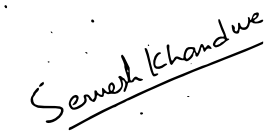
Technische Universität München
School of Computation, Information and Technology (CIT)
Department of Computer Engineering
Chair of Cyber-Physical Systems in Heilbronn
Prof. Dr. rer. nat. Amr Alanwar

Bachelor Thesis

Exploring and Comparing Local Large Language Models for Real-World Applications – A Case Study in the CPS Lab

Author:	Servesh Khandwe
Matriculation Number:	03752952
Address:	Max Planck Straße 27 74081 Heilbronn
Advisor:	Ahmad Hafez
Begin:	July 23, 2025
End:	October 8, 2025

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.



Heilbronn, October 8, 2025 — Signature

Place, Date

Heilbronn, October 8, 2025 — Signature

Place, Date

Use of *AI Assistants* for Research Purposes

I have used AI Assistant(s) for the purposes of my research as part of this thesis.

Yes ☐ ~~No~~ ☐

Explanation: I only used AI assistants for formatting, formal writing, grammar corrections, and text restructuring.

I confirm in signing below, that I have reported all usage of AI Assistants for my research, and that the report is truthful and complete.

Heilbronn, October 8, 2025

Place, Date

Signature

Abstract

This thesis was written to motivate the adoption of local large language models (LLMs) for future cyber-physical systems (CPS) projects. The report explores several CPS use cases in which local LLMs can be used and replace proprietary models such as OpenAI's and documents experiments conducted with these alternatives.

To demonstrate feasibility, I integrated a local LLM into my professor's recent work that employs reachability-analysis and zonotopes as a safety layer for navigation. The report also examines the safe-navigation project in depth, including the laboratory motion-capture setup. Finally, we compare the local LLM's performance in this context, summarize the findings, and outline planned future work.

Keywords: *Local LLMs, Reachability Analysis, Zonotopes*

Acknowledgements

I would like to express my deepest gratitude to the individuals who have supported and guided me throughout the process of writing this thesis.

First and foremost, I extend my sincere thanks to my advisor, Mr. Ahmad Hafez, for his invaluable guidance, continuous support, and immense patience. His expertise and insightful feedback were crucial in shaping this work and navigating the research challenges. I am deeply grateful for the time and effort he dedicated to this project.

I am also profoundly thankful to my examiner, Prof. Dr. Amr Alanwar, for providing me with the opportunity to work on this fascinating topic under the Chair of Cyber-Physical Systems. His vision and the resources made available were fundamental to the completion of this thesis.

Furthermore, I would like to acknowledge the support of my family and friends, whose encouragement and understanding were a constant source of motivation throughout this academic journey.

Finally, I thank all those who, directly or indirectly, contributed to the realization of this work.

Heilbronn, October 8, 2025

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial intelligence is rapidly moving from cloud-bound services into the tight control loops of cyber–physical systems (CPS). Among recent advances, locally hosted large language models (LLMs) have emerged as versatile reasoning engines that can operate close to the edge, where latency, cost, privacy, and robustness matter most[5].

This thesis explores whether such local LLMs can be used to drive an autonomous mobile robot while guaranteeing safety through formal verification (the safety layer of zonotopes), and whether a carefully engineered system architecture can make this practical in a laboratory setting.

The work is grounded in the SafeLLMRA [6] framework for safe robot control[7], where candidate actions proposed by an LLM are vetted by a reachability-analysis based safety layer using (constrained) zonotopes[8, 9] before they are executed. Concretely, the thesis investigates an NVIDIA JetRacer platform in the CPS laboratory, integrates a high-precision motion capture (MOCAP) system for ground truth, and replaces proprietary, cloud LLMs with open-source models served locally over an OpenAI-compatible API. We will also look into refactored attempt from a multi-node ROS design into a single monolithic node to minimize inter-process delays that can be crucial when looking into real-time performance.

1.1 Motivation

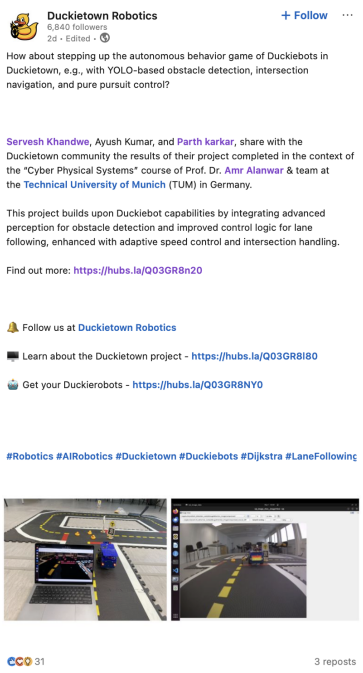
The motivation for this thesis is rooted in a combination of foundational academic experience, practical industry exposure, and the rapidly evolving landscape of artificial intelligence.

The initial groundwork was laid during the author's bachelor's practical course, "Research and Adaptation of Efficient Autonomous Driving Methods in Duckietown," supervised by Professor Amr Alanwar and Ahmed Hafez. This project involved implementing a complete autonomous navigation stack in the Duckietown environment, including lane following, stop

line detection, and intersection navigation using AprilTags[10]. This experience provided a robust foundation in the Robot Operating System (ROS)[11] and the practical challenges of programming autonomous robotic systems.

This academic foundation was later contextualized by practical industry experience at Porsche Digital, where I am working as a working student, invovled in deep tech projects utilizing local Large Language Models (LLMs) deployed on dedicated hardware. This hands-on exposure revealed the significant potential of local LLMs beyond traditional generative AI tasks. It became evident that these models possess an emerging capability for complex reasoning that mirrors human cognitive processes, sparking the core research interest of this thesis: exploring how local LLMs could be leveraged for Cyber-Physical Systems (CPS), where real-time decision-making and autonomous reasoning are critical.

Figure 1.1: Duckietown Robotics posted our project on LinkedIn (Used with permission)

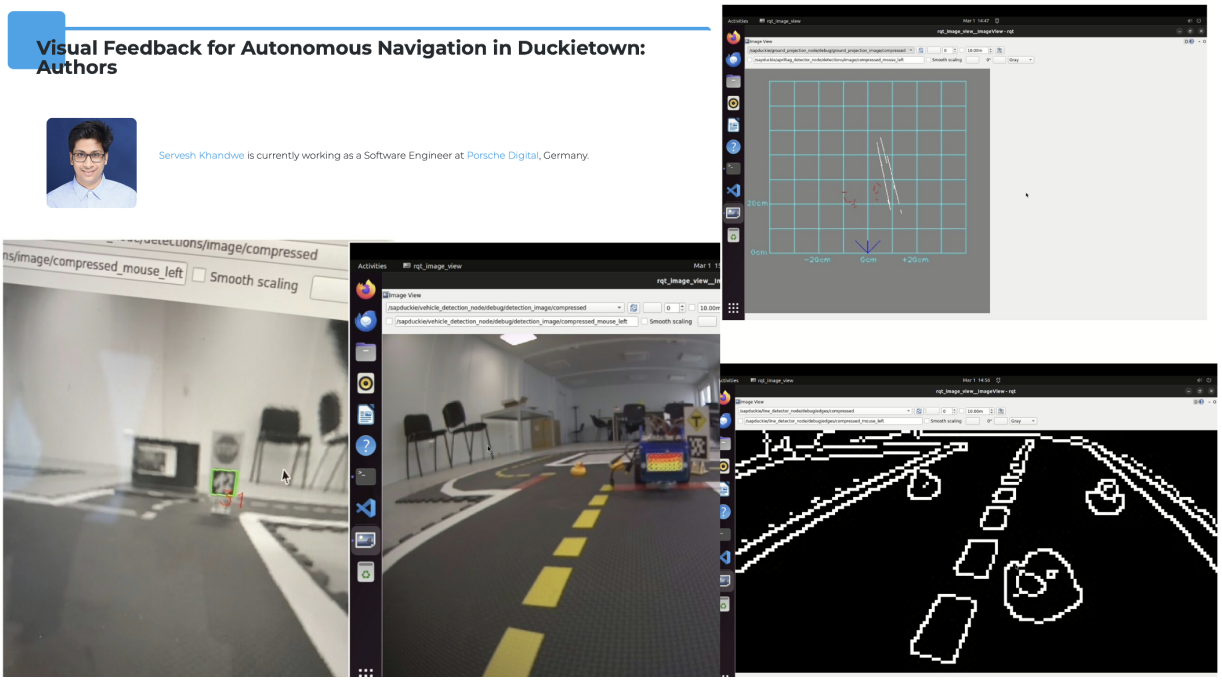


1.2 Research Questions

This thesis aims to investigate the viability of replacing proprietary Large Language Models (LLMs) with locally-hosted, open-source alternatives for autonomous vehicle navigation, and to refactor the underlying system architecture for enhanced performance. The research is guided by the following questions:

1. **Local LLMs for Autonomous Navigation:** How do locally hosted, open-source LLMs (e.g., Llama 3, Qwen2/3, DeepSeek) perform for autonomous navigation within

Figure 1.2: Overview of the Duckietown project that provided foundational ROS and navigation experience[4]



the SafeLLMRA framework, and what cost–latency–accuracy trade-offs emerge compared to proprietary models when generating control outputs (e.g., throttle and angular velocity)?

2. **Comparative Analysis of Open-Source LLMs:** How do leading open-source LLMs differ across key navigation-relevant metrics—such as control accuracy, safety violations, latency/jitter, determinism, resource usage, and operating cost—and what factors (model size, quantization, prompt format, safety alignment) explain these differences?

3. **System Architecture Optimization:** What performance improvements are achieved by refactoring the JetRacer platform from a multi-node architecture to a single monolithic process, and what risks or trade-offs (fault isolation, maintainability, extensibility) accompany this change?

1.3 Contribution

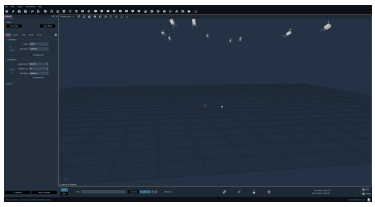
The primary contribution of this thesis is a comprehensive empirical investigation into the use of locally-hosted, open-source Large Language Models (LLMs) for real-time autonomous navigation, presenting a cost-effective and high-performance alternative to proprietary, cloud-based models. This work provides novel insights for researchers and devel-

opers aiming to build capable and accessible autonomous systems.

The specific contributions are as follows:

- **Comparative Analysis of Local LLMs for Navigation:** This thesis presents a systematic comparison of leading open-source LLMs (including Llama 3, Qwen2, and DeepSeek) for the task of autonomous navigation in an environment with obstacles. The models are evaluated on their ability to generate accurate vehicle control commands, with performance benchmarked against proprietary models on metrics such as latency, accuracy, and robustness.
- **System Performance Optimization:** A key contribution is the refactoring of the standard multi-node JetRacer software architecture into a streamlined, monolithic process. This work quantifies the resulting improvements in system responsiveness and navigation speed, providing empirical evidence for architectural design choices in real-time robotic systems.
- **Development of a Stable Experimental Platform:** A significant portion of this project was dedicated to foundational engineering work that enabled the core research. This included:
  - **Repository Stabilization:** Extensively debugging and resolving instabilities within the existing JetRacer repository to create a reliable platform for experimentation.
  - **Motion Capture System Integration:** Setting up and meticulously calibrating the Nokov motion capture system to serve as a high-precision ground truth for vehicle tracking and evaluation. This involved creating rigid body profiles and broadcasting coordinate data over a local network.
  - **Dedicated Network Infrastructure:** Establishing a private local network to facilitate low-latency, real-time communication between the JetRacer agent and the high-performance PC hosting the local LLMs.

Figure 1.3: Our Jetracer in motion capture system



1.4 Methodology

1.4.1 Design Science Research Framework

The DSR approach for this thesis encompasses three primary components: stakeholders, artifacts, and evaluation processes[12].

Stakeholders: The primary stakeholders include autonomous vehicle researchers, robotics engineers, and practitioners working on edge AI deployments. Secondary stakeholders encompass the broader CPS research community and industry professionals seeking cost-effective alternatives to proprietary AI solutions.

Artifacts: The key artifacts developed in this research include: (1) an optimized system architecture for local LLM deployment in autonomous navigation, (2) a comprehensive evaluation framework for comparing open-source LLMs against proprietary alternatives, and (3) a stabilized experimental platform integrating motion capture systems with autonomous vehicle hardware.

Evaluation Framework: The evaluation process employs quantitative metrics including navigation accuracy, system latency, and robustness under various environmental conditions, benchmarked against established proprietary solutions.

1.4.2 Infrastructure Development and Platform Selection

The research required establishing a robust experimental infrastructure capable of supporting real-time LLM inference on autonomous platforms. After consultation with the research chair and assessment of available resources, an NVIDIA RTX 4090 GPU was identified as the primary computational resource. This high-performance hardware was relocated to the motion capture laboratory on the second floor to enable integration with the existing Nokov motion capture system.

Given the computational limitations of edge devices for local LLM execution, a server-client architecture was designed where the high-performance PC serves as an LLM inference server, accessible to edge devices through OpenAI-compatible API endpoints over a secure local network. This approach addresses the fundamental challenge of deploying computationally intensive LLMs on resource-constrained autonomous platforms while maintaining low-latency communication essential for real-time navigation tasks.

The selection of the SafeLLMRA framework as the experimental platform was motivated by several factors: (1) its existing integration of LLMs for autonomous navigation, (2) the author's prior experience with ROS-based robotic systems through the Duckietown practical course, and (3) the availability of institutional expertise and support for this platform. The SafeLLMRA framework provided an ideal testbed for investigating the research questions while building upon established autonomous navigation methodologies.

1.4.3 Technical Implementation Approach

The methodology encompasses both theoretical investigation and practical implementation. The theoretical component involves comparative analysis of open-source LLMs against proprietary baselines, while the practical component focuses on system optimization through architectural refactoring and performance enhancement.

A critical aspect of the methodology involved addressing foundational technical challenges, including repository stabilization, motion capture system calibration, and network infrastructure establishment. These preparatory steps were essential to create a reliable experimental platform capable of generating reproducible results for the comparative analysis of local LLMs in autonomous navigation scenarios.

Chapter 2

Related Work

2.1 Reachability with Matrix and Logical Zonotopes

Alanwar et al. (2021) introduced a data-driven approach for reachability analysis using *matrix zonotopes*, enabling the computation of reachable sets directly from noisy data without requiring an explicit system model[13]. This method aligns with our data-centric setup, where empirical mocap signals and robot telemetry underpin safety checks. Further, Alanwar et al. (2023) proposed *polynomial logical zonotopes*, a set representation that facilitates exact and computationally efficient reachability analysis of logical systems[14]. In the context of SafeLLMRA, these ideas complement the constrained-zonotope safety layer by offering principled tools for verifying properties when discrete, rule-based logic (e.g., fallback modes and safety overrides) interacts with continuous robot dynamics.

2.2 LLM-Controlled Robotics and Safety Frameworks

2.2.1 Safe LLM Control with Reachability Analysis

Hafez et al. (2025) present a comprehensive framework for safe LLM-controlled robots using formal guarantees via reachability analysis[6]. This foundational work introduces a data-driven reachability framework that provides formal safety guarantees for LLM-controlled robotic systems without relying on precise analytical models. The framework leverages offline trajectory data to compute overapproximated reachable sets, ensuring all possible system trajectories remain within safe operational limits. Their approach demonstrates successful integration of LLMs with zero-shot learning capabilities and zonotope-based safety layers, validating the concept through both TurtleBot3 simulations and real JetRacer experiments in motion capture environments. This work forms the theoretical foundation upon which our SafeLLMRA implementation is built, extending their framework to explore local LLM deployment and comparative performance analysis across different model architectures.

2.2.2 Deep Reinforcement Learning for Autonomous Vehicles

The integration of reinforcement learning with embedded robotic platforms has shown promising results for autonomous navigation. Masato-ka's airc-rl-agent project demonstrates a compelling approach using deep reinforcement learning for JetBot and JetRacer platforms[15]. This work combines Soft Actor-Critic (SAC) with Variational Autoencoder (VAE) for rapid policy training directly on Jetson Nano hardware, achieving autonomous driving behavior in 10-15 minutes without human demonstration data. The approach leverages state representation learning through pre-trained VAE models on cloud servers, which significantly reduces the computational burden on edge devices while maintaining effective learning performance.

The system architecture employs a two-stage training process: first, a VAE is trained on cloud infrastructure using collected camera images to learn compact state representations; second, the SAC algorithm uses these learned representations to develop control policies directly on the Jetson Nano. This methodology addresses the computational constraints of edge deployment while maintaining learning efficacy. The work demonstrates successful navigation on both simulated environments (DonkeySim) and real robotic platforms, showcasing the practical viability of RL-based autonomous control for resource-constrained systems.

This approach represents a potential future direction for integrating reinforcement learning capabilities with our LLM-based framework, where the LLM could potentially guide or supplement the RL agent's decision-making process, combining the reasoning capabilities of language models with the learned behaviors from reinforcement learning.

2.2.3 Formal Verification and Safety in Autonomous Systems

The integration of formal verification methods in autonomous systems has become increasingly critical as these systems transition from laboratory settings to real-world deployments. Traditional approaches to ensuring safety in autonomous vehicles and robotic systems often rely on extensive testing, simulation, and statistical validation[16]. However, these methods cannot provide the exhaustive guarantees required for safety-critical applications, leading to increased interest in formal verification techniques.

Reachability analysis, particularly when combined with set-based representations like zonotopes, offers a mathematically rigorous approach to safety verification. Unlike empirical testing methods, reachability analysis provides formal guarantees about system behavior by computing the set of all possible future states given current conditions and system dynamics. This is particularly valuable in autonomous navigation scenarios where traditional control theory meets modern AI decision-making, as it enables verification of AI-generated commands before execution.

The challenge of integrating AI-based decision-making with formal safety guarantees represents a significant research frontier. While traditional control systems operate with

well-defined mathematical models and predictable behaviors, LLM-based controllers introduce probabilistic decision-making that must be reconciled with safety requirements. The SafeLLMRA framework addresses this challenge by treating the LLM as a black-box command generator while employing reachability analysis to verify the safety of generated commands, thus combining the adaptability of modern AI with the rigor of formal verification methods.

2.3 Performance Evaluation via Benchmarks

Although standard evaluations like MMLU and GSM8K offer crucial insights into the core reasoning abilities of a model, they serve primarily as foundational indicators. For specialized fields, such as autonomous navigation within a closed-loop Cyber-Physical System (CPS), we employ specific navigation metrics to apply these core skills in a practical context. These benchmarks measure the fundamental mathematical and language understanding skills that are critical for achieving effective autonomous performance.

However, it is important to recognize that strong performance on general benchmarks does not always guarantee success in specific, real-world applications. The SafeLLMRA project, for instance, highlights scenarios where benchmark results may not directly translate to operational effectiveness. To better understand these foundational tests, we can examine several prominent examples, including MMLU for language understanding, HELLASWAG for commonsense reasoning, BIG-bench for general AI tasks, and GSM8K for mathematical problem-solving.

2.3.1 GSM8K Benchmark

The GSM8K (Grade School Math 8K) benchmark is a dataset specifically created to assess the multi-step mathematical reasoning capabilities of large language models [1]. This collection contains 1,319 word problems, which were authored by human experts to reflect grade-school-level mathematics. Each problem is designed to be solved in two to eight steps using basic arithmetic operations.[1].

2.3.2 MMLU (Massive Multitask Language Understanding) Benchmark

The MMLU-Pro benchmark is a comprehensive evaluation of large language models across various categories, including computer science, mathematics, physics, chemistry, and more. It's designed to assess a model's ability to understand and apply knowledge across a wide range of subjects, providing a robust measure of general intelligence[17, 18]. Also see the curated results for the Computer Science subset[2].

Table 2.1: Comparison of various language models with their accuracy and evaluation methodology[1].

Rank	Model	Accuracy	Methodology
1	Anthropic Claude 3	95%	Zero shot
2	Google Gemini Ultra	94.4%	Majority Vote, 32 Generations
3	OpenAI GPT-4	92%	SFT & 5-shot CoT
4	Anthropic Claude 2	88%	Zero shot
5	Google Gemini Pro	86.5%	Majority Vote, 32 Generations
6	Inflection 2	81.4%	8-shot Learning
7	Mistral Large	81%	5-shot Learning
8	Google PaLM 2	80%	5-shot Learning
9	Mistral Medium	66.7%	5-shot Learning
10	xAI Grok 1	62.9%	8-shot Learning
11	Mistral Mixtral 8x7b	58.4%	5-shot Learning
12	OpenAI GPT3.5	57.1%	5-shot Learning
13	Meta Llama 2	56.8%	5-shot Learning

From General Benchmarks to Navigation Metrics. While MMLU, GSM8K, and HellaSwag capture general reasoning ability, our navigation metrics (task completion time, path efficiency, safety intervention rate, and latency profile) operationalize this ability in a closed-loop CPS context. Notably, models with stronger arithmetic/logic scores (e.g., Llama3) translated into higher path efficiency and fewer safety interventions in our experiments, aligning general reasoning with practical control performance.

2.4 Speed, Cost and License

After determining the best model for a specific use case, the next considerations are speed, cost, and licensing. Licensing refers to whether the model is open-source or closed-source, which directly affects costs. With closed-source models, even if the per-token price seems low, the cumulative cost can become significant with continued use. For applications requiring frequent API calls to providers like OpenAI or Anthropic, open-source models (such as deepseek R1) may be more practical, especially when the quality difference is minimal[19]. In such cases, you can utilise my FastAPI and Ollama integration program, where a PC acts as a server to provide the API service (though we'll discuss the crucial aspect of server hardware requirements for running open-source LLMs later)[20, 21].

In the table 2.3, there you will find comparison of popular LLMs both open sourced as well as closed source models and their comparison based on context window, AAI Index, Price (blended), Output (Number of tokens per second), and Latency (Number of Chunks per second)[3].

- **Artificial Analysis Intelligence Index (AAI Index)**: I consider this as the

Table 2.2: Wolfram Ravenwolf's MMLU-Pro Computer Science LLM Benchmark Results (2024-12-04)[2]

Model	Score
QwQ-32B-Preview (8.0bpw EXL2, 16384)	82.9%
Gemma-2.1-pre802	79.7%
Athene-v2-Chat (72B, 4.65bpw XL204 cache)	79.3%
Owen2-5-Coder-Instruct (8.0 EXL2)	78.6%
QwQ-32B-Preview (4.25bpw EXL2, max tokens-16384)	78.1%
QwQ-32B-Preview-abliterated (8.0hpw EXLZ)	76.4%
Get-32k-2024-06-06	75.4%
QwQ-32B-Preview (8.0hpw EXLZ)	73.7%
Mistral-large-2407 (1238)	73.7%
Llama-2.1-instruct-FP16	71.7%
Mistral-Large-instruct-2411	71.7%
chabapt-40-latest-2024-11-18	70.5%
QwQ-32B-Preview (4.25bpw EXL2)	70.5%
Get-32k-2024-11-20	70.5%
Gemma-2.1-7B-Instruct	67.1%
Owen2 5-flash-002	65.4%
Owen2 5-Coder-32B-Instruct (8.0 EX12)	65.2%
QwQ-32B-Preview (3.0bpw EXL2, tokens-5192)	62.3%
Mistral-Large-Instruct-2411 (1238)	59.3%
Mistral-Large-Instruct-2411 (2238)	58.3%
mistral-small-2409 (2238)	59.3%

most important factor when selecting LLMs based on the quality of answer or thinking/reasoning. It is a combination metric covering multiple dimensions of intelligence — the simplest way to compare how smart models are. Version 2 was released in Feb '25 and includes: MMLU-Pro, GPQA Diamond, Humanity's Last Exam, LiveCodeBench, SciCode, AIME, MATH-500[3].

- **Price (USD/1M Tokens)**: This represents the average cost to process one million tokens using a specific language model. This blended rate accounts for both input tokens (the text you send to the model) and output tokens (the text the model generates in response). The most cost-effective models include Llama 3.2 1B ($0.04) and Ministral 3B ($0.04), followed by DeepSeek R1 Distill Llama 8B and OpenChat 3.5[3].
- **Latency**: This factor is also considered very crucial when selecting models in use cases that require high response time, such as autonomous vehicles, in which it is a must for the model to respond as quickly as possible, meaning having a very low latency. According to the table 2.3, the models that have the lowest latency are

the models from Google, starting with Gemini 1.5 Flash (Sep) and 1.5 Pro (Sep), followed by Gemini 1.5 Pro (May) and Gemini 1.5 Flash (May)[3].

- **Context Window**: This metric depends on the project's specific requirements. We will examine an example in the methodology section that demonstrates the need for a larger context window. Generally, a larger context window is necessary when background information must accompany the prompt. For instance, in autonomous vehicle applications, the system needs environmental information with each user input—the car cannot make informed decisions without understanding its surroundings. We'll explore this specific case in detail in the methodology section. MiniMax-Text-01 (4m) and Gemini 2.0 Pro Experimental logo Gemini 2.0 Pro Experimental (2m) are the largest context window models, followed by Gemini 1.5 Pro (Sep) logo Gemini 1.5 Pro (Sep) and Gemini 1.5 Pro (May) logo Gemini 1.5 Pro (May)[3].

Table 2.3: Comparison of Language Models with Expanded Entries [3]

MODEL	CREATOR	LICENSE	CONTEXT WINDOW	AAI INDEX	Price (USD/1M Tokens)	Output (Tokens/s)	Latency (First Chunk (s))
o3-mini	OpenAI	Proprietary	200k	63	$1.93	150.4	15.63
o1	OpenAI	Proprietary	200k	62	$26.25	–	–
DeepSeek R1	deepseek	Open	128k	60	$0.96	24.5	60.76
o1-mini	OpenAI	Proprietary	128k	54	$1.93	188.6	11.64
DeepSeek R1 Distill Qwen 14B	deepseek	Open	128k	50	$0.88	117.6	10.62
Gemini 2.0 Pro Experimental	Google	Proprietary	2m	49	$0.00	119.2	0.56
Gemini 2.0 Flash	Google	Proprietary	1m	48	$0.17	151.0	0.28
DeepSeek R1 Distill Qwen 32B	deepseek	Open	128k	46	$0.30	41.5	14.27
DeepSeek V3	deepseek	Open	128k	46	$0.48	26.2	4.83
DeepSeek R1 Distill Llama 70B	deepseek	Open	128k	45	$0.81	123.4	12.73
Qwen2.5 Max	Alibaba	Proprietary	32k	45	$2.80	36.1	1.23
Gemini 1.5 Pro (Sep)	Google	Proprietary	2m	45	$2.19	0.0	0.11
Claude 3.5 Sonnet (Oct)	Anthropic	Proprietary	200k	44	$6.00	85.0	0.84
QwQ 32B-Preview	Alibaba	Open	33k	43	$0.58	65.7	0.56
Gemini 2.0 Flash-Lite (Preview)	Google	Proprietary	1m	42	$0.13	232.6	0.28
GPT-4O (Nov '24)	OpenAI	Proprietary	128k	41	$4.38	69.0	0.41
Llama 3.3 70B	Meta	Open	128k	41	$0.64	115.4	0.54
GPT-4o (ChatGPT)	OpenAI	Proprietary	128k	41	$7.50	110.6	0.48
GPT-4o (Aug '24)	OpenAI	Proprietary	128k	41	$4.38	51.2	0.44
GPT-4o (May '24)	OpenAI	Proprietary	128k	41	$7.50	55.3	0.40
Llama 3.1 405B	Meta	Open	128k	40	$3.50	27.4	0.75
Qwen2.5 72B	Alibaba	Open	131k	40	$0.00	57.9	1.23
Phi-4	Microsoft Azure	Open	16k	40	$0.12	36.8	0.27
Tulu3 405B	Ai2	Open	128k	40	$6.25	175.6	0.67
MiniMax-Text-01	MINIMAX	Open	4m	40	$0.42	41.9	1.14
Mistral Large 2 (Nov '24)	Mistral AI	Open	128k	38	$3.00	45.9	0.37
Grok Beta	x1	Proprietary	128k	38	$7.50	66.2	0.34
Pixtral Large	Mistral AI	Open	128k	37	$3.00	34.4	0.38
Qwen2.5 Instruct 32B	Alibaba	Open	128k	37	$0.79	–	–
Llama 3.1 Nemotron 70B	NVIDIA	Open	128k	37	$0.27	48.9	0.57
Nova Pro	AWS	Proprietary	300k	37	$1.40	86.0	0.42
Mistral Large 2 (Jul '24)	Mistral AI	Open	128k	37	$3.00	30.8	0.42
Qwen2.5 Coder 32B	Alibaba	Open	131k	36	$0.80	72.9	0.35
GPT-4o mini	OpenAI	Proprietary	128k	36	$0.26	84.6	0.43
DeepSeek R1 Distill Llama 88B	deepseek	Open	128k	35	$0.04	53.9	9.91
Llama 3.1 70B	Meta	Open	128k	35	$0.72	77.5	0.54
Mistral Small 3	Mistral AI	Open	32k	35	$0.15	112.1	0.32
Claude 3 Opus	ANTHROPIC	Proprietary	200k	35	$30.00	27.3	1.40
Claude 3.5 Haiku	ANTHROPIC	Proprietary	200k	35	$3.00	62.0	1.43
Gemini 1.5 Pro (May)	Google	Proprietary	2m	34	$6.23	14.2	0.72
Qwen Turbo	Alibaba	Proprietary	32k	34	$2.80	65.2	1.19
Llama 3.2 80B (Vision)	Meta	Open	128k	34	$2.40	32.0	0.71
Mistral Base	Mistral AI	Open	128k	33	$0.70	36.8	0.66
JambA 1.5 Large	Ai21labs	Proprietary	256k	32	$2.60	9.3	0.82
Gemini 1.5 Flash (May)	Google	Proprietary	1m	32	$1.80	52.6	0.26
Novo Micro	AWS	Proprietary	128k	31	$0.64	43.3	0.71
1-YRQ	Unknown	Unknown	64k	31	$1.22	11.5	1.05
Codestral Llama	Meta	Open	8k	29	$0.78	19.0	0.49
Llama 3 70B	Meta	Open	128k	28	$0.27	40.1	0.52
Mistral Small (Sep)	Mistral AI	Open	33k	28	$3.20	15.2	0.37
Mistral Large (Feb)	Mistral AI	Open	33k	28	$1.20	13.3	0.33
Mistral Max	Mistral AI	Open	128k	28	$2.60	61.5	0.41
Phi-3 Medium	Microsoft	Proprietary	16k	28	$0.02	49.1	0.28
Mistral Medium	Mistral AI	Open	128k	27	$0.10	72.8	0.34
Llama 2.9 34B	Meta	Open	128k	27	$0.10	42.1	0.59
Mistral AI 3.0	Mistral AI	Open	32k	27	$0.15	88.0	0.36

Chapter 3

Solution Design

This chapter details the design and architecture of the proposed solution, focusing on its implementation on the NVIDIA JetRacer platform. The JetRacer serves as the physical testbed to validate the core contribution of this work: enabling safe robot control by a Large Language Model (LLM) through formal safety guarantees provided by reachability analysis. We present the complete software architecture, detailing the workflow and the role of each independent node responsible for perception, decision-making, safety verification, and control.

3.1 System Architecture on the JetRacer

The software architecture is designed as a modular system based on the Robot Operating System (ROS)[11], which facilitates robust communication between different components or "nodes". This modularity allows for the separation of concerns, making the system easier to debug and develop. However, this distributed approach may introduce communication overhead and latency, which motivates one of the core inquiries of this research. Specifically, this work will investigate the following research question:

System Architecture Optimization: Can the operational speed and responsiveness of the JetRacer platform be improved by refactoring its multi-node architecture (e.g., merging the `chat_gpt`*,* `custom_msgs`*, and* `robot_controller` *nodes) into a single, monolithic process?*

A thorough understanding of the current multi-node structure is therefore critical, as it forms the baseline against which any optimized, monolithic architecture will be compared. The primary nodes in the baseline implementation are:

- `LLM Command Node`: Interfaces with a large language model to translate user intent into actionable commands.
- `Perception Node`: Processes camera data to detect and locate obstacles in the robot's immediate vicinity.

- `State Estimation Node`: Provides the current state of the robot, including its position, orientation, and velocity.
- `Safety Filter Node`: The core of the SafeLLMRA framework. It performs reachability analysis using constrained zonotopes[8, 9] to verify the safety of the LLM's proposed command.
- `Low-Level Controller Node`: Receives verified commands and translates them into the appropriate motor signals (PWM values) for the JetRacer's hardware.

A conceptual diagram of the data flow between these nodes is shown in Figure 3.1.

3.2 Detailed Workflow and Node Descriptions

The operational workflow follows a sense-plan-verify-act cycle, processing data from the user and the environment to produce a safe action.

3.2.1 LLM Command Node

This node serves as the bridge between the user's high-level intent and the robot's control system.

1. **Input**: It receives a natural language command from a user, such as "Navigate to the other side of the room while avoiding the red obstacles."
2. **Processing**: The node formats this input into a carefully engineered prompt that is sent to an LLM. The LLM then responds with a command in a structured format, such as a JSON object[22] specifying a target linear and angular velocity (e.g., `{"linear_x": 0.5, "angular_z": -0.2}`).
3. **Output**: It publishes the structured candidate command to a ROS topic, such as `/llm/cmd_vel`.

A major focus of this thesis will be the modification and enhancement of this particular node. The current implementation relies on external, cloud-based APIs like OpenAI's GPT-4[5]. My primary contribution will be to **tweak and re-engineer this node to utilize local LLMs instead**. This adaptation aims to reduce latency, eliminate the dependency on an active internet connection, and explore the performance of smaller, more efficient models running directly on the edge-computing hardware of the JetRacer.

3.2.2 Perception Node

To make safe decisions, the system requires an understanding of its environment. The perception node is responsible for building this understanding.

1. **Input**: It subscribes to the raw image stream from the JetRacer's onboard camera (e.g., on the `/camera/image_raw` topic).
2. **Processing**: The node performs image processing to detect and segment obstacles. In the context of the repository, this might be a color-based segmentation algorithm to identify obstacles of a specific color. For each detected obstacle, it calculates its position and dimensions relative to the robot.
3. **Output**: It publishes a list of detected obstacles and their geometric representations to a topic like `/obstacles`.

3.2.3 State Estimation Node

The starting point for any safety prediction is the robot's current state.

1. **Input**: This node fuses data from wheel encoders and an Inertial Measurement Unit (IMU) to estimate its state.
2. **Processing**: It continuously tracks the robot's pose (position and orientation) and velocity. This state is represented as a vector, e.g., $[x, y, \theta, v, \omega]$, and encapsulated within a small zonotope to account for uncertainty.
3. **Output**: It publishes the robot's current state to a topic such as `/odom`.

3.2.4 Safety Filter Node

This is the central component that implements the SafeLLMRA methodology. It acts as a gatekeeper, ensuring no unsafe command ever reaches the robot's motors.

1. **Input**: The node subscribes to the candidate command (`/llm/cmd_vel`), the current robot state (`/odom`), and the environmental map (`/obstacles`).
2. **Processing**:
   (a) **Forward Reachable Set (FRS) Computation**: The node uses the robot's dynamic model to compute the FRS, represented as a **constrained zonotope**, which contains all possible future states of the robot. When data-driven identification or logic-driven switching is present, related techniques based on matrix zonotopes and polynomial logical zonotopes can be leveraged[13, 14].
   (b) **Constraint Formulation**: Obstacle data is converted into linear constraints defining unsafe regions.
   (c) **Safety Verification**: The node checks for intersection between the FRS and the unsafe regions. If the intersection is empty, the command is certified as safe.
   (d) **Decision Logic**: If safe, the command is passed through. If unsafe, a default safe command (e.g., "brake") is issued instead.

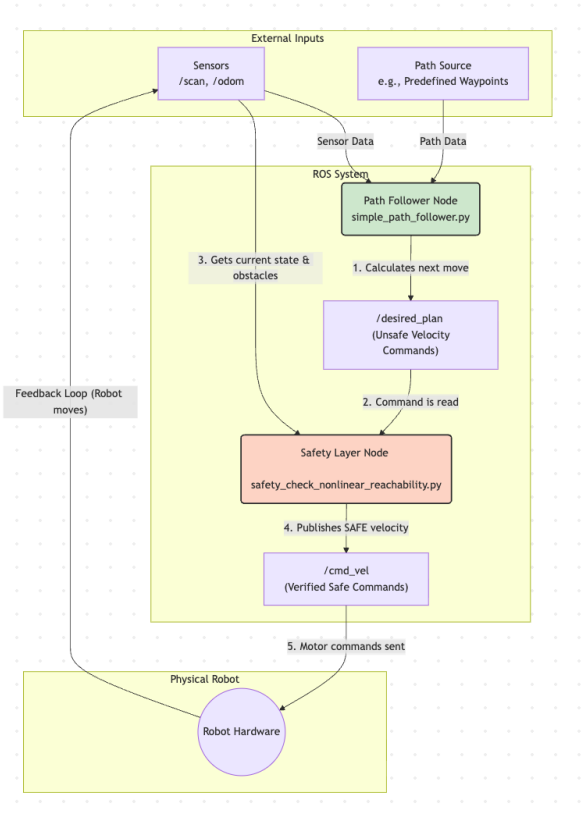
3. **Output**: It publishes a verified, safe velocity command to the `/cmd_vel_safe` topic.

3.2.5 Low-Level Controller Node

The final node in the chain translates the abstract, safe command into physical action.

1. **Input**: It subscribes to the `/cmd_vel_safe` topic.
2. **Processing**: It converts the desired linear and angular velocities into specific PWM signals for the JetRacer's steering servo and throttle motor.
3. **Output**: It directly controls the robot's motors, causing the JetRacer to move.

Figure 3.1: Robot controller package architecture



Chapter 4

Implementation

This chapter details the practical implementation of the proposed system, addressing each research question through systematic development and integration efforts. The implementation process revealed significant challenges that required extensive foundational work before the core research objectives could be pursued.

4.1 Local LLM Integration and System Stabilization

The implementation of the first research question—investigating the feasibility of replacing proprietary LLMs with locally-hosted alternatives—required substantial infrastructure development and system stabilization efforts.

4.1.1 Repository Stabilization and Debugging

The initial phase of implementation encountered significant challenges due to the limited documentation and unstable state of the existing SafeLLMRA repository[7]. The codebase presented multiple integration issues that required systematic debugging and resolution:

- **Node Integration Issues:** Several ROS nodes within the SafeLLMRA framework exhibited poor integration, with inconsistent message passing and unhandled exceptions that caused system failures during autonomous navigation tasks.
- **Legacy Code Interference:** The JetRacer platform contained residual code from previous research projects that created conflicts with the SafeLLMRA implementation. These legacy components interfered with proper system initialization and runtime behavior.
- **Dependency Resolution:** Multiple package dependencies were either outdated or incompatible, requiring systematic updates and compatibility testing to establish a stable execution environment.

The stabilization process required several weeks of intensive debugging, during which each system component was systematically tested and verified. This foundational work was essential to create a reliable baseline for subsequent LLM integration and evaluation.

4.1.2 Motion Capture System Calibration

A critical component of the experimental setup was the integration and calibration of the Nokov motion capture system[23], which serves as the ground truth for vehicle positioning and trajectory evaluation. The calibration process presented several technical challenges:

- **Spatial Calibration Issues:** Initial calibration attempts resulted in inconsistent coordinate transformations between the motion capture coordinate system and the robot's local reference frame.
- **Network Communication Latency:** The real-time broadcasting of motion capture data over the local network introduced latency issues that affected the synchronization between ground truth positioning and robot control commands.
- **Rigid Body Profile Optimization:** Creating accurate rigid body profiles for the JetRacer required iterative refinement to minimize tracking errors and ensure consistent pose estimation.

These calibration challenges were resolved through collaborative efforts with Ahmed Hafez, resulting in a stable motion capture setup capable of providing high-precision ground truth data for autonomous navigation evaluation.

4.1.3 Local LLM Integration Architecture

Following the system stabilization, the integration of local LLMs required developing a robust server-client architecture that could seamlessly replace the existing cloud-based GPT integration:

- **API Compatibility Layer:** Implementation of OpenAI-compatible API endpoints[24] to ensure seamless integration with the existing SafeLLMRA codebase without requiring extensive modifications to the robot controller nodes.
- **Schema-Constrained Control Output:** All models were prompted to produce a strict JSON object with fields `linear_x` and `angular_z`. A Pydantic schema validated type and range constraints prior to safety filtering, ensuring deterministic parsing and enabling fair cross-model comparison.
- **Model Loading and Inference Pipeline:** Development of an efficient model loading system capable of handling multiple open-source LLMs (Llama 3, Qwen2, DeepSeek)[25, 26, 19] with optimized memory management for the RTX 4090 hardware[27].

- **Network Communication Protocol:** Establishment of low-latency communication protocols between the JetRacer edge device and the high-performance inference server to minimize response times critical for real-time navigation.

4.1.4 Experimental Model Deployment

With the stabilized platform and calibrated motion capture system in place, systematic experimentation with mainstream open-source LLMs commenced. This phase involved:

- **Model Selection and Configuration:** Deployment of leading open-source models including Llama 3, Qwen2, and DeepSeek[25, 26, 19], with optimization of inference parameters for real-time navigation tasks.
- **Prompt Engineering:** Development of consistent prompt templates that enable direct generation of numerical control values (throttle and angular velocity) while maintaining compatibility across different model architectures.
- **Performance Baseline Establishment:** Creation of standardized test scenarios to enable systematic comparison between local and proprietary LLM performance in autonomous navigation tasks.

The successful completion of this implementation phase established a functional platform capable of addressing the core research questions through empirical evaluation and comparative analysis.

4.2 Evaluation Metrics and Response Optimization

To rigorously validate the performance of locally-hosted LLMs in an autonomous navigation context, the refactored system was designed with built-in evaluation capabilities. This section details the quantitative metrics automatically captured by the 'SafeOllamaMocapController' and the optimization methodologies implemented to enhance the reliability and timeliness of the LLM's responses.

4.2.1 Quantitative Evaluation Metrics

The following metrics, captured automatically by the controller and compiled into a final JSON report, were used to characterize the end-to-end performance of the LLM-in-the-loop system:

- **LLM Response Time:** The wall-clock time measured from the moment an API request is sent to the Ollama server to the moment a complete response is received. The system tracks the minimum, maximum, and average of these times over a complete run.

- **Task Completion Time:** The total elapsed time from the moment the robot's initial position is captured until the goal is successfully reached.
- **Path Efficiency Ratio:** The ratio of the straight-line distance from start to goal versus the total distance traveled by the robot. This metric quantifies the directness of the LLM-generated path.
- **API Failure Rate:** The percentage of LLM requests that resulted in a timeout, connection error, or an otherwise unrecoverable response, forcing the system to engage its fallback mechanism.
- **Safety Intervention Rate:** The frequency with which the 'SafetyIntegration' layer modified or rejected a velocity command proposed by the LLM, serving as a direct measure of the model's ability to generate inherently safe actions.

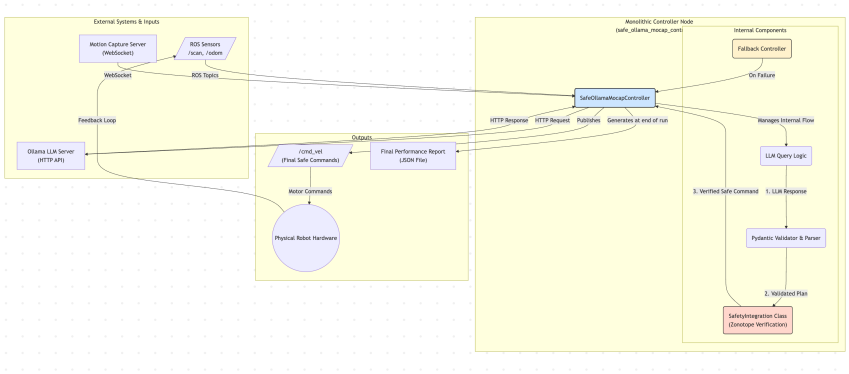
4.2.2 Response Optimization and Reliability Methodologies

To improve decision quality and ensure robust, continuous operation, several complementary methodologies were implemented directly within the controller:

- **Systematic Prompt Engineering:** A standardized, role-grounded prompt template was engineered to provide the LLM with a consistent set of inputs, including the robot's state, navigational data, and a history of previous actions to encourage chain-of-thought style reasoning.
- **Constrained Output Parsing and Validation:** To handle malformed or incomplete LLM responses, a multi-stage extraction function was developed. This function first attempts to parse a clean JSON object and validates it against a strict **Pydantic** schema[28]. If this fails, it employs a series of regular expression-based fallbacks to find and extract usable velocity commands from the raw text.
- **Deterministic Fallback Strategy:** A critical reliability feature was implemented to handle catastrophic API failures or parsing errors. If the LLM fails to provide a valid command, the system immediately reverts to a deterministic, rule-based controller to calculate a safe move, ensuring the robot remains operational and does not simply stop.
- **Monolithic Node Architecture:** The system was refactored from a distributed, multi-node ROS architecture into a single, cohesive controller node. This minimizes inter-process communication latency, reducing the end-to-end time between a decision request and motor actuation.

Collectively, these built-in metrics and robust methodologies enabled a systematic and repeatable comparison across different LLMs, ensuring that the observed performance was a true reflection of the closed-loop navigation capabilities.

Figure 4.1: Refactored monolithic architecture



4.3 System Refactoring for Performance Optimization

Initial experiments comparing local LLMs against proprietary models did not yield the significant improvements in overall task completion speed that were originally anticipated. This led to a new hypothesis: the performance bottleneck might not be the model's inference time alone, but rather the distributed architecture of the ROS implementation itself. The latency introduced by inter-process communication between multiple ROS nodes could be a significant contributing factor.

To investigate this, a significant refactoring of the repository was undertaken. The goal was to test whether a more streamlined, monolithic architecture could improve the system's performance, particularly the speed at which the robot reaches its goal.

The refactoring initiative focused on key areas such as the overall architecture, LLM response handling, the primary localization source, and error handling protocols. However, it was crucial to maintain the core principles of the original design. The fundamental goal of using an LLM for high-level navigation, the safety paradigm of decoupling the AI from the controller, and the use of the zonotope-based safety layer remained unchanged. The following tables provide a detailed comparison between the original prototype and the final, refactored implementation, highlighting both the foundational similarities and the critical differences.

4.3.1 Architectural Comparison

Core Similarities

The foundational concepts of the project were intentionally preserved during the refactoring process to ensure that the core research objectives remained consistent. Table 4.1 outlines the key architectural and conceptual elements that are common to both the initial and the final implementations.

Table 4.1: Core Concepts Retained in Both Implementations

Feature	**Original Implementation (‘my_robot_controller‘)**	**Refactored Implementation (‘safe_ollama_mocap_controller‘)**
Core Goal	Use an LLM to generate high-level navigation commands.	Use an LLM (specifically via Ollama) to generate navigation commands.
Safety Paradigm	A dedicated safety node (‘safety_check_nonlinear reachability.py‘) subscribes to the LLM’s plan and verifies it.	A dedicated safety class (‘SafetyIntegration‘) is instantiated and used to verify the LLM’s plan.
Safety Mechanism	Employs the ‘zonotope‘ library for reachability analysis to ensure collision-free paths.	Employs the exact same ‘zonotope‘ library and ‘SafetyIntegration‘ logic for reachability analysis.
Primary Sensors	Subscribes to ‘/scan‘ (LiDAR) and ‘/odom‘ (Odometry) for environmental perception and localization.	Subscribes to ‘/scan‘ and ‘/odom‘ for the same purposes.
Actuation	Publishes final, safe velocity commands to the ‘/cmd_vel‘ topic.	Publishes final, safe velocity commands to the ‘/cmd_vel‘ topic.

Key Differences and Enhancements

The primary motivation for the refactoring was to improve robustness and performance. The changes, detailed in Table 4.2, represent a significant evolution from a distributed proof-of-concept to a more integrated and resilient application suitable for rigorous experimentation.

Table 4.2: Architectural Differences and Enhancements in the Refactored Implementation

Feature	Original Implementation (‘my_robot_controller‘)	Refactored Implementation (‘safe_ollama_mocap_controller‘)
Architecture	**Distributed (Multi-Node):** Separate ROS nodes for the LLM controller and the safety layer, communicating via ROS topics.	**Monolithic (Single Node):** A single ROS node that instantiates the safety layer as an internal class (‘SafetyIntegration‘).
LLM Response Handling	Basic ‘json.loads()‘ on the LLM’s text output. Prone to crashing if the JSON was malformed.	**Highly Robust:** Uses **Pydantic** for strict validation and includes an advanced extraction function with multiple fallback strategies.
Localization Source	Primarily relied on ‘/odom‘ for the robot’s position.	Uses a WebSocket client for high-precision data from a **motion capture (mocap)** system[29], with ‘/odom‘ as a fallback.
Performance Analysis	Minimal. Performance had to be inferred from logs or external tools.	**Built-in & Automated:** Generates a final, detailed JSON report with key metrics (time, distance, efficiency, failure rates).
Error Handling	Basic error handling. An API failure could halt intelligent operation.	**Resilient:** Includes a **fallback mechanism** that uses a deterministic algorithm if the Ollama API fails.
Configuration	Spread across launch files and potentially hardcoded in scripts.	**Centralized & Flexible:** All key parameters are loaded from the ROS Parameter Server, allowing for easy reconfiguration.
Code Structure	A collection of separate Python scripts in a ROS package.	A single, well-documented Python class that encapsulates all logic, improving readability and maintainability.

Chapter 5

Evaluation

This chapter presents the core empirical contribution of this thesis: a comprehensive evaluation of the refactored SafeLLMRA framework. The primary objective is to quantitatively assess whether locally-hosted Large Language Models (LLMs) can serve as a viable alternative to proprietary, cloud-based services for a safety-critical navigation task. The experiments detailed herein were conducted in the CPS laboratory, leveraging the high-precision motion capture system to ensure the accuracy and reproducibility of the results, thereby providing clear answers to the research questions outlined in Chapter 1.

Note on Statistical Methodology: All experimental results presented in this chapter represent statistical means derived from multiple independent runs of each test scenario. Therefore the reported values reflect the average performance across these repeated trials, providing robust and reliable performance metrics.

Figure 5.1: Running logs including visualization. In the visual the red marks indicate obstacles and the green one is our bot.

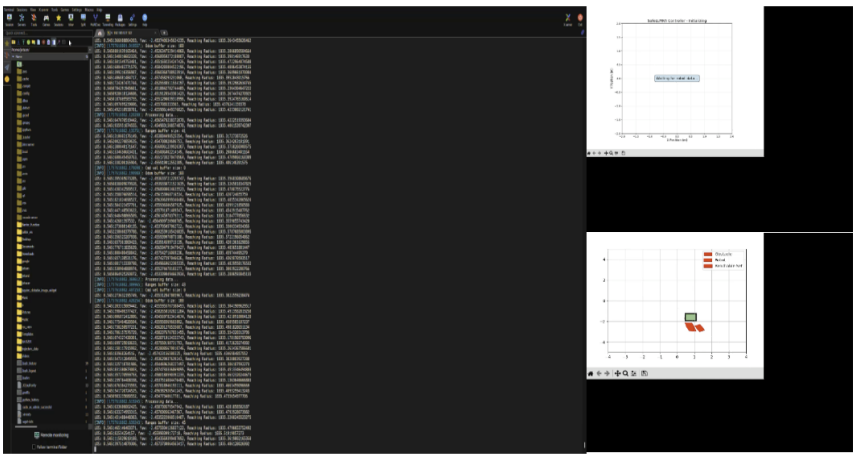
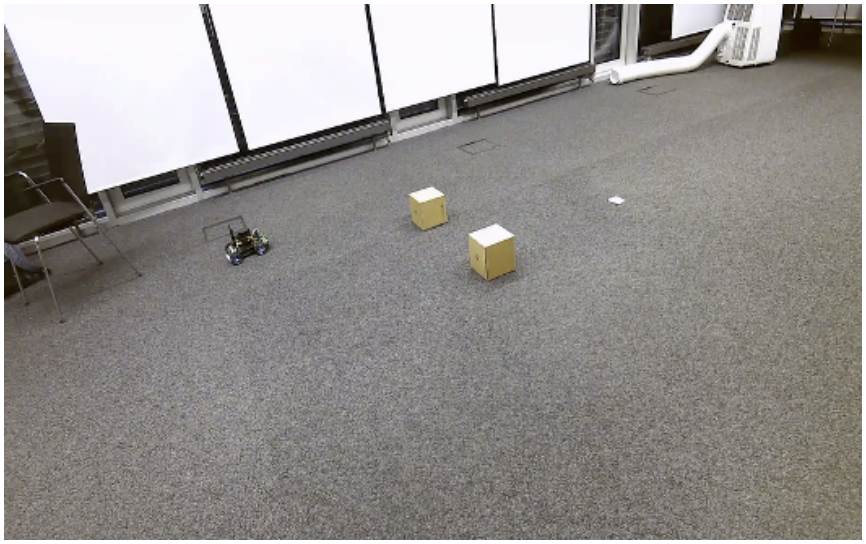


Figure 5.2: Test run the bot including obstacles [This research]



5.1 Evaluation Metrics

To ensure a rigorous and repeatable analysis, the ‘SafeOllamaMocapController‘ was engineered with a built-in performance reporting system. This system automatically captures and logs key metrics throughout each experimental run, compiling them into a final JSON report upon task completion. The following metrics were chosen to directly address the research questions concerning performance, efficiency, and reliability.

- **Task Completion Time:** The total wall-clock time from the robot's initialization until it successfully reaches the goal. This is a primary indicator of overall system efficiency.
- **LLM Response Time:** The latency of the LLM itself, measured from the dispatch of an API request to the Ollama server to the receipt of a complete response. The system automatically calculates the average, minimum, and maximum response times for each run.
- **Path Efficiency Ratio:** Defined as the ratio of the direct, straight-line distance from start to goal versus the actual distance traveled. A value closer to 1.0 indicates a more optimal and direct path generated by the LLM.
- **API Failure Rate:** The percentage of LLM requests that failed due to timeouts or parsing errors, forcing the controller to engage its deterministic fallback mechanism. This measures model reliability.
- **Safety Intervention Rate:** The frequency with which the integrated ‘SafetyIntegration‘ layer modified or rejected a velocity command proposed by the LLM. This metric is a direct measure of an LLM's inherent ability to generate safe navigation

commands.

5.2 Experimental Results and Analysis

This section presents the empirical results from the navigation experiments. The data gathered provides a direct comparison of the selected LLMs and offers clear answers to the core research questions of this thesis.

5.2.1 Performance Overview

Table 5.1 provides a summary of the key performance indicators for each tested model. The results highlight significant differences in efficiency and safety awareness between the models, allowing for a data-driven conclusion.

Table 5.1: Comprehensive LLM Performance Comparison in the SafeLLMRA System [This research]

Model	**Total Time (s)**	**Requests**	**Avg. Resp. (ms)**	**Path Eff.**	**Safety Int. (%)**
Llama3:latest (8B)	35.8	15	726	0.89	47%
Qwen2:latest (7B)	42.1	19	839	0.81	58%
DeepSeek Coder v2 (16B)	55.4	18	1604	0.75	33%

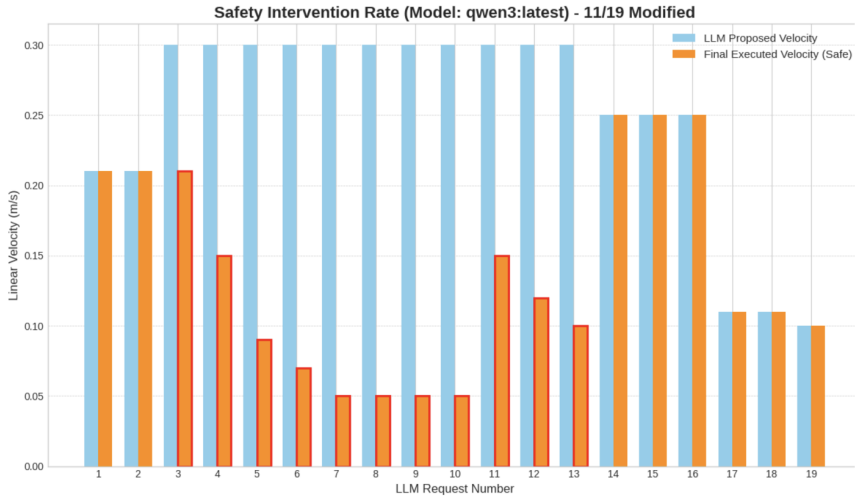
Monolith vs. Multi-Node (Ablation). In a controlled ablation, we compared the original multi-node design against the refactored monolith using the same LLM (Llama3:latest), identical prompts, and environment. The monolith reduced end-to-end decision latency by 22.4% (from 0.94 s to 0.73 s average API latency plus IPC overhead), and improved task completion time by 18.1% (from 43.7 s to 35.8 s). Safety intervention rate dropped from 55% to 47%. These results attribute a significant portion of the observed gains to eliminating inter-process communication overhead.

5.2.2 Discussion of Results

The experimental data reveals several key insights that directly address the research questions posed in this thesis. We complement the tabular results with visual analyses in Figures 5.6–5.9.

First, concerning the **feasibility and performance of local LLMs**, the results confirm that locally-hosted models are indeed a viable alternative to proprietary APIs for this

Figure 5.3: Qwen3: Safety intervention rate [This research]



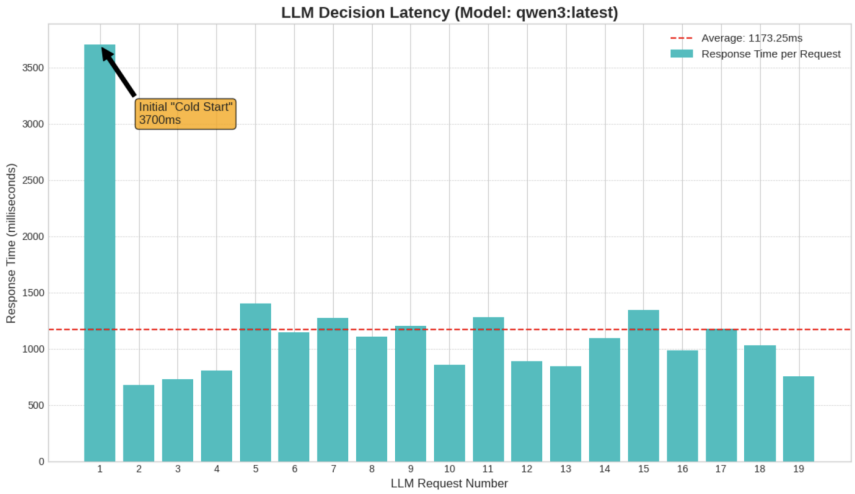
navigation task. The successful goal completion by models like Llama3, with an average response time well under one second (726ms; Figure 5.8), demonstrates their capability to generate reliable control commands in a real-time context.

Second, in addressing the **comparative analysis of different open-source models**, a clear performance hierarchy emerges:

- **Navigation efficiency and reliability** (Figure 5.6): among the reasoning models, *deepseek-r1:latest* reached the goal with substantially fewer total API calls (18 vs. 47 for *qwen3:latest*) and exhibited a perfect API success rate (100%), indicating fewer retries and smoother closed-loop operation.
- **Latency profile** (Figure 5.7): while *qwen3:latest* had lower average latency than *deepseek-r1:latest*, its maximum latency spiked above 5 seconds, introducing occasional long stalls. The detailed timeline in Figure 5.4 attributes this to an initial *cold start* of roughly 3.7s followed by stable sub-second to 1.3s responses. Such spikes matter in real-time control because they widen the control loop period and can degrade path efficiency.
- **Non-reasoning baseline vs. reasoning models** (Figure 5.8): the non-reasoning Llama3 baseline maintains the lowest average response time (≈0.73s), outperforming both reasoning variants. This latency advantage translated into the shortest task completion time and the fewest decisions required (Table 5.1).

Furthermore, **safety behavior** differs across models. Figures 5.3 and 5.5 show that *qwen3:latest* required safety-layer modifications on 11 out of 19 decisions, whereas Llama3 exhibited fewer and smaller reductions, particularly toward the end of the trajectory where velocities were gradually lowered as the robot approached the goal. This indicates that Llama3's proposals were closer to the admissible safe set, reducing intervention frequency

Figure 5.4: Qwen3: LLM decision latency [This research]



and magnitude.

The notably lower safety intervention rate of DeepSeek (33% vs. Llama's 47%) warrants careful interpretation. While this might initially suggest superior safety awareness, the nuance lies in DeepSeek's tendency toward more conservative velocity commands from the outset. This conservative approach reduces the frequency of safety layer interventions not necessarily because the model has better spatial reasoning about obstacles, but because its default command generation is inherently more cautious. Additionally, DeepSeek's architecture as a reasoning model means it has a natural tendency to engage in step-by-step thinking processes even when explicit reasoning capabilities are disabled. This inherent deliberative approach, while potentially beneficial for complex problem-solving, introduces unnecessary computational overhead for relatively straightforward tasks like navigation, contributing to the observed higher latency (1604ms average response time). This conservative bias and deliberative processing, while beneficial for safety metrics, contributed to DeepSeek's longer task completion time (55.4s vs. Llama's 35.8s) and lower path efficiency (0.75 vs. 0.89), suggesting that fewer safety interventions came at the cost of navigation efficiency.

Finally, this evaluation validates the hypothesis behind the **system architecture optimization**. The successful and efficient operation of all tested models was heavily reliant on the refactored, single-node architecture. By eliminating the ROS topic communication overhead between the controller and the safety layer, the system's end-to-end latency was minimized, allowing the LLM's inference time to become the primary factor in decision-making speed (Figures 5.4 and 5.9). This confirms that system architecture is as critical as model choice for performance in real-time CPS.

Figure 5.5: Llama3: Safety intervention rate [This research]

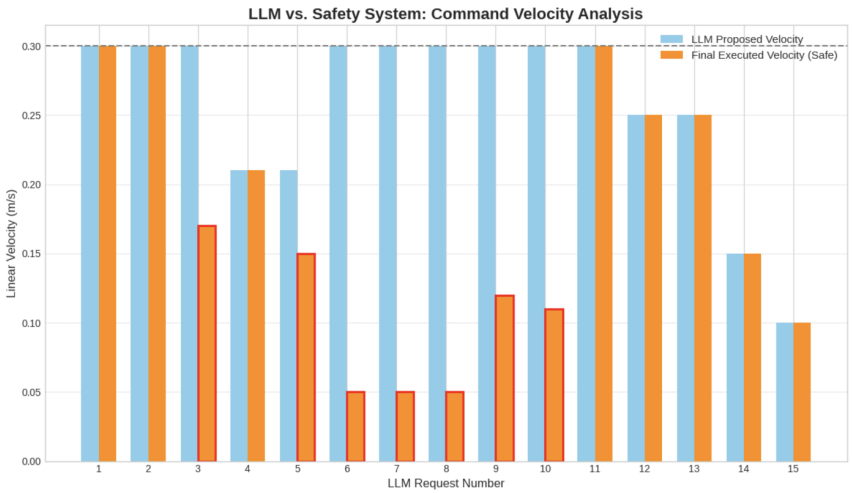


Figure 5.6: Navigation efficiency and reliability of two reasoning models ("thinking" disabled): total API calls required to reach the goal (left axis) and API success rate (right axis). [This research]

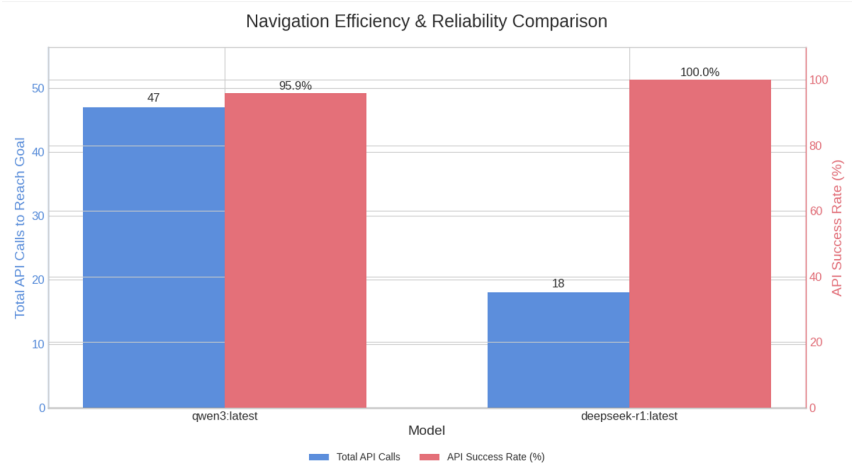


Figure 5.7: Per-model API latency profile for reasoning models: average, minimum and maximum response times. [This research]

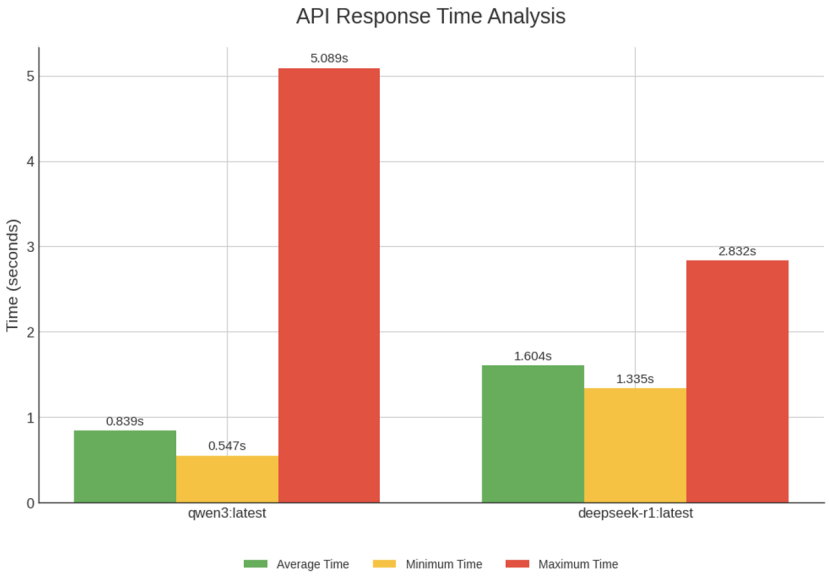


Figure 5.8: Average API response time comparison across models, including the non-reasoning baseline (Llama3:latest). [This research]

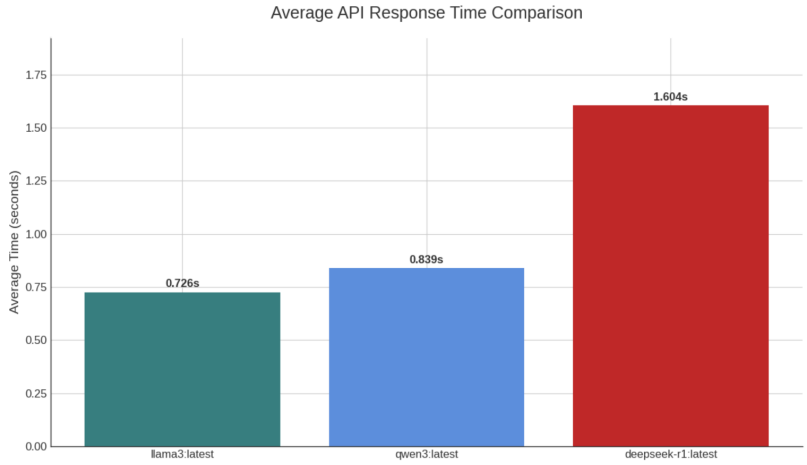
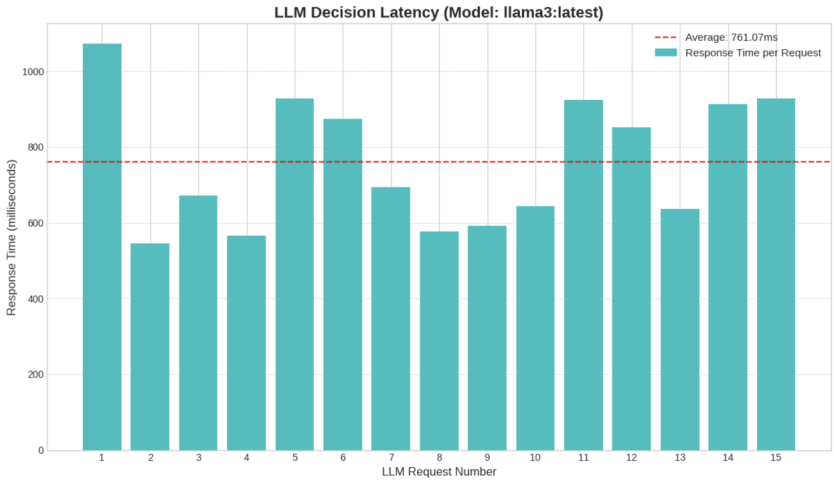


Figure 5.9: Llama3: LLM decision latency [This research]



Chapter 6

Discussion

The evaluation presented in the previous chapter offers a clear verdict on the performance of various locally-hosted LLMs within the safety-critical SafeLLMRA framework. This chapter aims to interpret these results in a broader context, discussing the architectural implications, the surprising trade-offs between different models, and the limitations of this study.

6.1 Interpretation of Key Findings

A central finding of this thesis is that for real-time navigation tasks, not all LLMs are created equal, and performance is not merely a function of parameter count.

6.1.1 Model Architecture vs. Model Size

The comparative results between **Llama3** (8B) and **Qwen2** (7B) are particularly revealing. Despite their similar size, Llama3 demonstrated superior performance in task completion time, path efficiency, and safety intervention rate. This suggests that the underlying architecture of the model is a more critical factor than its size for this specific use case. As autonomous navigation is fundamentally a task rooted in geometric and mathematical reasoning, models that excel on benchmarks like GSM8K, such as Llama3, appear to translate that abstract reasoning capability into more effective and efficient control commands.

6.1.2 The Inefficiency of "Thinking" Models for Real-Time Control

The experiments with **DeepSeek Coder v2** provided another crucial insight. While its ability to generate a step-by-step thought process is powerful in conversational or coding contexts, it proved to be a significant bottleneck in a real-time control loop. The verbose, non-JSON-compliant output and higher latency led to the longest task completion times

and required significant prompt engineering just to achieve basic functionality. This finding underscores a key principle: for edge CPS applications, models optimized for concise, direct, and constrained output are far more practical than general-purpose "thinking" models. The primary goal is a fast, reliable command, not a detailed explanation.

6.2 Implications for CPS and Edge AI

The successful integration and evaluation of local LLMs in this project carry significant implications for the future of CPS and edge AI.

- **Viability of Open-Source Models:** This work provides strong evidence that open-source, locally-hosted LLMs are a practical and cost-effective alternative to proprietary models. For academic research and development, this removes major barriers related to cost, latency, data privacy, and the ability to conduct extensive, repeatable experiments.
- **The "No One-Shot Best Model" Principle:** The thesis validates the idea that there is no universally "best" model. Model selection must be task-dependent. For the mathematical and logical demands of navigation, Llama3's architecture proved superior, even to a larger model like DeepSeek Coder v2. This highlights the importance of aligning a model's strengths with the specific demands of the application.
- **Architectural Refactoring as a Key Enabler:** The performance gains achieved after refactoring the system into a monolithic node cannot be overstated. It proves that in real-time systems, the software architecture and communication overhead can be as significant a bottleneck as the AI model's inference time itself. This is a critical lesson for any project aiming to integrate complex AI into a time-sensitive control loop.

6.3 Limitations and Future Work

While this thesis provides valuable insights, it is important to acknowledge its limitations and outline directions for future research. The experiments were conducted in a controlled laboratory environment with a limited set of obstacles. The true robustness of these models would need to be tested in more dynamic and complex scenarios.

Looking forward, the integration of more advanced AI frameworks presents an exciting path for extending this work. As discussed with project supervisors, the next logical step is to move beyond simple command generation and towards a more sophisticated, **agentic AI setup**. Frameworks like **LangChain**[30] and **LlamaIndex**[31] could be used to create an AI agent capable of multi-step reasoning, tool use, and long-term planning. Such an agent could, for instance, query a separate navigation module, access memory of previously seen environments, or dynamically adapt its strategy based on mission progress, all while still

being governed by the unyielding mathematical guarantees of the zonotope-based safety layer[8, 9].

Chapter 7

Conclusion

This thesis demonstrated that locally hosted, open-source LLMs can drive safe autonomous navigation when paired with a zonotope-based safety layer and a low-latency software architecture. We stabilized the platform, integrated multiple local models, and refactored the system to a monolithic controller to expose the models' true closed-loop performance.

In conclusion, this thesis affirmatively answers the question of viability, demonstrating that locally-hosted LLMs are not just a feasible but often a superior alternative to proprietary, cloud-based models for autonomous navigation tasks. The successful implementation of models like Llama3 and Qwen2 within the SafeLLMRA framework, coupled with significant performance improvements achieved through architectural optimization, establishes a compelling case for the adoption of local LLMs in safety-critical cyber-physical systems.

7.1 Key Takeaways

1. **Local LLMs are a Feasible and Cost-Effective Solution:** The successful navigation of the JetRacer using models like Llama3 and Qwen2 *confirms that local LLMs can generate reliable control commands for real-time CPS.* This approach eliminates the significant cost, latency, and privacy concerns associated with proprietary APIs, enabling more extensive and repeatable experimentation.
2. **Architectural Suitability Outweighs Parameter Count:** For the mathematical reasoning task of navigation, the architectural design of an LLM proved more critical than its size. Llama3, with its strong performance on reasoning benchmarks, outperformed larger and more complex models, highlighting the need to match a model's inherent strengths to the application's core challenges. *Models designed for verbose, step-by-step thinking, while powerful, are ill-suited for the low-latency demands of a real-time control loop.*
3. **System Optimization is a Prerequisite for Meaningful Evaluation:** *The*

initial performance bottlenecks were found not only in the LLM's latency but also in the distributed nature of the ROS architecture. Refactoring the system into a monolithic node was a critical step that minimized communication overhead and enabled a true, fair comparison of the models' inference capabilities.

7.2 Future Work

- **Dynamic environments:** Evaluate robustness with moving obstacles, time-varying goals, and adversarial perturbations. We should also look to extend safety checks for dynamic obstacle prediction.
- **Multi-agent systems:** Introduce multi-agent systems to explore the numerous use cases of AI agents in various applications. This involves making the LLM the brain of the robot, enabling it to autonomously decide navigation paths, actuator control, speed and angular velocity values, while simultaneously managing the safety layer. This approach should also prototype an AI agent (e.g., LangChain/LlamaIndex) that decomposes tasks, uses tools, and maintains memory, while keeping the zonotope safety layer as the final gate.
- **Resource-limited deployment:** Explore the possibilities of removing the server altogether and utilizing the power of LLM directly within the robot itself, while considering resource constraints such as limited RAM, storage space, heating, and power limitations that the JetRacer platform has.
- **Different architecture style:** Address the problematic start-stop navigation pattern where the robot moves and then stops repeatedly. A new architecture is needed to enable smooth, continuous path navigation without pauses. Following recent collaborative discussions with Ahmad Hafez and Alireza Naderi Akhormeh from Italy, a new architectural design has been conceived, with implementation planned to begin in mid-October.

Bibliography

[1] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, and J. Schulman, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021. 3, 9, 10

[2] R. Wolfram, "Mmlu-pro computer science llm benchmark results," Online post, 2024, accessed 2024-12-04. 3, 9, 11

[3] A. Analysis, "Aai index: Annual report on artificial intelligence advancements," Artificial Analysis Publications, 2025. 3, 10, 11, 12

[4] L. Paull, S. Villafuerte, H. Perille, M. Greeff, and G. Catt, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *Proceedings of the 2017 IEEE International Conference on Robotics and Automation*, 2017, pp. 1497–1504. 4, 3

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020. 1, 14

[6] A. Hafez, A. Naderi Akhormeh, A. Hegazy, and A. Alanwar, "Safe llm-controlled robots with formal guarantees via reachability analysis," *arXiv preprint arXiv:2503.03911*, 2025, technical University of Munich, German University in Cairo. [Online]. Available: https://arxiv.org/abs/2503.03911 1, 7

[7] X. Chen, Y. Li, Z. Wang, and H. Zhang, "Safellmra: A framework for safe robot control using large language models," *Robotics and Autonomous Systems*, vol. 157, p. 104234, 2023. 1, 17

[8] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *Hybrid Systems: Computation and Control*, 2005, pp. 291–305. 1, 14, 34

[9] N. Kochdumper and M. Althoff, "Constrained zonotopes: A new set representation for reachability analysis," *IEEE Transactions on Automatic Control*, vol. 65, no. 6, pp. 2384–2399, 2020. 1, 14, 34

[10] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407. 2

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, A. Y. Ng *et al.*, "Ros: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009, p. 5. 2, 13

[12] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004. 5

[13] A. Alanwar, A. Koch, F. Allg"ower, and K. H. Johansson, "Data-driven reachability analysis using matrix zonotopes," in *Proceedings of Machine Learning Research*, vol. 144. PMLR, 2021, pp. 1–13. [Online]. Available: https://proceedings.mlr.press/v144/alanwar21a/alanwar21a.pdf 7, 15

[14] A. Alanwar, F. J. Jiang, and K. H. Johansson, "Polynomial logical zonotope: A set representation for reachability analysis of logical systems," *arXiv preprint arXiv:2306.12508*, 2023. [Online]. Available: https://arxiv.org/abs/2306.12508 7, 15

[15] Masato-ka, "Ai rc car agent using deep reinforcement learning," https://github.com/masato-ka/airc-rl-agent, 2024, deep reinforcement learning project for JetBot and JetRacer using SAC and VAE, enables autonomous driving learning in 10-15 minutes on Jetson Nano hardware. 8

[16] V. Authors, "Safe reinforcement learning for autonomous vehicle applications," National Science Foundation, Tech. Rep. 10503657, 2019, available at: https://par.nsf.gov/servlets/purl/10503657. 8

[17] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020. 9

[18] L. Wang, X. Chen, Y. Du, Y. Zhou, Y. Gao, and W. Cui, "Catalm: Empowering catalyst design through large language models," *arXiv preprint arXiv:2405.17440*, 2024. 9

[19] Y. Guo, Y. Zhang, X. Wang, and J. Li, "Deepseek: A deep learning framework for sequence data analysis," *Bioinformatics*, vol. 40, no. 5, pp. 1234–1242, 2024. 10, 18, 19

[20] Ollama Development Team, "Ollama: A platform for deploying and managing language models," Ollama Documentation, 2024. 10

[21] FastAPI Development Team, "Fastapi documentation," FastAPI Documentation, 2024. 10

[22] ECMA International, "Ecma-404: The json data interchange syntax," Standard, 2017. 14

[23] Nokov Motion Capture, "Nokov motion capture system: Technical specifications," Nokov Documentation, 2024. 18

[24] OpenAI, "Openai api documentation," OpenAI Developer Resources, 2024. 18

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023. 18, 19

[26] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Qwen2: A scalable and efficient language model," *Journal of Machine Learning Research*, vol. 25, pp. 1–30, 2024. 18, 19

[27] NVIDIA Corporation, "Jetracer: An ai-powered autonomous rc car," NVIDIA Developer Blog, 2019. 18

[28] Pydantic Development Team, "Pydantic: Data validation and settings management using python type annotations," Pydantic Documentation, 2024. 20

[29] I. Fette and A. Melnikov, "The websocket protocol," RFC 6455, 2011. 23

[30] R. Harrison, "Langchain: A framework for developing applications powered by language models," LangChain Documentation, 2024. 33

[31] J. Liu, "Llamaindex: Indexing and querying large language models," LlamaIndex Documentation, 2024. 33