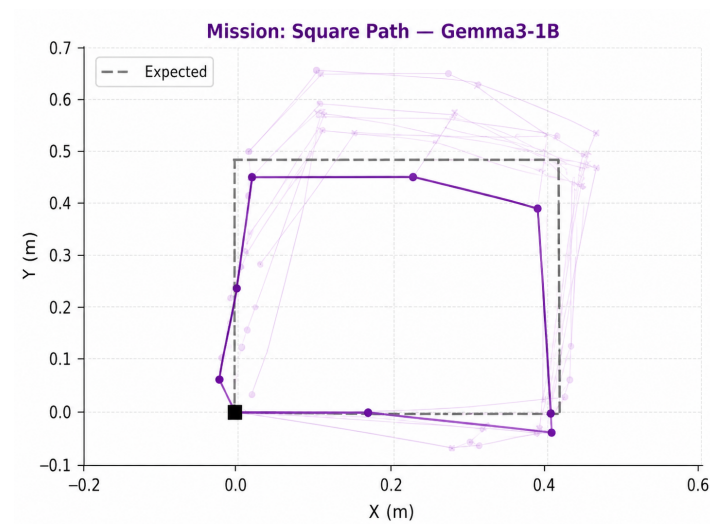
6.4.3. Error Accumulation

Figure 10 details how waypoint errors increase over time during the Square Path mission. While error increases with step count for all models, SmolLM2’s trajectory diverges much faster due to turning errors compounding over the eight-step path.

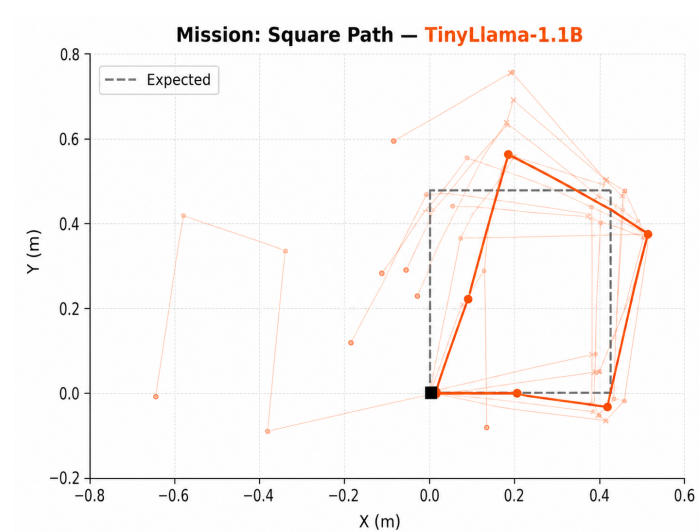
The final-step endpoint error across all evaluated scenarios is illustrated in Figure 11, demonstrating how linguistic grounding directly impacts physical execution over extended paths. For simple, low-turn paths such as the Back-and-Forth mission, drift is minimal and easily tolerated across all 1B-class architectures. However, for multi-step tasks including the Slalom and Zigzag, spatial drift quickly compounds. While the deterministic RBCP baseline completely breaks down under unconstrained human phrasing, its trajectory drift is mathematically zero on seen templates because its execution rate is instantaneous. In contrast, the sub-200M parameter model experiences significant turn-overshoot, resulting in cumulative position errors exceeding several meters. TinyLlama-1.1B and Gemma3-1B display consistent bounds, with Gemma3 achieving tighter trajectory envelopes despite its lower raw parsing efficiency.



(a) SmolLM2-135M

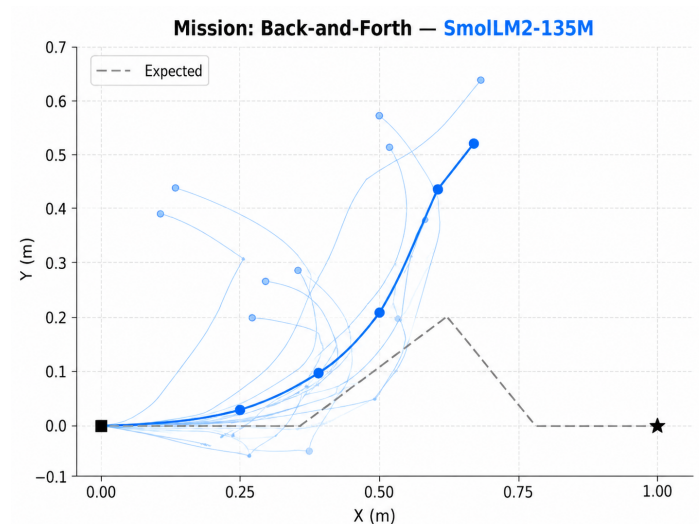


(b) Gemma3-1B

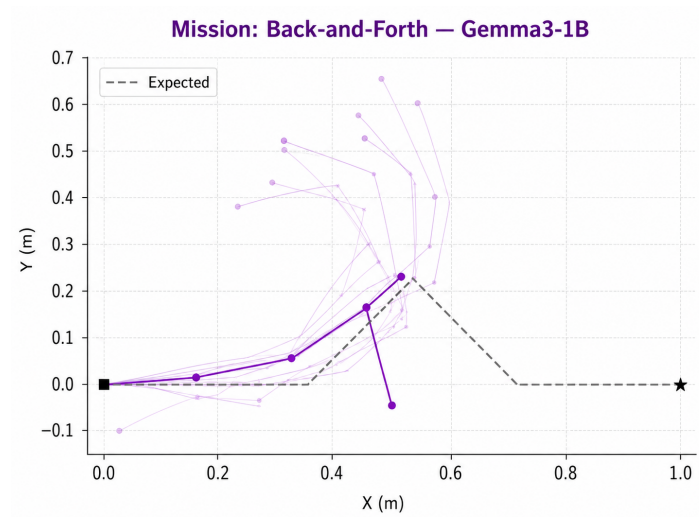


(c) TinyLlama-1.1B

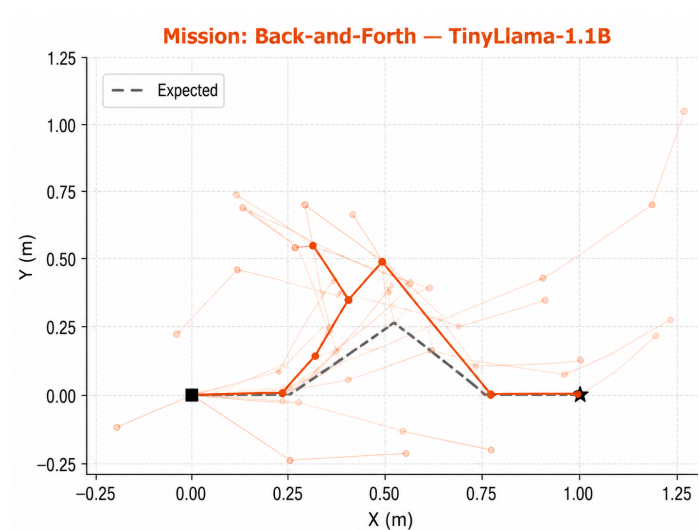
Figure 8. Spatial trajectory profiles for the multi-step closed-loop Square Path mission. Blue for SmolLM2-135M, purple for Gemma3-1B, and orange for TinyLlama-1.1B.



(a) SmolLM2-135M



(b) Gemma3-1B



(c) TinyLlama-1.1B

Figure 9. Spatial trajectory profiles for the directional inversion Back-and-Forth mission. Blue for SmolLM2-135M, purple for Gemma3-1B, and orange for TinyLlama-1.1B.

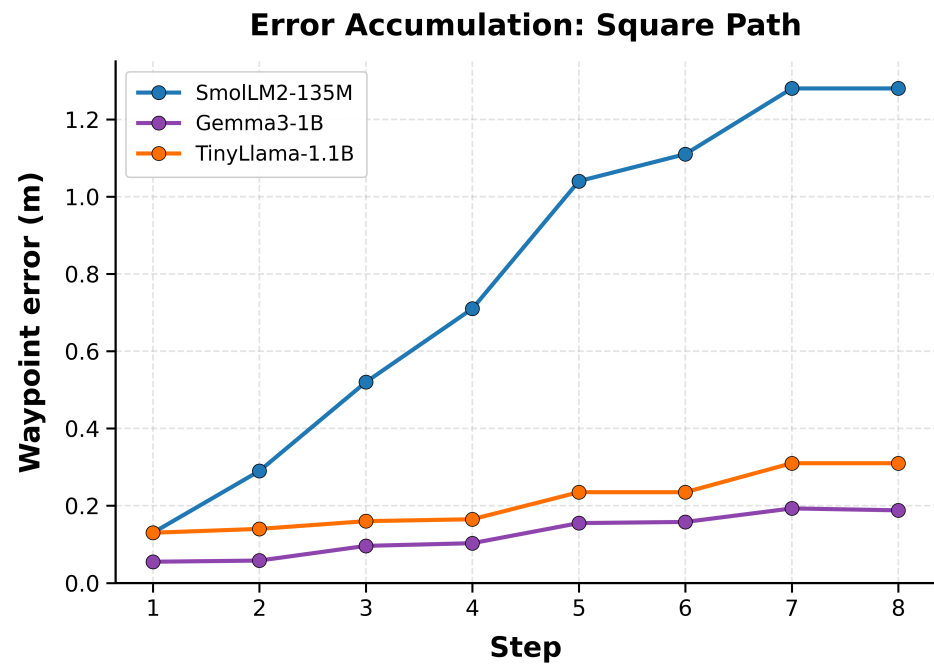


Figure 10. Cumulative Square Path waypoint error.

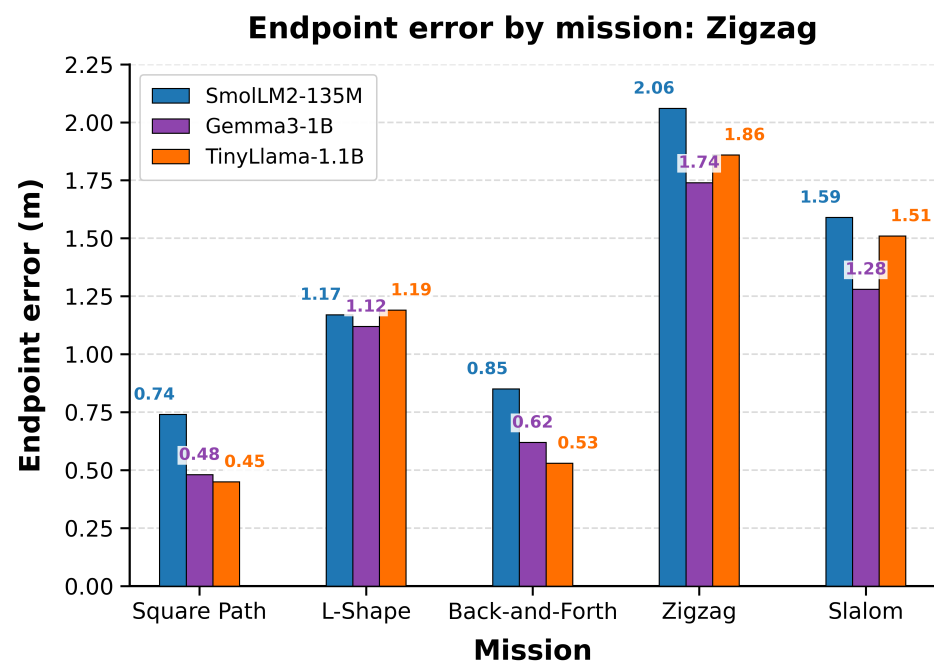


Figure 11. Endpoint error by mission.

To evaluate the operational cost associated with this spatial fidelity, Figure 12 tracks the mean per-step inference latency across each mission run. Although SmoLLM2-135M exhibits the poorest trajectory precision due to compounding turning errors, it completes steps roughly three times faster than larger models, consistently keeping per-step latency under 0.7s. This rapid response curve is critical for real-time avoidance adjustments on reactive platforms. On the other hand, Gemma3-1B and TinyLlama-1.1B require a substantial computational penalty per step (2.5s and 1.8s respectively), verifying their suitability as higher-level supervisory path planners rather than fast, direct motor-torque loops.

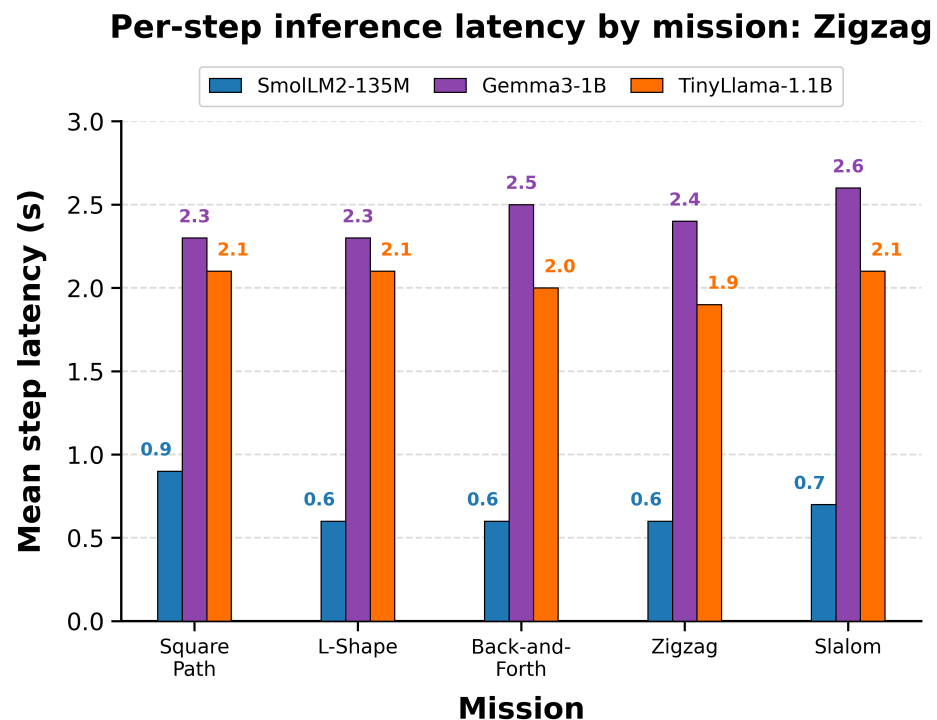


Figure 12. Per-step latency by mission.

6.5. Atomic vs. Mission Correlation

An important finding is that atomic semantic accuracy does not map monotonically to mission performance. TinyLlama-1.1B has the best overall atomic semantic accuracy (72.7%), yet Gemma3-1B attains slightly lower aggregate mission waypoint and endpoint errors in this rerun (0.654 m and 1.022 m vs. TinyLlama’s 0.667 m and 1.077 m). This discrepancy arises because mission success depends on where errors occur within a sequence, not just how often they occur in isolation. TinyLlama’s strong reverse and composite-command behavior still translates into the best Back-and-Forth and Square Path endpoints, whereas Gemma3’s highly stable “Stop” behavior and stronger pure-turn responses help on the longer turn-heavy missions. The mission table alone, however, is not sufficient to claim statistical superiority.

7. Discussion

7.1. Latency, Cold-Start, and the Parse–Semantic Distinction

The model-load cold-start effect remains operationally significant: the first warmup inference after loading inflates latency by roughly $6\text{--}9\times$ (4.8 s for SmolLM2, 14.1 s for TinyLlama, 17.3 s for Gemma3), driven by KV-cache initialization, memory-mapped weight paging, GGUF metadata parsing, and CPU cache warm-up in llama.cpp. The penalty does not scale linearly with model size (Gemma3-1B (769 MB)) still incurs a larger cold-start than the comparably-sized TinyLlama-1.1B (638 MB), implicating architectural rather than parametric causes. A practical mitigation is to issue a dummy “Stop” inference before deployment; once warmed, SmolLM2 supports reactive control at roughly half-second latency, while TinyLlama (1.7 s) and Gemma3 (2.9 s) are better viewed as slower supervisory controllers than as high-rate closed-loop policies.

Parse success and semantic accuracy are orthogonal metrics, with direct safety implications. Gemma3-1B achieves 100% parse compliance but only 51.7% semantic accuracy ($N = 300$), demonstrating that a correctly formatted output can still encode the wrong

motion. A controller validating parse success alone would accept directionally incorrect velocities, which is a dangerously optimistic assessment of model capability.

7.2. Model-Specific Failure Modes

Each model exhibits a distinct failure signature that maps to its capacity and architecture. SmolLM2-135M suffers from sign blindness and parse fragility: it scores only 2% on “Back up 0.1”, consistently mapping “back up” to angular rather than negative linear velocity ($z \in [0.00, 1.00]$), and produces empty or malformed outputs on 22% of “Stop” trials, indicating insufficient representational capacity for compositional operations such as sign inversion at the 135M/Q4 scale. Gemma3-1B exposes a compositional bottleneck: despite 100% accuracy on “Go forward” and 72% on “Turn left fast”, it collapses to 4% on “Move forward while turning left”, suggesting it can resolve individual primitives but cannot fuse them into a single twist. TinyLlama-1.1B is the most balanced, leading on 4 of 6 commands and producing the only non-zero score (38%) on “Rotate right slightly” with the correct sign ($\mu_z = -0.20$); its remaining weakness is angular-sign instability on “Turn left fast” ($\sigma_z = 0.64$). The command “Rotate right slightly” is still a universal weakness (0%, 0%, 38%): it requires compound grounding of direction (right $\Rightarrow z < 0$) and magnitude (slightly $\Rightarrow$ small), and Gemma3’s near-zero output effectively reduces the command to “Stop”, suggesting that even when individual components are tractable, their composition exceeds the capacity of sub-1.5B models.

7.3. Mission-Level Insights and a Hybrid Path Forward

The mission benchmark surfaces three findings invisible at the atomic level. First, atomic accuracy does not determine a single mission winner: TinyLlama leads the atomic benchmark, but Gemma3 edges it on aggregate mission waypoint and endpoint error in this rerun. Second, turn-heavy missions remain the primary error amplifier: Square Path, Zigzag, and Slalom widen the gap between SmolLM2 and the two larger models. Third, the latency–accuracy trade-off is non-trivial: SmolLM2 completes mission steps roughly $3\times$ faster (0.7 s vs. 2.1 s–2.5 s per step) but with materially higher trajectory error.

These findings suggest a hybrid architecture: SmolLM2-135M for latency-critical, unambiguous commands, TinyLlama-1.1B for reverse and mixed linear-angular commands, and Gemma3-1B for turn-heavy or stop-critical sequences where its stability is strongest. A confidence-based router that accepts the fast model’s output only when the parsed velocity falls within a learned plausibility window could reduce average latency for common commands while reserving more accurate handling for harder cases. In the present form, however, the benchmark supports this architecture as a design hypothesis rather than as a validated deployment recipe.

7.4. Quantization, PTQ Knowledge Loss, and Limitations

All models were evaluated in Q4_K_M GGUF format, which is an aggressive compression scheme necessary for embedded CPU deployment but known to trigger knowledge degradation. Post-training quantization (PTQ) affects neural network parameters unevenly [31]. While syntactic capability (such as complying with strict few-shot output formatting template structures) is generally highly preserved under 4-bit compression, semantic grounding and numerical reasoning features are significantly more vulnerable. Research shows that weights encoding precise directional vectors and proportional scaling factors undergo severe degradation under sub-8-bit quantization. SmolLM2’s total sign blindness under back-up commands may be directly caused by 4-bit quantization eroding the precise floating-point weight boundaries of these directional tokens; whether high-precision alternatives (such as Q8_0, FP16, or FP32) would resolve this remains an open question.

Notably, TinyLlama-1.1B and Gemma3-1B, both at a ~ 1 B parameter scale, exhibit markedly different performance profiles, confirming that network architecture and pre-training data composition play as crucial a role as post-training compression. TinyLlama's Llama-based backbone with a 2K native context window appears far better aligned with short-prompt, constrained-output velocity generation than Gemma3's 32K-context architecture. The challenges of executing CPU-local inference of small generative models within tight control loops are well documented [32], illustrating how resource contention on edge processors affects physical execution rates. Future work will systematically isolate these compression constraints by comparative benchmarking of identical models across multi-bit schemes (FP16 vs. Q8 vs. Q4) on embedded platforms to evaluate exact trade-offs [33].

Several threats to validity bound the scope of these findings. Internally, the default temperature ($T = 0.8$) introduces stochasticity without a fixed seed, though the large per-cell trial count ($N = 50$) yields narrow confidence intervals; the semantic-accuracy significance tests also reuse the same six prompts across repeated trials and therefore rest on a trial-level independence approximation.

Furthermore, our experiments are conducted under ideal Gazebo simulation conditions (no wheel slip, perfect observation, zero communication latency) representing a best-case performance lower bound. In real-world robotic environments, physical disturbances would inevitably accumulate and further amplify trajectory errors. For instance, physical wheel slip and mechanical misalignments on a dual-motor differential drive platform like the TurtleBot3 Burger significantly degrade pure rotational and translational accuracy over time, independent of parsing correctness. Similarly, high-frequency low-level sensor noise can introduce minor command jitter, while communication latencies within the ROS stack (e.g., publisher/subscriber delays, serial packet dropouts) would exacerbate the stochastic delays of CPU-based edge LLM inference, potentially disrupting real-time obstacle avoidance. The complexities of carrying out a successful sim-to-real transfer with language-driven robotic platforms are a major active area of study [34], showing that simulation benchmarks must be paired with physical safety wrappers to protect hardware.

Externally, results reflect a single CPU (i7-7500U), a simulation-only TurtleBot3 setup, and a six-command, five-mission suite; generalization to other edge platforms (Raspberry Pi 4, Jetson), real robots, and richer instruction sets requires further study. Another major external threat is the models' robustness to typing and lexical deviations. Because actual users will invariably produce grammatical irregularities, typing mistakes (e.g., "Turn leff at moderte speed" instead of "Turn left"), or colloquial paraphrases, checking model resilience to unconstrained input noise represents a crucial threshold for real-world suitability. While edge LLMs map synonyms and phrases into continuous latent spaces, extreme character typos might still trigger formatting failures or hallucinated directions. Explicitly benchmarking spelling and grammatical variations constitutes a key priority for future empirical work.

For construct validity, the semantic windows (Table 4) involve judgment calls; in particular, the binary boundary for the "Rotate right slightly" command (requiring $z < -0.05$) creates a strict threshold effect. An output like $z = -0.01$ correctly captures the clockwise directional intent of "right" and qualitative context of "slightly", yet is discarded as incorrect, which artificially depresses semantic success rates. Future work should implement continuous error curves (such as magnitude root-mean-square deviations) to complement binary thresholds. The few-shot prompt also reuses command families that are close to the test set, so the results should be interpreted as in-distribution instruction following rather than as a strong paraphrase generalization benchmark. Finally, although we compared our models against a deterministic rule-based baseline (RBCP) to highlight the trade-offs between execution speed and semantic robustness, we did not evaluate against larger LLM

baselines (such as 7B-parameter models) or ML-based intent classifiers, so the present study is limited to evaluating relative and absolute differences among quantized models under 1.5 B parameters.

8. Conclusions

We presented a two-tier benchmark of three sub-1.5B quantized LLMs for physical-velocity control on a TurtleBot3 Burger in Gazebo, spanning 900 atomic trials and 150 multi-step missions (870 steps). The key findings are: (i) warm-start latency ranges from 0.506 s (SmolLM2) to 2.903 s (Gemma3), while model-load cold-start incurs a further 6–9× penalty and therefore still requires explicit warmup in any practical deployment; (ii) TinyLlama-1.1B achieves the strongest atomic results in this benchmark, reaching 72.7% semantic accuracy versus 51.7% for Gemma3-1B and 42.7% for SmolLM2-135M under the trial-level independence approximation used here; (iii) the mission-level evaluation does not yield a single dominant model, with Gemma3-1B attaining the lowest aggregate waypoint and endpoint errors while TinyLlama-1.1B remains competitive, with Wilcoxon tests demonstrating that their physical path deviation differences are highly mission-dependent and do not establish absolute spatial dominance; and (iv) parse success and semantic accuracy remain distinct metrics, because 100% parse compliance can still mask substantial motion-level error.

Importantly, because our trial-level evaluations are conducted sequentially within loaded model sessions, the nominal significance *p*-values reported in this benchmark must be interpreted with caution as descriptive sample statistics rather than strict inferential general population estimates.

Overall, the results support small quantized LLMs as promising components for edge-local, supervised natural-language robot interfaces, but not yet as reliable standalone controllers for unsupervised real-world deployment. Closing the remaining accuracy gap will likely require domain-specific fine-tuning, confidence-aware routing, stronger baselines, and evaluation on real embedded hardware with held-out paraphrase command sets. Future work will therefore pursue physical anonymized TurtleBot3 validation, LoRA-based fine-tuning [35] on robot velocity corpora, greedy decoding ($T = 0$) to reduce output variance, multi-platform benchmarking on Raspberry Pi 4 and Jetson Orin Nano, fine-tuned intent classifier baselines, and an expanded command set covering paraphrases, obstacle avoidance, and waypoint navigation.

Author Contributions: Conceptualization, B.A. and K.A.; methodology, B.A., G.E. and K.A.; software, B.A.; validation, B.A. and G.E.; formal analysis, B.A. and G.E.; investigation, B.A.; resources, B.A. and K.A.; data curation, B.A.; visualization, B.A.; writing—original draft preparation, B.A.; writing—review and editing, G.E. and K.A.; supervision, G.E. and K.A.; project administration, K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by Istanbul Topkapi University within the scope of its publication incentive program.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The complete software codebase, ROS node implementations, automated evaluation harnesses, and anonymized raw experimental trial databases for both the atomic and mission-level benchmarks are publicly available on GitHub <https://github.com/burakaggul/quantized-edge-llm-robotics-benchmark>, accessed on 23 June 2026. All experimental conditions, models, execution scripts, and calibration configurations are fully documented within the repository readme to ensure absolute transparency and complete local reproducibility of the academic benchmarking results.

Acknowledgments: The authors acknowledge the open-source robotics, ROS, Gazebo, llama.cpp, and model-development communities whose tools made the benchmarking workflow possible.

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

References

- Kim, M.J.; Pertsch, K.; Karamcheti, S.; Xiao, T.; Balakrishna, A.; Nair, S.; Rafailov, R.; Foster, E.P.; Sanketi, P.R.; Vuong, Q.; et al. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of Machine Learning Research*; PMLR: Cambridge, MA, USA, 2025; Volume 270, pp. 2679–2713.
- Zhang, W.; Kong, X.; Dewitt, C.; Bräunl, T.; Hong, J.B. Enhancing Reliability in LLM-Integrated Robotic Systems: A Unified Approach to Security and Safety. *J. Syst. Softw.* **2026**, *231*, 112614. <https://doi.org/10.1016/j.jss.2025.112614>.
- Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. In *Proceedings of the Advances in Neural Information Processing Systems*, Virtual, 6–12 December 2020; Volume 33, pp. 1877–1901.
- Ichter, B.; Brohan, A.; Chebotar, Y.; Finn, C.; Hausman, K.; Herzog, A.; Ho, D.; Ibarz, J.; Irpan, A.; Jang, E.; et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of Machine Learning Research*; PMLR: Cambridge, MA, USA, 2023; Volume 205, pp. 287–318.
- 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. In *Proceedings of Machine Learning Research*; PMLR: Cambridge, MA, USA, 2023; Volume 202, pp. 8469–8488.
- Huang, W.; Xia, F.; Xiao, T.; Chan, H.; Liang, J.; Florence, P.; Zeng, A.; Tompson, J.; Mordatch, I.; Chebotar, Y.; et al. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Proceedings of Machine Learning Research*; PMLR: Cambridge, MA, USA, 2023; Volume 205, pp. 1769–1782.
- Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; Zeng, A. Code as Policies: Language Model Programs for Embodied Control. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 29 May–2 June 2023; pp. 9493–9500. <https://doi.org/10.1109/ICRA48891.2023.10160591>.
- Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; Garg, A. ProgPrompt: Generating Situational Robot Plans with Spatially Grounded Language Models. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 29 May–2 June 2023. <https://doi.org/10.1109/icra48891.2023.10161317>.
- Shah, D.; Osinski, B.; Ichter, B.; Levine, S. LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action. In *Proceedings of Machine Learning Research*; PMLR: Cambridge, MA, USA, 2023; Volume 205, pp. 492–504.
- 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; pp. 10608–10615. <https://doi.org/10.1109/ICRA48891.2023.10160969>.
- Oelerich, T.; Hartl-Nesic, C.; Kugi, A. Language-guided Manipulator Motion Planning with Bounded Task Space. In *Proceedings of the 8th Conference on Robot Learning*; Agrawal, P., Kroemer, O., Burgard, W., Eds.; PMLR: Cambridge, MA, USA, 2025; Volume 270, pp. 749–779.
- Chen, W.; Koenig, S.; Dilkina, B. RePrompt: Planning by Automatic Prompt Engineering for Large Language Models Agents. *arXiv* **2025**, arXiv:2406.11132. <https://doi.org/10.48550/arXiv.2406.11132>.
- 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. <https://doi.org/10.1007/s11370-024-00550-5>.
- Wang, J.; Shi, E.; Hu, H.; Ma, C.; Liu, Y.; Wang, X.; Yao, Y.; Liu, X.; Ge, B.; Zhang, S.; et al. Large Language Models for Robotics: Opportunities, Challenges, and Perspectives. *J. Autom. Intell.* **2025**, *4*, 52–64. <https://doi.org/10.1016/j.jai.2024.12.003>.
- Brohan, A.; Brown, N.; Carbajal, J.; Chebotar, Y.; Chen, X.; Choromanski, K.; Ding, T.; Driess, D.; Dubey, A.; Finn, C.; et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. *arXiv* **2023**, arXiv:2307.15818. <https://doi.org/10.48550/arXiv.2307.15818>.
- Liu, Y.; Sun, Q.; Kapadia, D.R. Integrating Large Language Models into Robotic Autonomy: A Review of Motion, Voice, and Training Pipelines. *AI* **2025**, *6*, 158. <https://doi.org/10.3390/ai6070158>.
- Frantar, E.; Ashkboos, S.; Hoefler, T.; Alistarh, D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *arXiv* **2023**, arXiv:2210.17323. <https://doi.org/10.48550/arXiv.2210.17323>.
- Gerganov, G. llama.cpp: Inference of LLaMA Model in Pure C/C++, 2023. Available online: <https://github.com/ggerganov/llama.cpp> (accessed on 23 June 2026).
- Allal, L.B.; Lozhkov, A.; Bakouch, E.; Blázquez, G.M.; Penedo, G.; Tunstall, L.; Marafioti, A.; Kydlíček, H.; Lajarín, A.P.; Srivastav, V.; et al. SmolLM2: When Smol Goes Big—Data-Centric Training of a Small Language Model. *arXiv* **2025**, arXiv:2502.02737. <https://doi.org/10.48550/arXiv.2502.02737>.

20. Team, G.; Kamath, A.; Ferret, J.; Pathak, S.; Vieillard, N.; Merhej, R.; Perrin, S.; Matejovicova, T.; Ramé, A.; Rivière, M.; et al. Gemma 3 Technical Report. *arXiv* **2025**, arXiv:2503.19786. <https://doi.org/10.48550/arXiv.2503.19786>
21. Zhang, P.; Zeng, G.; Wang, T.; Lu, W. TinyLlama: An Open-Source Small Language Model. *arXiv* **2024**, arXiv:2401.02385. <https://doi.org/10.48550/arXiv.2401.02385>.
22. Dettmers, T.; Lewis, M.; Belkada, Y.; Zettlemoyer, L. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *arXiv* **2022**, arXiv:2208.07339. <https://doi.org/10.48550/arXiv.2208.07339>
23. Mittal, S. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *J. Syst. Archit.* **2019**, *97*, 428–442. <https://doi.org/10.1016/j.sysarc.2019.01.011>.
24. NVIDIA. TensorRT-LLM: An Open-Source Library for Optimizing Inference Performance of LLMs. 2024. Available online: <https://github.com/NVIDIA/TensorRT-LLM> (accessed on 23 June 2026).
25. Raptis, E.K.; Kapoutsis, A.C.; Kosmatopoulos, E.B. Agentic LLM-based Robotic Systems for Real-World Applications: A Review on Their Agenticity and Ethics. *Front. Robot.* **2025**, *12*, 1605405. <https://doi.org/10.3389/frobt.2025.1605405>.
26. Jeong, H.; Lee, H.; Kim, C.; Shin, S. A Survey of Robot Intelligence with Large Language Models. *Appl. Sci.* **2024**, *14*, 8868. <https://doi.org/10.3390/app14198868>.
27. Das, M.; Hussain, Z.; Nawaz, M. Latency-Aware Benchmarking of Large Language Models for Natural-Language Robot Navigation in ROS 2. *Sensors* **2026**, *26*, 608. <https://doi.org/10.3390/s26020608>.
28. Tan, Z.; Li, Z.; Liu, T.; Wang, H.; Yun, H.; Zeng, M.; Chen, P.; Zhang, Z.; Gao, Y.; Wang, R.; et al. Aligning Large Language Models with Implicit Preferences from User-Generated Content. In Proceedings of the Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Vienna, Austria, 27 July–1 August 2025. <https://doi.org/10.18653/v1/2025.acl-long.384>.
29. ROBOTIS. TurtleBot3: Overview. 2025. Available online: <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/> (accessed on 23 June 2026).
30. Kruskal, W.H.; Wallis, W.A. Use of Ranks in One-Criterion Variance Analysis. *J. Am. Stat. Assoc.* **1952**, *47*, 583–621.
31. Zhao, J.; Wang, M.; Zhang, M.; Shang, Y.; Liu, X.; Wang, Y.; Zhang, M.; Nie, L. Benchmarking Post-Training Quantization in LLMs: Comprehensive Taxonomy, Unified Evaluation, and Comparative Analysis. *arXiv* **2025**, arXiv:2502.13178. <https://doi.org/10.48550/arXiv.2502.13178>.
32. Psiris, A.; Argyriou, V.; Markakis, E.K.; Sarigiannidis, P.; Gavves, E.; Bekris, K.; Ajoudani, A.; Papadopoulos, G.T. Foundation Models in Robotics: A Comprehensive Review of Methods, Models, Datasets, Challenges and Future Research Directions. *arXiv* **2026**, arXiv:2604.15395. <https://doi.org/10.48550/arXiv.2604.15395>.
33. Williams, J.; Gupta, K.D.; George, R.; Sarkar, M. LiteVLA-Edge: Quantized On-Device Multimodal Control for Embedded Robotics. *arXiv* **2026**, arXiv:2603.03380. <https://doi.org/10.48550/arXiv.2603.03380>.
34. Zhao, W.; Queralta, J.P.; Westerlund, T. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, Australia, 1–4 December 2020; pp. 737–744. <https://doi.org/10.1109/SSCI47803.2020.9308468>.
35. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv* **2021**, arXiv:2106.09685 <https://doi.org/10.48550/arXiv.2106.09685>.

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.