COMPARATIVE ANALYSIS OF LARGE LANGUAGE MODELS FOR CODE GENERATION: IMPLICATIONS FOR ROS AND GAZEBO SIMULATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ANKARA UNIVERSITY

by

Fatih GÖKMENOĞLU

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN
ARTIFICIAL INTELLIGENCE TECHNOLOGY

ANKARA
2024

ABSTRACT

Master Thesis

COMPARATIVE ANALYSIS OF LARGE LANGUAGE MODELS FOR CODE GENERATION: IMPLICATIONS FOR ROS AND GAZEBO SIMULATIONS

Fatih GÖKMENOĞLU

Ankara University
Graduate School of Natural and Applied Sciences
Department of Artificial Intelligence Technology

Supervisor: Assoc. Prof. Hale ILGAZ

This study comparatively investigated the potential contributions of large language models to the development process for robotic applications by analyzing the simulation performance of codes generated by different models. Within the scope of the study, different robotic tasks such as basic navigation and object manipulation were defined and ROS-compliant code generation was performed by large language models by determining appropriate scenarios for these tasks. The generated codes were employed in the Gazebo simulation environment and their task completion success was compared in reference to resource utilization and simulation speed. Qualitative evaluation of the generated code, such as readability and maintainability, was also performed. This study demonstrated the aptitude of large language models for robotic application development processes by realizing the code writing process in a shorter time and in a systematic way without losing efficiency.

August 2024, 57 pages

Key Words: Large Language Models, Code Generation, Robotics, ROS, Gazebo

ÖZET

Yüksek Lisans Tezi

KOD ÜRETİMİ İÇİN BÜYÜK DİL MODELLERİNİN KARŞILAŞTIRMALI ANALİZİ: ROS VE GAZEBO SİMÜLASYONLARI İÇİN ÇIKARIMLAR

Fatih GÖKMENOĞLU

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Yapay Zekâ Teknolojileri Anabilim Dalı

Danışman: Doç. Dr. Hale ILGAZ

Bu çalışma, robotik uygulamalar için büyük dil modellerinin geliştirme sürecine olan potansiyel katkılarını, farklı modeller üzerinden üretilen kodların simülasyon performans analizini yaparak karşılaştırmalı bir şekilde araştırmıştır. Çalışma kapsamında basit navigasyon, nesne manipülasyonu gibi farklı robotik görevler tanımlanmış olup bu görevler için uygun senaryolar belirlenerek ROS uyumlu kod üretimleri büyük dil modelleri tarafından gerçekleştirilmiştir. Üretilen kodlar Gazebo simülasyon ortamında çalıştırılmış ve kaynak kullanımı ve simülasyon hızı metrikleri ile bu kodların görevi tamamlama başarıları karşılaştırılmıştır. Ayrıca, üretilen kodların okunabilirliği ve sürdürülebilirliği gibi nitel değerlendirmesi de gerçekleştirilmiştir. Bu çalışma, büyük dil modellerinin robotik uygulama geliştirme süreçleri için potansiyel katkısını, kod yazma sürecini verim kaybı yaşamadan daha kısa sürede ve sistematik bir şekilde gerçekleştirerek göstermiştir.

Ağustos 2024, 57 sayfa

Anahtar Kelimeler: Büyük Dil Modelleri, Kod Üretimi, Robotik, ROS, Gazebo

FOREWORD AND ACKNOWLEDGEMENTS

I would like to express my gratitude to my sole supporter, Assoc. Prof. Hale ILGAZ for her supervision in the development journey of my master's thesis. Despite of all the difficulties that I have gone through, tolerating me and providing her tremendous guidance, I was able to complete my work. Her inspiring counselling will be of further value in my future studies.

Fatih GÖKMENOĞLU
Ankara, August 2024

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANOVA	Analysis of Variance
CPU	Central Processing Unit
GPT	Generative Pre-trained Transformer
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KDL	Kinematics and Dynamics Library
LLM	Large Language Model
NLP	Natural Language Processing
ROS	Robot Operating System
RViz	ROS Visualization
TII	Technology Innovation Institute
URDF	Unified Robotics Description Format
XML	Extensible Markup Language

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A large language model (LLM) is an intriguing application of artificial intelligence (AI) studies; and this follows the fact that many are considering LLMs' utilization as a part of their professional activities. These include education, finance, healthcare and many more areas. Code generation being another subject of interest for the employment of LLMs and being investigated by different parties either in academia or industry. Since it has a wide range of different applications, one has a greater chance of coming across studies that dwell on the use of LLMs for specific code generation activities such as class-level Python code generation (Du et al., 2024) and Verilog hardware design generation (D'Hollander et al., 2024). In addition, the intersection of AI and robotics has turned out to be a focal point of technological advancement; and LLMs may prove to be a powerful tool for automating complex tasks, one of which is code generations.

This thesis aims to conduct a comparative analysis of different LLMs in the context of code generation for various robotic applications, with a particular focus on Robot Operating System (ROS) and Gazebo simulation environment (Koenig & Howard, 2004). The primary objective of the study is to evaluate and compare the performance of four prominent LLMs; namely, GPT-4o (OpenAI, 2024), Llama 3 70B (Meta, 2024), Claude 3.5 Sonnet (Anthropic, 2024) and Gemini 1.5 Flash (Google, 2024). These LLMs are easily accessible for a broad audience for free. The study is to particularly assess the capabilities of LLMs in code generation for a robot arm to execute a planned path movement and for a mobile robot to perform its navigation within a simulated household environment. The assessment is also to extend beyond mere code generation by including Unified Robot Description Format (URDF) file creation, which is pivotal in ROS and hence Gazebo simulations. This aspect of the study highlights the practical implications of code generated by LLMs in real-world robotic applications; and a multi-faceted approach is employed to better understand how LLMs could help designers streamline processes by addressing complexity in the robotic application development, and to explore AI-assisted development methods in robotics.

The research is primarily targeted at researchers studying robotics with the aim of providing insights into the faculties and limitations of current LLMs in code generation for robotic applications. By carrying out this comparative analysis, answers for how LLMs perform in ROS-compatible code generation, what the strengths and limitations of different LLMs in robotics-specific tasks, and how LLMs can be effectively integrated into robotics development workflows are sought after.

1.1 Background on Robotics Simulation and LLMs

In latest years, the field of Natural Language Processing (NLP) has beheld a notable reform with the advent of LLMs. Due to extensive training on vast textual datasets, LLMs have demonstrated exceptional proficiency in understanding, generating, and reasoning over human language (Brown et al., 2020). This capability has led to state-of-the-art performance across a wide array of NLP tasks, including text generation, translation, summarization, and question answering, among others (Raffel et al., 2020).

Being one of the leading examples of LLMs, (Generative Pre-trained Transformer) GPT-3, of OpenAI, has demonstrated exceptional language understanding and generation faculties, triggering extensive interest and exploration of its potential applications across a number of domains (Brown et al., 2020). Despite the fact that they have initially been designed for text-based tasks, the adaptability of LLMs has paved the way for their investigation in domains other than traditional NLP, including robotics and simulation.

The development and testing of robotic systems have proved the imperative role played by robotics simulation. Granting a safe and controlled environment for researchers and engineers to assess the performance of robotic architectures, algorithms, and control policies before employing them in real world, robotics simulation has achieved it by means of simulation components such as ROS (Macenski et al., 2022) and Gazebo (Koenig & Howard, 2004). These have turned out to be industry standards, providing potent tools for modeling, simulating, and visualizing robotic systems and their interactions with virtual environments.

1.2 Potential of LLMs in Code Generation and Robotics

LLMs have mainly been studied for text-based tasks, yet their capability to comprehend and generate human-like language unlocks attracting potentials for their application in robotics simulation elements like ROS and Gazebo. By leveraging the language understanding and generation capabilities of LLMs, robotics research and development could be promoted by virtue of the simplified and streamlined aspects of robotics simulation including environment creation, task specification, and system configuration.

In one potential application of LLMs in robotics simulation; that is, the generation of virtual environments and scenarios, LLMs could enable users to describe desired environments and scenarios in natural language instead of manually constructing complex 3D environments or writing lengthy configuration files. These descriptions are then taken and turned out to be the corresponding simulation environments, which are complete with accurate representations of objects, terrain, lighting conditions, and other relevant elements.

An additional favorable application, the specification of robotic tasks and behaviors, traditionally involves writing intricate code or using specialized languages and interfaces to program robotic behaviors and task sequences. Having researchers and developers describe desired robot behaviors and tasks using natural language instructions, LLMs could then interpret these instructions and generate the corresponding code or configurations required to execute those tasks within the simulation environment (Wang et al., 2024).

1.3 Aim and Objectives of the Study

The chief goal of this study is to examine the potential of LLMs in the development of ROS applications and their following simulation in Gazebo. Particularly, the study seeks to:

Investigate LLM Capabilities in Code Generation: Evaluate the effectiveness of LLMs in generating ROS-specific code, focusing on tasks such as node creation, message passing, and service handling.

Performance Comparison: Assess the performance of distinct LLMs in generating ROS code, using metrics such as code accuracy, efficiency, and the quality of generated simulations.

Simulation Validation: Assess the legitimacy and reliability of the generated code by running simulations in Gazebo, checking how fairly the generated code performs in a simulated environment.

Identify Challenges and Limitations: Distinguish the challenges and limitations accompanying the use of LLMs for ROS application development, offering insights into areas for future improvement.

1.4 Scope and Limitations

This study dwells on the application of LLMs in the development of ROS applications and their corresponding simulation in Gazebo. The scope involves the following.

Selection of LLMs: The study will assess a number of leading LLMs; that is, GPT-4o (OpenAI, 2024), Llama 3 70B (Meta, 2024), Claude 3.5 Sonnet (Anthropic, 2024) and Gemini 1.5 Flash (Google, 2024), to find out their effectiveness in generating ROS code.

Code Generation Tasks: The evaluation will contain numerous code generation tasks pertaining to ROS, such as node creation, message type definition, and service call implementation.

Simulation Scenarios: The generated code will be trialed in Gazebo simulations, covering a range of scenarios to evaluate the robustness and reliability of the code.

Performance Metrics: The study will make use of metrics including accuracy and efficiency of the code along with the simulation quality to measure the performance of different LLMs.

Nevertheless, the study has certain limitations, listed below.

Training Data of LLMs: Despite of their power, LLMs are limited by the data they were trained on. The code generated may not always be optimal or error-free.

Generalizability: The findings of this study are exclusive to the selected LLMs and the particular tasks evaluated. They may not be generalizable to all LLMs and/or all robotics applications.

Simulation Constraints: This is more of a warning that simulations in Gazebo, while realistic, cannot fully replicate real-world conditions. The performance of the generated code in simulations may not always translate to real-world performance.

1.5 Significance of the Study

The main significant point of this study is leveraging the capabilities of LLMs, simplification and thus acceleration of the development of robotic applications, which could in turn promote the faster prototyping and iteration, enabling more rapid advancements in the field.

LLMs generating the initial code pieces can pave the road to entry for robotics development, which could lead to the democratization of robotics development. Individuals without extensive programming expertise can utilize natural language instructions to develop robotic applications, broadening active participation in the field.

Furthermore, the findings of this study deliver practical insights into the strengths and limitations of current LLMs in robotics development, which can in turn contribute to the

development of future LLMs and their integration into robotics frameworks. It adds to the growing body of research on AI-assisted software development. In other words, establishing a framework for evaluating the performance of LLMs in robotics code generation, this study delivers a foundation for future research. Succeeding studies can build on this framework to delve into new applications and improvements.

2. THEORATICAL FRAMEWORK AND LITERATURE REVIEW

2.1 Overview of LLMs

Today, many view LLMs as a notable advancement in artificial intelligence studies, particularly in NLP. These models are trained on gigantic datasets including a wide variety of text, enabling them to learn the statistical properties of language and thus be able generate coherent, contextually relevant text. Well-known examples of LLMs include GPT-4 (OpenAI, 2024), Gemini (Google, 2024), and Llama 3 (Meta, 2024).

Operation of LLMs depends on the transformer architecture utilizing attention mechanisms to process input data and generate output (Vaswani et al., 2017). This architecture enables LLMs to handle long-range dependencies in text and produce high-quality text generation. LLMs learn from a large corpus of text, capturing general language patterns and structures. Moreover, there is a further stage to LLMs, which is called fine-tuning. It includes adapting the pre-trained model to specific tasks or domains using smaller, task-specific datasets, which results in extending the capabilities of LLMs beyond simple text generation. At this time, they can perform tasks such as translation, summarization, question answering, and even complex problem-solving, making them versatile tools in both research and practical applications.

GPT-3 shows that few-shot learning performance across various NLP tasks is significantly improved by scaling up language models (Brown et al., 2020). The research demonstrated the strong performance of the model in translation, question-answering, and reasoning tasks, suggesting that this could be achieved by large-scale models without task-specific fine-tuning being required.

A separate study, a comprehensive overview of LLMs (Zhao et al., 2023), presents their evolution from statistical approaches to neural models and pre-trained language models (PLMs). The scaling effects leading to emergent abilities in LLMs are explored in this

survey, and key aspects such as pre-training, adaptation tuning, utilization, and capacity evaluation are discussed.

The capabilities of LLMs extend beyond simple text generation. Tasks such as translation, summarization, question answering, and even complex problem-solving can be performed by them, making them versatile tools in both research and practical applications. Contemporary LLMs have unlocked many new features such as AI-assisted programming. Being able to understand and generate code, LLMs can support developers by automating repetitive coding tasks, suggesting code completions, and even identifying and fixing bugs. This proficiency is particularly relevant in the context of robotics, where the complexity of software development can be an important barrier.

However, important considerations are raised by the development and deployment of LLMs. The broader implications of these models, termed "foundation models," are examined by (Bommasani et al., 2022). The adaptability and impact of these models across various domains are highlighted in this research, while the need to address potential risks and societal impacts is emphasized.

Besides, efforts to align LLMs with user intent and ethical considerations are continuing. InstructGPT (Ouyang et al., 2022) offers an approach to fine-tuning language models with human feedback. Improvements in truthfulness and reduced toxic output generation are shown in this research. Likewise, Constitutional AI (Bai et al., 2022) is a novel approach for safer AI systems to be developed without direct human labeling of harmful outputs being required.

Considering the fact that LLMs will continue to evolve, numerous fields are expected to be revolutionized by them, while challenges that require ongoing research and ethical consideration are also presented.

2.2 Robotics and Simulation in ROS and Gazebo

ROS is an adaptable framework for developing robotic software. It presents tools and libraries that make the process of building, simulating, and deploying robots easier. ROS's architecture is modular such that different functionalities are encapsulated in nodes that communicate with each other through messages. This modularity simplifies the integration of various components such as sensors, actuators, and control algorithms, turning ROS into a powerful tool for both research and industrial applications (Macenski et al., 2020).

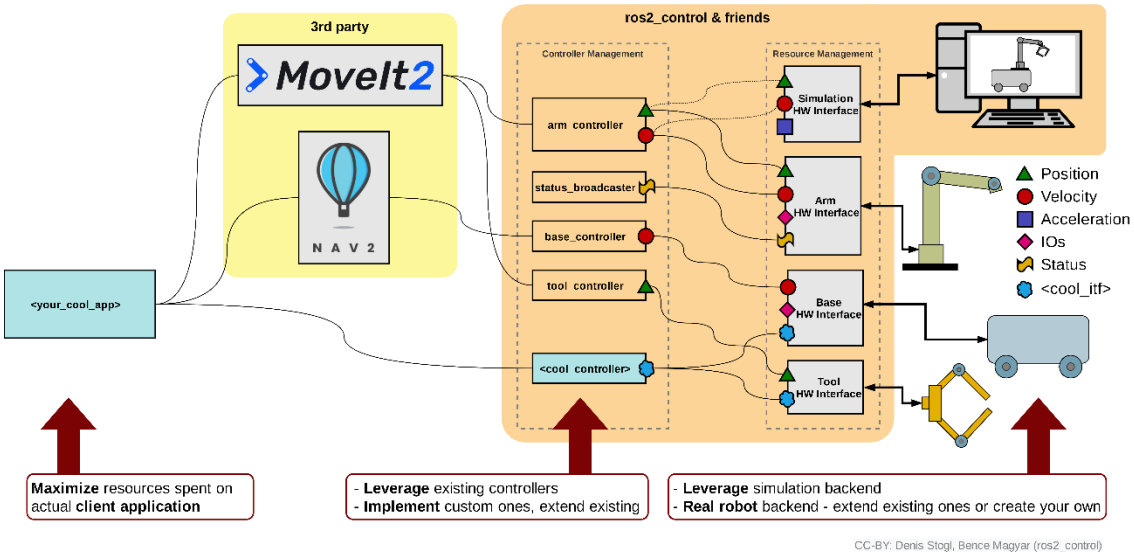

Figure 2.1 ROS framework (Stogl & Magyar, 2022)

Gazebo is an open-source toolbox that enables to simulate robots, sensors and environments by providing an ample set of libraries, cloud services and plugins to iterate promptly on design. Gazebo offers high-fidelity physics simulation, realistic rendering, and a diversity of sensors and actuators, which grants developers with simulation of complex interactions between robots and their environments. This aptitude is decisive for testing algorithms under various conditions and improving them before deployment in real-world scenarios (Koenig & Howard, 2004).

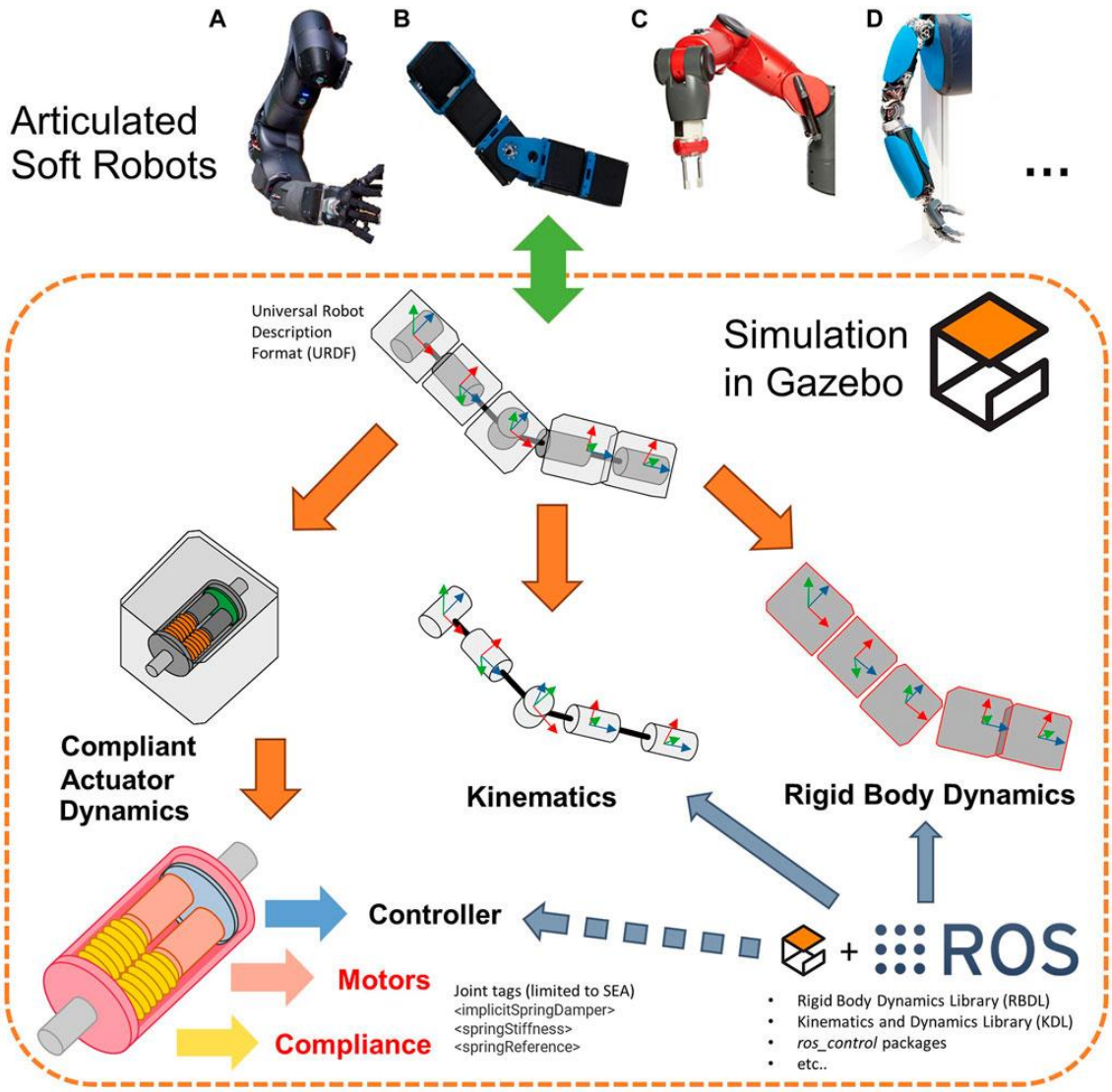

Figure 2.2 Simulation in gazebo (Mengacci et al., 2021)

ROS and Gazebo jointly make a complete toolkit available for the development and testing of robotic systems. ROS manages the software architecture and communication between components; and Gazebo provides a realistic simulation environment for testing and validation. Collectively, they empower developers to build robust, scalable, and reliable robotic applications.

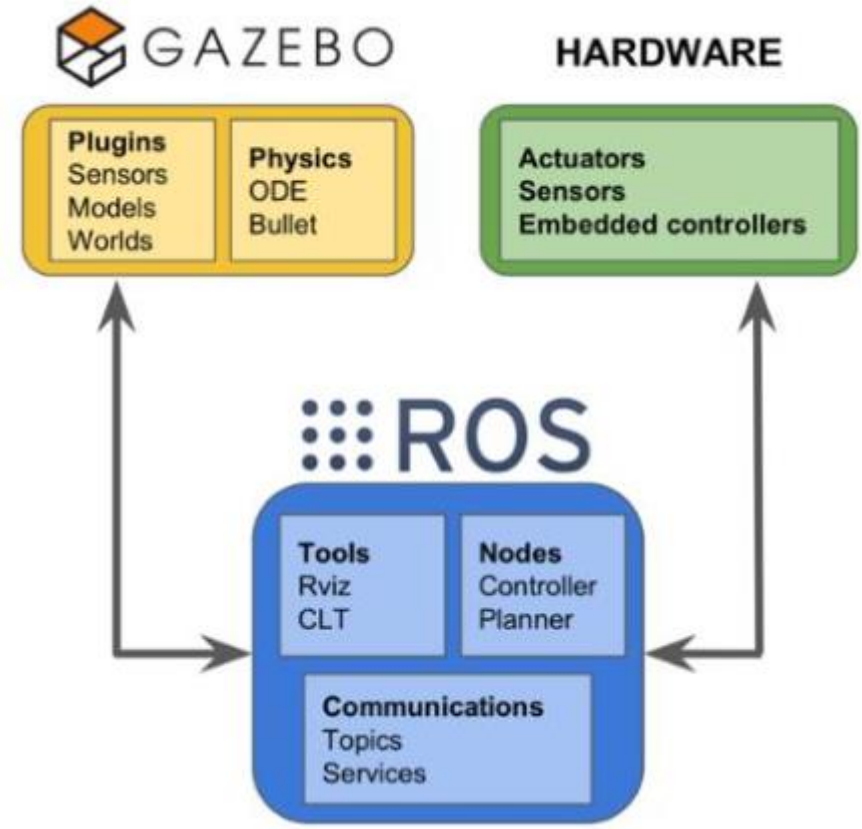

Figure 2.3 ROS-gazebo interaction (Mittal, 2018)

2.3 Previous Studies on Code Generation Using AI

The use of AI in code generation has proved to be an active area of research for several years. Early efforts were on rule-based systems and expert systems that could generate code in reference to predefined templates and rules. Nonetheless, these approaches turned out to be limited in their capability to manage the complexity and variability of real-world programming tasks.

Beginning with the application of machine learning techniques, and particularly advances in deep learning, automatic code generation pressed forward. Neural network-based AI models, especially sequence-to-sequence models and transformers revealed the competence to generate high-quality code from natural language descriptions by learning

from large datasets of code and text, and capturing the syntactic and semantic structures of programming languages.

Later studies focused on the use of LLMs for code generation and demonstrated promising results. For instance, OpenAI's Codex, descendent of GPT-3, exposes an impressive aptitude in generating code snippets or completing code, and even transforming instructions in natural language into functional code (Du et al., 2024). These also emphasize the potential of LLMs in numerous programming tasks including code synthesis, code summarization, and bug detection (Hendrycks et al., 2021; Austin et al., 2021).

In the meantime, reviews accompanied the applications. Discussions on code generation using LLMs, covering both the application of the techniques and the evaluation of generated code presented that LLMs' strong code understanding and writing abilities have enabled their use across various software engineering tasks, significantly boosting developer productivity. Nevertheless, a notable imbalance in the research focus; that is, evaluation receiving less attention compared to application studies, was also shown. To keep pace with rapid advancements in application techniques, more effort was called to develop robust evaluation methods, thereby bridging the gap between evaluation and application research in LLM-based code generation (Wang & Chen, 2023).

Over the course of time, studies have been conducted in an effort to make LLMs more robust against peculiar problems. In the face of this generalization challenge, various methods were offered. A novel self-iteration code generation framework utilizing LLMs was developed to address complex coding tasks (Chang et al., 2023). The approach integrates software development methodologies, incorporating four key roles: analyst, designer, developer, and tester. Each role performs specific tasks in cycles. In other words, analysts and designers refine requirements and task designs based on tester feedback, whereas developers refactor or modify code until it passes testing. Through extensive experimentation across multiple benchmarks yielding promising results, the framework demonstrates robust generalization capabilities, thereby improving code generation quality across various LLMs. There is also Knowledge-Aware Code

Generation (KareCoder), which is a new approach. Focusing on the generalization challenges that LLMs faces while tackling peculiar problems, it presents CodeF, a dataset featuring programming problems unfamiliar to ChatGPT as well as a specialized knowledge library for Python programming contests. Those are integrated into the LLMs' reasoning process during code generation, significantly improving their problem-solving capabilities (Huang et al., 2024). Furthermore, in another study that investigates ChatGPT for its use in code generation for real-world software development tasks, authors have built a web-based user interface for coming up with more effective prompts to be fed into ChatGPT so that how prompting affects the code generation process could be demonstrated. The results indicate that building effective and comprehensive prompts is crucial to make better use of LLMs in modern software development by improving efficiency, reducing errors and promoting consistency (Li et al., 2024).

In parallel to the advancement in the application, evaluation methods have been also revised accordingly. To measure the capabilities of LLMs in more complex scenarios, a study has introduced ClassEval, the first benchmark for evaluating LLMs in class-level code generation. The benchmark consists of 100 class-level Python code generation tasks in the company of about 500 person-hours. Different LLMs have been investigated based on this benchmark on class-level code generation; and the results indicate that LLMs do considerably worse on class-level code generation tasks in comparison to method-level ones. Their inability to generate dependent code calls for software engineering expertise for practical and more complicated software development cases (Du et al., 2024). Another empirical study to assess ChatGPT's zero-shot learning capability focuses on simple code snippet generation and explores its ability to improve partially successful solutions when provided with additional feedback (Yan et al., 2023).

While this historical context provides an overview, particular use cases may help to illustrate how LLMs are applied in practice for code generation. In an attempt to assist electronic systems' design, research into automating Verilog code generation using LLMs has shown promising results. The study focuses on fine-tuning LLMs using available datasets from various sources. An evaluation framework has also been constructed for both syntactical and functional analysis of generated Verilog code for a given problem.

Fine-tuned LLMs demonstrate an overall increase in performance for producing syntactically correct code. Also, functionality of the generated code by those may outperform other commercial LLMs (Thakur et al., 2023).

There are also studies examining the potential use of AI for building ROS-based applications. A recent study dwells on the faculty of LLMs to generate code for designing manipulation plans which allow robots to perform actions such as grasping an object and packing it into a box. To evaluate their abilities, different LLMs have been used in an experimental framework to create a plan for a given target action coupled with a reference example. Generated plans are evaluated against particular ground truth using code generation metrics, along with a check for whether they compile or not. Although, there is a difference between the performance of LLMs compared in the study, the overall results indicate that LLMs perform good for the initial plan generation; however, they are not reliable enough to be used for the further refinement of what has been initially created (Töberg & Cimiano, 2023). Yet, the integration of LLMs with robotics platforms like ROS and Gazebo remains an emerging area of research, with opportunities for further investigation.

2.4 Integration of LLMs into Robotics

Incorporating LLMs into robotics elements such as ROS and Gazebo signifies a frontier in AI-assisted robotics development. This addition encompasses use of the language understanding and generation faculties of LLMs to simplify and even promote the process of developing robotic applications.

One of the key benefits of combining LLMs with ROS is the ability to interpret natural language descriptions regarding robotic tasks and produce the corresponding ROS nodes, messages, and services, which could considerably reduce the time and effort required to develop robotic systems, democratizing robotics programming to a broader audience. Their ability to process and internalize lots of chunks of data, including instructions, technical manuals, and academic articles, enables them to provide factual and coherent responses to complex queries. This faculty of LLMs extends beyond code synthesis and

natural language instruction translation, potentially addressing the bottleneck in robot design (Stella et al., 2023). Besides, assisting in the iterative development process by providing real-time feedback and suggestions, LLMs could enhance the efficiency and accuracy of the development process, leading to more robust and reliable robotic applications.

Although LLM assisted code generation is a popular research subject, the collaboration between LLMs and robotics application development calls for more attention. Yet, there has been a few recent studies that investigated how LLMs could be utilized. One of them exploring the potential of LLMs to translate goals expressed in natural language into structured plans leverages LLMs' strong natural language processing capabilities (Xie et al., 2023). While LLMs of interest excel at translation by effectively using commonsense knowledge to fill in missing details from under-specified goals, challenges persist in tasks involving numerical or physical reasoning. Also, LLMs show sensitivity to prompt phrasing, which requires a careful implementation by the developer.

Multi-object rearrangement, a critical skill that service robots shall have, has been a research topic; and a novel approach called LLM-GROP has been developed (Ding et al., 2023). Using prompts to extract commonsense knowledge about semantically valid object configurations through LLMs and implements them using a task and motion planner, it translates natural language commands into plans for object arrangement across varied environments. An implementation of the system on a mobile manipulator for real-world scenarios demonstrates its potential for enhancing robot performance in complex tasks, emphasizing the growing significance of LLMs in robotics. In spite of that, challenges remain since LLMs still lack the faculty of physical reasoning or handling complex environments.

While text-only LLMs face challenges especially in embodied tasks for the reason that they have limited visual perception compatibility, a framework utilizing multimodal GPT-4V to enhance embodied task planning has been proposed (Wang et al., 2024). The framework combines natural language instructions with robot visual perceptions. Results indicate that GPT-4V considerably boosts robot performance in embodied tasks. Also,

the research provides valuable insights into LLM-centric embodied intelligence and offers perspectives on advancing application development.

Finally, in simulation environments such as Gazebo, LLMs can be used to build up simulation scenarios. By describing the required simulation parameters in natural language, developers can make use of LLMs to set up complex simulation environments with no difficulty. This can simplify the testing of robotic algorithms under diverse conditions and help identify potential issues before deployment.

3. MATERIALS AND METHODS

3.1 Overview of LLMs

Pipelining the expertise of LLMs into ROS development and experimenting within Gazebo simulation environment is the groundwork of this comparative analysis. LLMs chosen for this study include GPT-4o (OpenAI, 2024), Gemini 1.5 Flash (Google, 2024), Llama 3 70B (Meta, 2024), and Claude 3.5 Sonnet (Anthropic, 2024), each of which is a representative of AI-driven natural language processing and coding capabilities. These models are leveraged owing to their demonstrated aptitude for grasping and generating crucial programming elements and applicable potential for robotics task automation.

GPT-4o, extensively trained on a rich and diverse scope of web data along with industry-standard machine learning datasets including code and math data to help the model develop strong reasoning skills, can generate code snippets utilizing best coding practices, which makes it a good option for generating ROS-compatible scripts. Its parameter size is not publicly announced, yet it is a known fact that it has 8K context length. Also, the knowledge cutoff date is August 2023 as of the date of this study (OpenAI, 2024).

A powerful, multi-task capable model, Gemini 1.5 Flash excels in output quality, which is another strong candidate to produce the desired code pieces. Its training dataset consists of data sourced across many different domains including code content. Similar to GPT-4o, the parameter size is not explicitly available for Gemini 1.5 Flash. The context length is considerably larger compared to many other LLMs; that is 1M tokens. Regarding the knowledge cutoff date, it is November 2023 (Google, 2024).

Llama 3 70B is recognized for its commendable performance in specific coding tasks, particularly due to its ability to generate efficient and largely error-free code. This capability is attributed to its extensive training on a diverse set of publicly available online data. Having 8K of context length, the model will be made use of through its respective APIs, configured with parameters optimized for code generation, including adjustments

for verbosity, style, and complexity of the output to align with ROS standards (Meta, 2024).

The last LLM involved in this study, Claude 3.5 Sonnet, has a good command of various tasks. The model has been trained on a proprietary mix of information coupled with non-public data from third parties such as data labeling services and paid contractors. The parameter size is not available for Claude 3.5 Sonnet although it has a relatively large context length; that is, 200 K. As for the knowledge cutoff, it has the most recent information compared to other LLMs, with April 2024.

3.2 ROS and Gazebo Simulation Environment

The study makes use of ROS Humble Hawksbill and Gazebo on Ubuntu 22.04 (Sobell, 2015), installed on a personal computer. The setup is for ensuring compatibility with the latest robotics software libraries; and it delivers a stable platform for deploying and testing the generated code.

The Gazebo simulation scenarios include a set of standard tasks utilized in robotics research, such as navigation, object manipulation, and environment interaction. These are made to evaluate the real-world utility of the generated code, taking the robot's capability to run tasks accurately and efficiently under simulated real-world conditions into account.

3.3 MoveIt2 and Navigation 2 Stack

The simulations implemented based on navigation and object manipulation tasks use open-source frameworks, MoveIt2 and Navigation 2 Stack.

Being a robotic manipulation platform for ROS2, MoveIt2 serves as a significant tool in modern robotics development. The platform offers a suite of packages to be used in commercial application development, design prototyping and algorithm benchmarking. Main application areas of MoveIt2 includes motion planning, manipulation as well as 3D

perception. Having an easy-to-use interface coupled with documentation for guidance, which simplifies the development process, it enables the creation of intelligent and responsive robotic systems (Coleman et al., 2014).

Navigation 2 stack, or simply Nav2 is a comprehensive group of packages for ROS2 to be used for managing navigation tasks in robotic applications. It offers high performance and flexibility in mobile robotics development incorporating a set of key components including environment mapping, localization, path planning and obstacle avoidance. Nav2's modular architecture enables developers to easily customize and extend navigation capabilities in their applications, which is another aspect that makes Nav2 an essential tool in autonomous mobile robot development (Macenski et al., 2020).

3.4 URDF and KDL

In a robotics project, as the complexity of the system increases, there might be diverse software components that demands information of the robot's physical characteristics regarding its kinematics and dynamics. For the sake of simplicity and consistency, it is a best practice to reserve all this information in one place, from which it can be easily referenced. In ROS, this robot description is generally stored in a URDF (Unified Robot Description Format) file. URDF is an XML (Extensible Markup Language) specification for modeling multibody systems including robotic manipulator arms and animatronic robots. As it is the case in other types of XML files, URDF files also consist of several XML elements; and those include <robot>, <link> and <joint>, all nested in a hierarchical structure; that is, an XML tree (Tola & Corke, 2023).

A URDF file is particularly relevant in robotics studies for several reasons. Once it is prepared, a URDF file provides a standardized description of a robot's structure, defining its links, joints and their corresponding relations. Being widely used in ROS ecosystem, URDF is commonly used with tools such as RViz (ROS Visualization) and simulator including Gazebo (Kam et al., 2015).

KDL (Kinematics and Dynamics Library) is a toolchain developed in C++ that, as its name suggests, offers tools for kinematics and dynamics calculations. It is highly relevant to robotic studies for a developer can model a robotics system and kinematics chains easily. Furthermore, rapid algorithm implementation and behavior analysis are other featured aspects of the library. KDL is commonly used in both academic and industrial applications. It is particularly valuable for ROS-based projects due to its support for 3D frame and vector transformations. Kinematic chain construction from a URDF as well as forward position kinematics computation are faculties that make KDL a preferred framework in many robotic applications (Khokar et al., 2015).

3.5 Tasks

Three separate tasks have been defined to evaluate the code generation performance of large language models. These tasks are representative for the basic functions commonly encountered in the field of robotics and they provide a suitable basis for evaluating the potential of large language models in robotics applications.

3.5.1 Object manipulation task

The first task is object manipulation; and it involves robot actions such as sensing, grasping, lifting, moving and placing objects in an environment at a specific position. Object manipulation is a fundamental capability for industrial robots and service robots. In this thesis, the transition of a robotic arm from one state to another has been simulated.

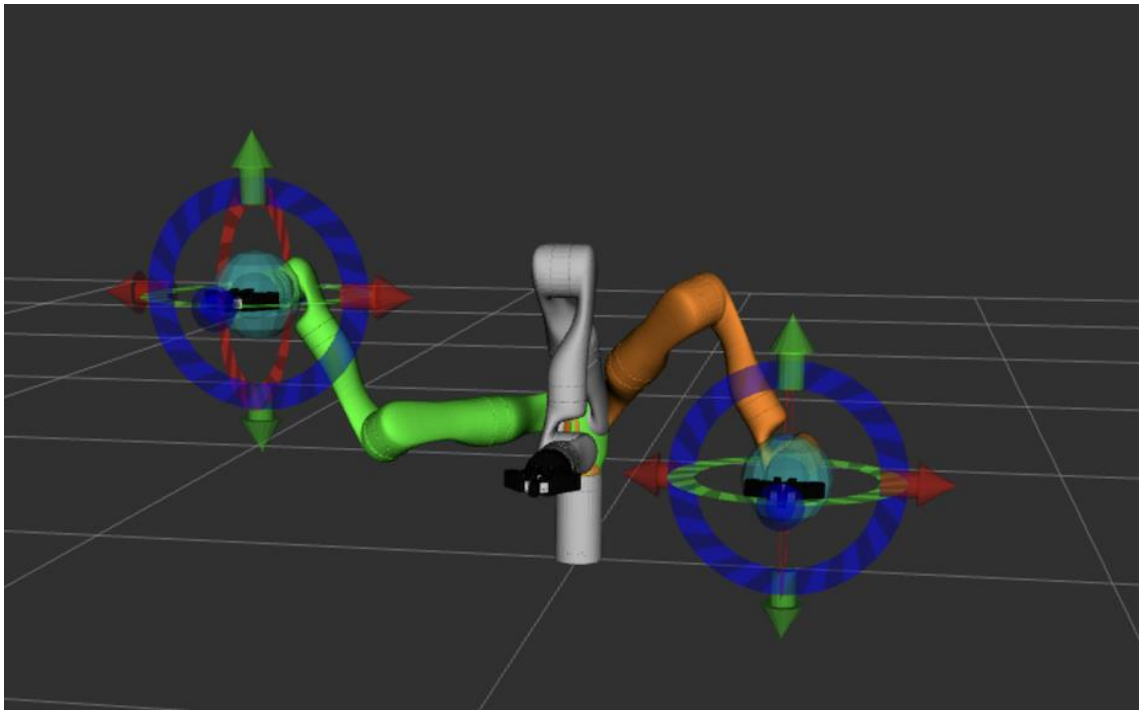

Figure 3.1 Object manipulation task robotic arm

3.5.2 Navigation task

Another task, navigation, is about the safe and efficient movement of a robot from its current location to a target location. This involves the robot recognizing its surroundings, determining its own position, planning a path to the target location, and detecting and avoiding potential obstacles while following that path. In this study, an example of a mobile robot moving from a given starting point to a predetermined target point in an environment with obstacles.

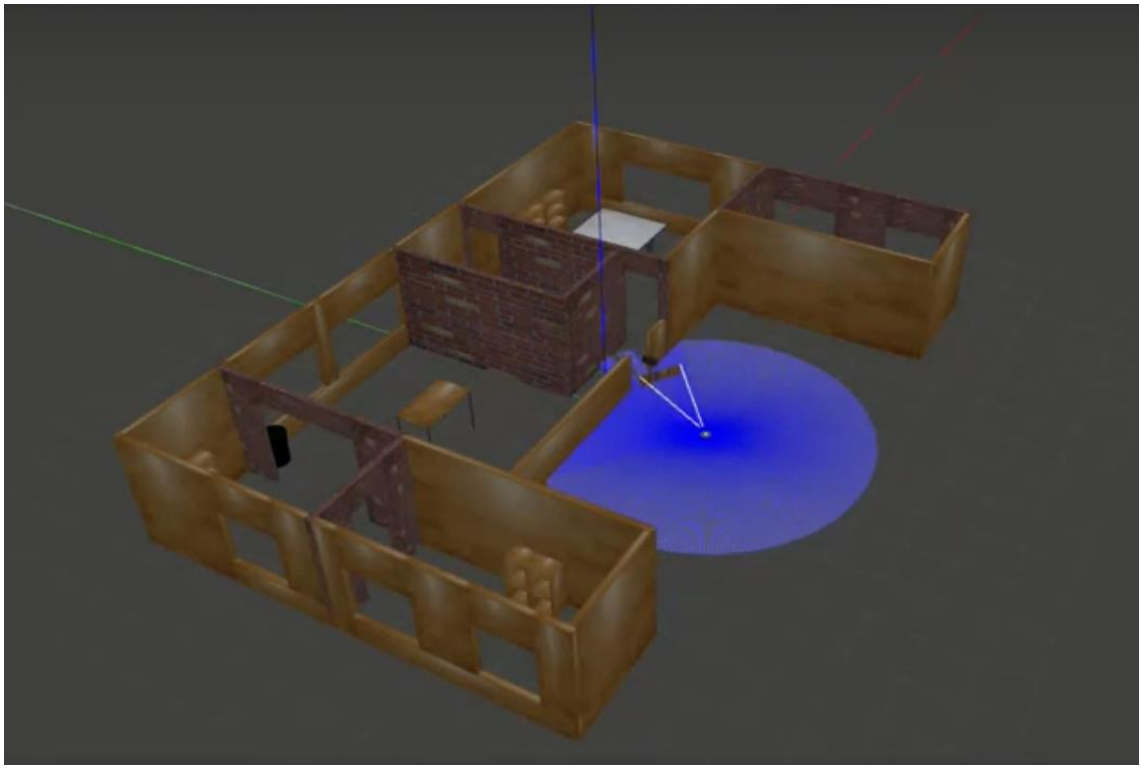

Figure 3.2 A mobile robot navigating in a household

3.5.3 Robot description generation task

URDF is an XML-based file that describes the physical structure, joints, connections, and sensors of a robot and determines its kinematic and dynamic properties, visual representation, and how it behaves in a simulation environment. In this thesis, the design of a simple robot with a box-shaped body on three wheels and a LiDAR sensor on top of it.

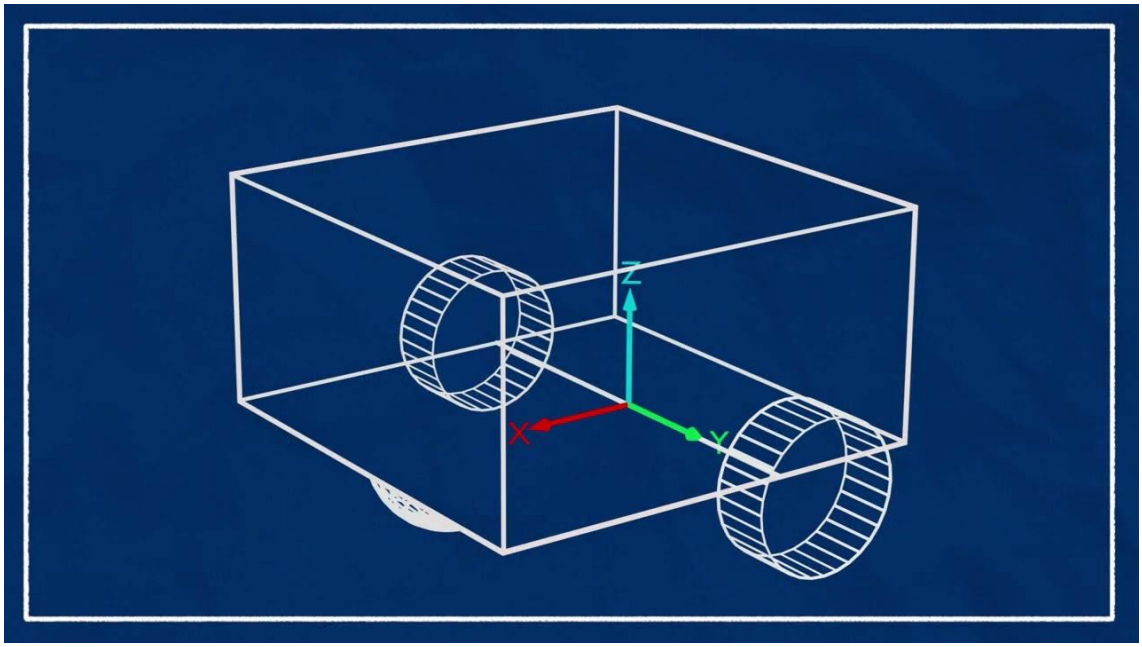

Figure 3.3 A simple robot rough sketch

3.6 Human Developer

The study involves a human developer along with the aforementioned LLMs. Human developer is an expert with a good command of different programming languages including C++ and Python as well as particular design patterns such as object-oriented programming. Having experience in AI and robotic application development, the expert does also have participation certificates for various online courses including C++ programming, machine learning and ROS-based application development. Before having LLMs generate code for a given task description through proper prompting, code pieces have been generated by the expert for each task to use those as a kind of ground truth.

3.7 Prompts

In this section, the prompts used in the study are provided. For each task, there is a prompt text along with the temperature and top_p values preferred. Temperature is a hyperparameter which controls the randomness of the model output. The higher the temperature is, more creative results a model produces; thus, for more deterministic and

conservative output low temperature values should be chosen. Top_p is another hyperparameter for a model that controls the randomness of its output. It is kind of a threshold probability which is responsible for selecting output tokens whose cumulative probability is greater than its value.

3.7.1 Object manipulation task prompt

For object manipulation task, the following prompt has been used for each LLM with temperature = 0.1 and top_p = 0.9.

Prompt: Act as a ROS2 developer. Your task is to complete the C++ code given below in reference to the following notes:

1. For your convenience, header files to be used in this code are given. This shall be enough to make the program up and running.

2. Inside main block, there are line comments that tells you what you shall be providing code for. Don't change these line comments. You can add extra comments.

3. Providing a code that is performant in terms of both execution time and resource utilization is a plus. Pay attention to it.

4. The node name will be hello_moveit. Use the same name for the logger as well.

5. The robot to be used is Panda Arm so use panda_arm as parameter in the corresponding part of the code.

6. The target pose orientation is 1.0. As for position values of x, y and z, they are 0.28, -0.2 and 0.5 respectively.

7. An error message such as "Planning failed!" would be nice to show in case of an error while executing the plan.

Code (hello_moveit.cpp):

```
#include <memory>

#include <rclcpp/rclcpp.hpp>

#include <moveit/move_group_interface/move_group_interface.h>

int main (int argc, char * argv[]) {

// Initialize ROS and create the Node

// Create a ROS logger

// Create the MoveIt MoveGroup Interface

// Set a target Pose

// Create a plan to that target pose

// Execute the plan

// Shutdown ROS

}
```

3.7.2 Navigation task prompt

For navigation task, the following prompt has been used for each LLM with temperature = 0.1 and top_p = 0.9.

Prompt: Act as a ROS2 developer. Your task is to complete the Python code given below in reference to the following notes:

1. For your convenience, header files to be used in this code are given. This shall be enough to make the program up and running.

2. Inside main block, there are line comments that tells you what you shall be providing code for. Don't change these line comments. You can add extra comments.

3. Providing a code that is performant in terms of both execution time and resource utilization is a plus. Pay attention to it.

4. A pose value consists of position and orientation. Position has three components; namely, x, y and z. Orientation has four components; namely, x, y, z, and w.

5. For poses, only position.x, position.y and position.z values are to be provided. Make sure the code manages the necessary transformations for the orientation components' values.

6. The position values for the initial pose are 0.0, 0.0 and 0.0 for position.x, position.y and position.z respectively.

7. The remaining positions values for the following poses in the waypoints list are as follows (the format is as x, y, z):

(2.0, -2.0, 1.57)

(4.0, 0.8, 0.0)

(8.0, 1.0, -1.57)

(8.0, -0.5, 1.57)

(5.0, 5.0, 3.14)

(3.0, 4.0, 1.57)

(4.0, 5.0, 0.0)

(5.0, 3.0, -1.57)

(4.0, 0.8, 3.14)

(-4.0, 3.5, -1.57)

(-4.0, 0.0, 1.57)

8. Getting navigation feedback within the code (no need for printing) is recommended.

9. Make sure that the navigation result is printed.

Code (nav2_test.py):

```
#!/usr/bin/env python3

import rclpy
```

```
from nav2_simple_commander.robot_navigator import BasicNavigator

from geometry_msgs.msg import PoseStamped

import tf_transformations

def main():

    # --- Init

    # --- Set initial pose

    # --- Wait for Nav2

    # --- Send Nav2 goal

    # --- Follow waypoints

    # --- Shutdown

if __name__ == '__main__':

    main()
```

3.7.3 Robot description generation task prompt

For robot description generation task, the following prompt has been used for each LLM with temperature = 0.1 and top_p = 0.9.

Prompt: I need your help to create a robot description file for my robotics project. The following list is the criteria that you shall consider.

1- The file is an XML file with .urdf extension.

2- The name of the robot is my_robot.

3- There will be three different materials; namely, grey, green and white. No need for transparency. The values of rgba should be chosen accordingly.

4- The robot will have a base with a lidar on top of it. The base will be a box with the following dimensions (in meters), length: 0.6, width: 0.4 and height: 0.2.

5- It will also have left and right wheels that are able to rotate. Last, the robot will have a caster wheel. Create appropriate links for these. Being appropriate means having proper geometry and origin. Regarding materials, base color will be green, wheels will be grey and lidar will be white.

6- Create appropriate joints for the links. Pay attention to the parent and the child of any joint. Origin of the joint is another crucial thing that you shall consider.

7- Create one final, virtual link called base_footprint and the corresponding joint with the base.

8- The robot description file will be visually inspected on RViz. Make sure that the developer who will use your .urdf XML file are able to see a robot on RViz.

9- Finally pay attention the naming, format and like.

3.8 Code Generation Process

To start code generation process, accordingly structured prompts are to be fed into the models, outputs of which will then be interfaced with ROS through a middleware layer. This layer is responsible for translating descriptions provided in natural language into ROS-compatible scripts or code, ensuring compatibility with ROS message passing and service calls. To illustrate, a navigation task, in which a robot has to arrive at a particular location by avoiding any obstacle it might come across, could be described in the form of a prompt to be fed to an LLM. Then the model generates script that implements related algorithms for pathfinding with obstacle avoidance.

3.9 Evaluation Metrics and Performance Assessment

LLMs have such astonishing natural language processing faculty that they do have a good command of reading and writing code. Also, they are capable of providing comments and processing given warnings or errors as a part of their prompts. Hence, LLMs has great potential in code generation by automatically managing various software engineering tasks. Despite of this potential, the code generation capability of LLMs massively depends on the code snippets that are available in their training data, which are mostly from open-source code repositories. Nonetheless, there exist potential vulnerabilities in these open-source codes, which may affect the output of LLMs and reduce the security of developed software. For this reason, it is of great importance to evaluate the quality and security of LLM-generated code from various aspects (Wang & Chen, 2023)..

Software quality involves various quality characteristics, which can also be called quality factors or quality attributions. For instance, functionality, performance efficiency, compatibility and like, and lots of corresponding sub-characteristics are part of these quality characteristics according to the systems and software engineering standards (ISO/IEC 25010, 2011).

This comparative analysis has certain evaluation metrics including the correctness of task execution, the efficiency of the code in terms of computation time and resource utilization and the adaptability of the generated code for future changes in parameters or environments. Also, there are qualitative metrics such as code readability and maintainability to judge how practical the generated software elements in ROS related project development.

Cyclomatic complexity (McCabe, 1976) metric is used to measure the formational complexity of a software. Measurement is performed based on the control flow graph of the software in reference to the number of linearly independent tracks through the source code. Cyclomatic complexity is particularly useful for the evaluation of the testability, maintainability and risk level of software modules. A higher score is a sign of more complex code that is not only more prone to errors but also harder to maintain. This metric is preferred by most software developers and quality assurance teams for checking software components whether they require refactoring or additional testing.

3.10 Data Collection and Analysis

ROS and Gazebo environment is suitable for collecting performance data through appropriate scripts, which will capture data on task completion times and resource usage as well as storing them in a format such that further analysis is available.

A few Python libraries are used in this study mainly for data analysis and visualization. These are NumPy (Harris et al., 2020), which is particularly useful in scientific computing and employed for efficient array operations, pandas (McKinney, 2011), which is for data analysis and manipulation on structured data, Matplotlib (Hunter, 2007), which is a practical tool for generating a wide range of visualizations, and seaborn (Waskom, 2021), which is yet another tool to create informative and visually appealing plots. All are utilized to process, analyze and visualize data collected during the study for deriving meaningful insights and presenting the findings effectively.

3.11 Ethical Considerations and Reproducibility

Ethical considerations are to be taken into account while describing task scenarios emphasizing the importance of accuracy and reliability in generated code to prevent failures that could lead to property damage or personal injury.

The study is to deliver complete documentation of all related details regarding code, models and environment for the sake of reproducibility. A public Git repository will be available for all software elements that are part of this comparative analysis to appeal further work to be conducted by other researchers who wish to replicate the findings and build upon them.

4. RESULTS

As for the object manipulation task, which is coded in C++, Claude 3.5 Sonnet emerges as the top performer in terms of elapsed time among other LLMs just before GPT-4o (13.07s vs 15.43s); and it also outputted more concise code (22 lines vs 27 lines). The end results indicate that Claude 3.5 Sonnet may have a slight edge in C++ coding for robotics tasks. However, the differences are relatively small, and the models performed well overall. Here, a task that requires more complex planning as well as having relatively longer execution duration might be a better candidate to compare the true performance of those LLMs.

The navigation task, on the other hand, are coded in Python; and it reveals more notable differences. Only the human developer and Claude 3.5 Sonnet generated code pieces achieved a successful waypoint follower. This suggests that Claude 3.5 Sonnet may have a particular competency in understanding and implementing navigation algorithms in Python for ROS applications. The fact that GPT-4o, Llama 3 70B, and Gemini 1.5 Flash failed to produce a running code for this task underlines the challenges involved and the potential superiority of Claude 3.5 Sonnet in this specific domain.

It is important to mention the human developer's performance especially in the navigation task. Although it takes significantly longer to complete the task (310 seconds compared to Claude 3.5 Sonnet's 293 seconds), the human developer code achieved a lower cyclomatic complexity (4 vs 7) and fewer lines of code (73 vs 90); which indicates that while LLMs may be faster in code generation, human supervision still has an edge in creating more elegant and efficient solutions, especially for complex tasks like navigation. In the object manipulation task, the human developer was actually the fastest (12.46 seconds), with slightly better performance across most metrics. This reinforces the idea that human expertise remains valuable, especially for tasks that may require domain-specific knowledge or a kind of intuition.

To sum up, albeit Claude 3.5 Sonnet shows promising performance, especially in navigation tasks, the results highlight how essential the human developer supervision is

in robotic application development. The complementary faculties of a human developer and an LLM, leveraging the speed of a model and the insight of a human developer, suggest that a collaborative approach could be the most effective strategy for robotic application development.

A final remark is for the tasks themselves. The disparity in performance between the object manipulation and navigation tasks is intriguing. All LLMs managed to complete the object manipulation task, but most failed at navigation. This could imply that navigation algorithms are more complex and require a deeper understanding of ROS and robotics concepts, which only Claude 3.5 Sonnet among the LLMs seems to possess. Object manipulation may be a more common task in programming datasets, making it easier for LLMs to generate working code. The specific requirements of the navigation task might align better with Claude 3.5 Sonnet's training or capabilities.

In the following subsections, for each different task, a detailed summary of the results coupled with the corresponding visualizations are provided. Following those, a visual comparison figure with an accompanying commentary is given for the URDF file generation task.

4.1 Object Manipulation Task

This section presents the findings for the comparative analysis of an object manipulation task implemented by different LLMs and human-generated code.

Table 4.1 Object manipulation task results summary

	Human	GPT-4	Claude	Llama 3	Gemini
# of Trials for a Successful Build	1	1	1	3	3+
Planning to the Target (sec)	0.0133	0.02453	0.0144	0.0143	N/A
Elapsed Time (sec)	12.4566	15.432	13.0654	13.8601	N/A
Resources					
CPU (%)	≈0	≈0	≈0	≈0	N/A
Memory (MB)	36.25	36.25	37.25	36.5	N/A
Code Complexity Analysis					
Cyclomatic Complexity	2	3	2	2	2
Lines of Code	32	27	22	19	21
Parameter Count	2	2	2	2	2

4.1.1 Performance metrics

The analysis focuses on the following key performance metrics: the number of trials required for a successful build, planning time, overall execution time, and resource usage.

4.1.1.1 Trials for successful build

All the models, except Gemini 1.5 Flash, successfully implemented the script for the object manipulation task. Notably, the human-generated code, GPT-4o, and Claude 3.5 Sonnet achieved a successful build in a single trial, demonstrating high reliability. Llama 3 70B, on the other hand, required three trials.

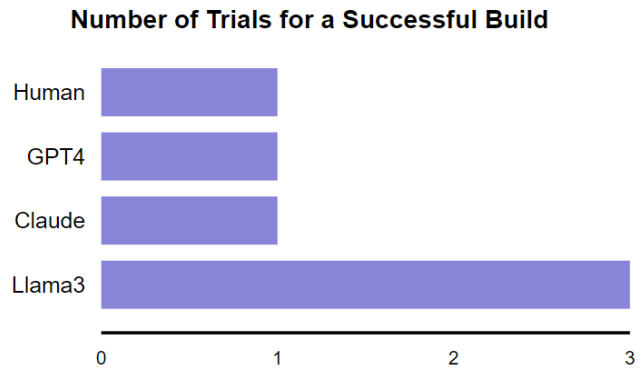

Figure 4.1 Number of trials for a successful build of object manipulation task script

4.1.1.2 Planning efficiency

The human-generated code produced the fastest planning time, closely followed by Claude 3.5 Sonnet and Llama 3 70B. Despite of its success, GPT-4o, required nearly twice the planning time of the human-generated code.

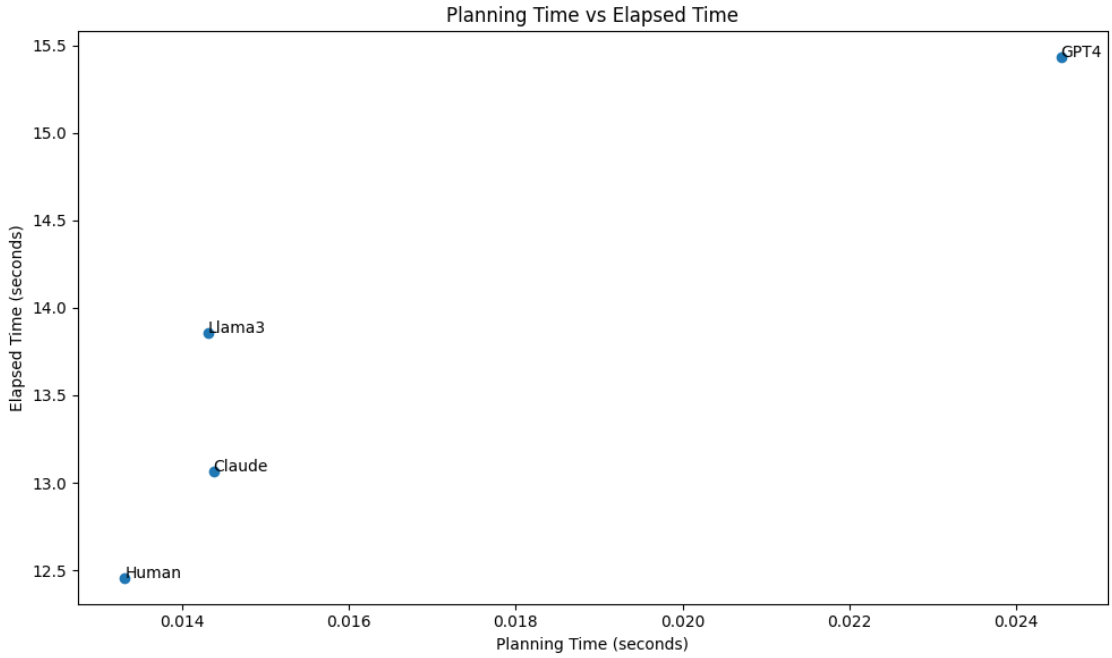

Figure 4.2 Planning time for the target pose compared to elapsed time

4.1.1.3 Overall execution time

The human-generated code maintained its efficiency lead in overall execution time. Interestingly, Claude 3.5 Sonnet outperformed Llama 3 70B in this metric, suggesting higher efficiency in execution despite their similar planning times.

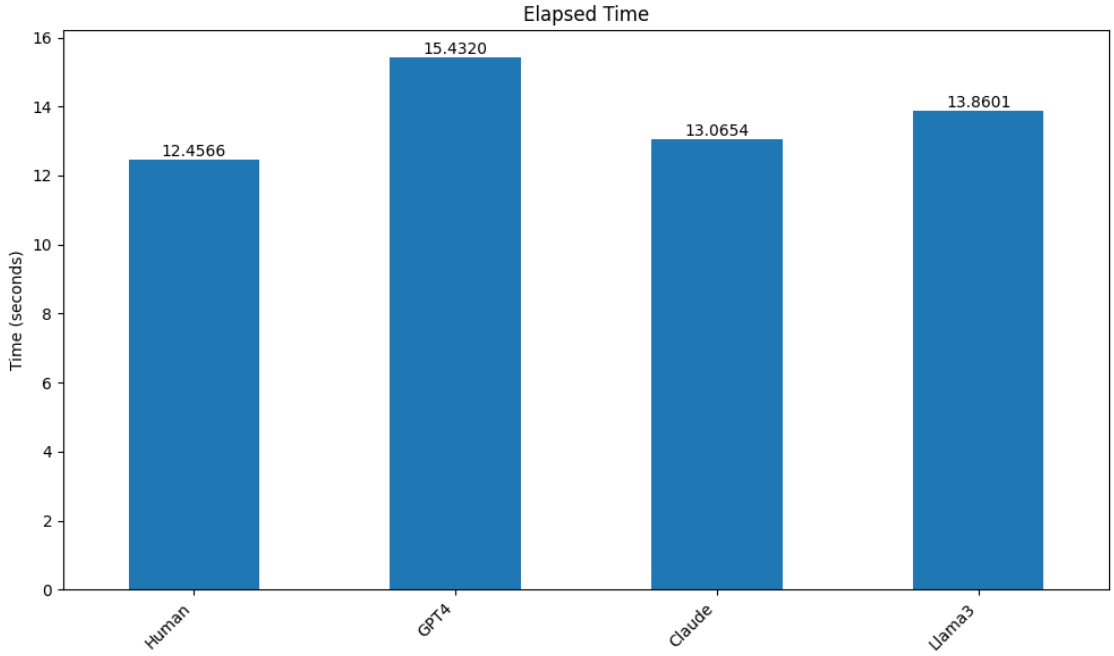

Figure 4.3 Execution time of object manipulation task script

4.1.1.4 Resource utilization

CPU usage was negligible (about 0%) across all models as well as the human-generated code, indicating the sample task does not require much processing power due to its being simple for the sake of experimenting in the scope of this study. Memory usage was consistently low, with minor variations. These results suggest that all models are well-optimized for memory efficiency, with only marginal differences in resource consumption.

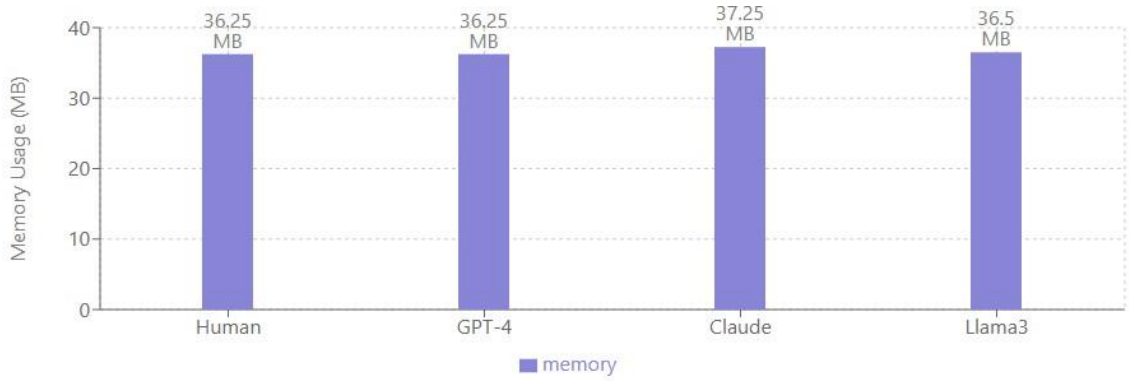

Figure 4.4 Memory usage in object manipulation task

4.1.2 Code complexity analysis

To assess the trade-off between performance and code complexity, three key metrics are analyzed; namely: cyclomatic complexity, lines of code, and parameter count.

Most models achieved a cyclomatic complexity of 2, indicating simple, straightforward logic. GPT-4o showed slightly higher complexity with a score of 3, potentially due to additional error handling or optimization attempts.

Interestingly, LLM-generated code was generally more concise than the human-generated code, with Llama 3 70B producing the most compact solution.

All models maintained a consistent parameter count of 2, suggesting a standardized approach to function inputs across different implementations.

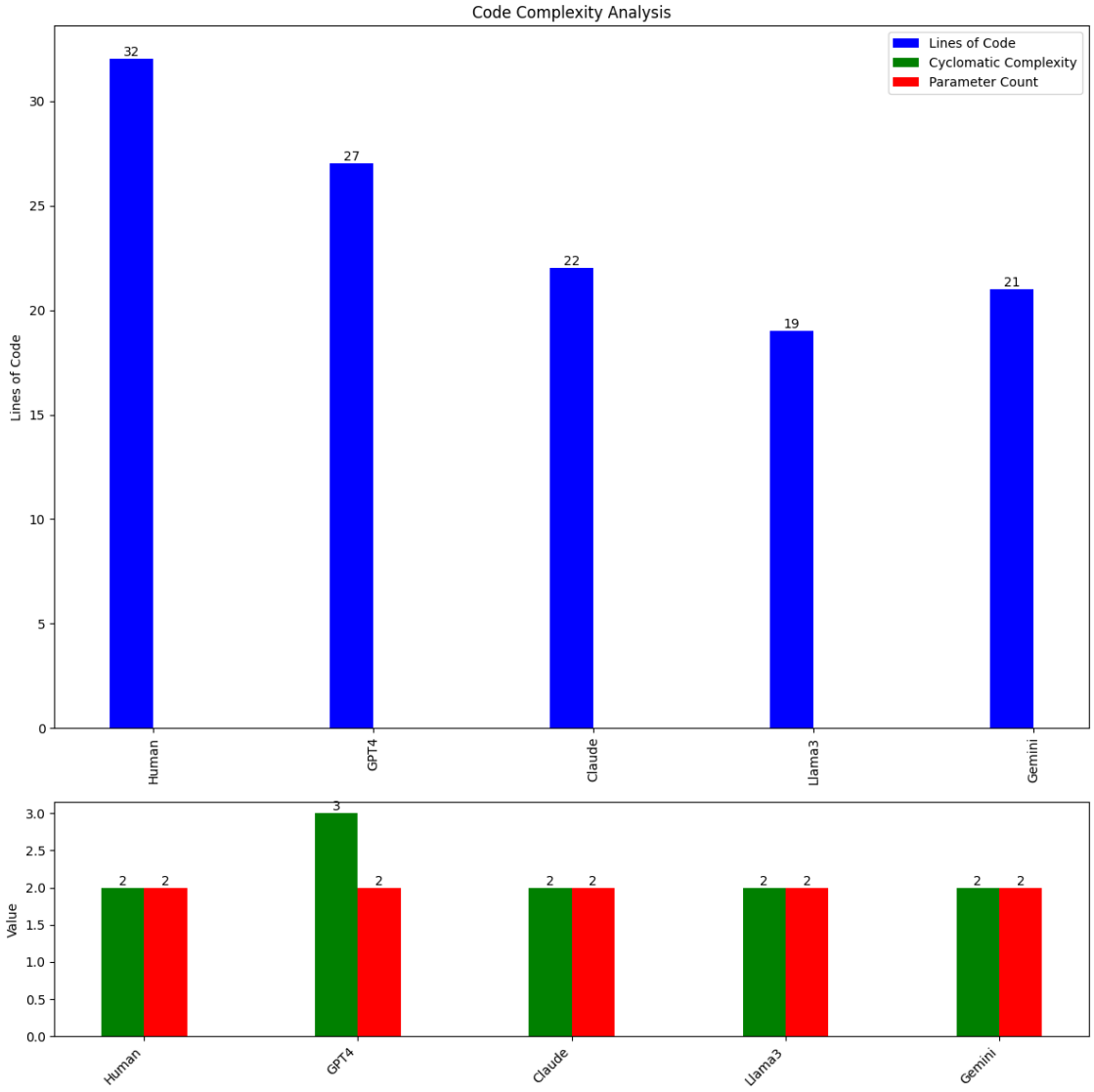

Figure 4.5 Code complexity analysis

4.2 Navigation Task

This section presents the findings for the comparative analysis of a navigation task implemented by different LLMs and human-generated code.

Table 4.2 Navigation task results summary

	Human	**GPT-4**	**Claude**	**Llama 3**	**Gemini**
Elapsed Time (sec)	310	14	293	12	14
Resources					
CPU (%)	5.91	4.90	4.28	3.73	4.20
Memory (MB)	19.90	19.59	19.33	19.00	19.33
Code Complexity Analysis					
Cyclomatic Complexity	4	5	7	4	4
Lines of Code	73	94	90	85	87
Parameter Count	4	0	4	0	4

4.2.1 Performance metrics

The analysis focuses on the following key performance metrics: overall execution time, and resource usage.

4.2.1.1 Overall execution time

The human-generated code and the code generated by Claude 3.5 achieved full navigation, whereas other LLMs caused the robot to spin around just after the beginning. That's why the elapsed time values are considerably lower than that of human and Claude 3.5 code pieces. Between those two, Claude 3.5 Sonnet has better resource utilization performance with a shorter execution time.

4.2.1.2 Resource utilization

The results indicate that the human-generated code showed the highest mean CPU usage (5.91% ± 0.86%), with the widest range (3.80% - 8.00%). Among the LLMs, GPT-4o demonstrated the highest average CPU utilization (4.90% ± 0.36%), followed by Claude 3.5 Sonnet (4.28% ± 0.40%), Gemini 1.5 Flash (4.20% ± 0.27%), and Llama 3 70B (3.73% ± 0.17%). Notably, Llama 3 70B exhibited the lowest and most consistent CPU usage across all systems, suggesting a potentially more efficient computational approach.

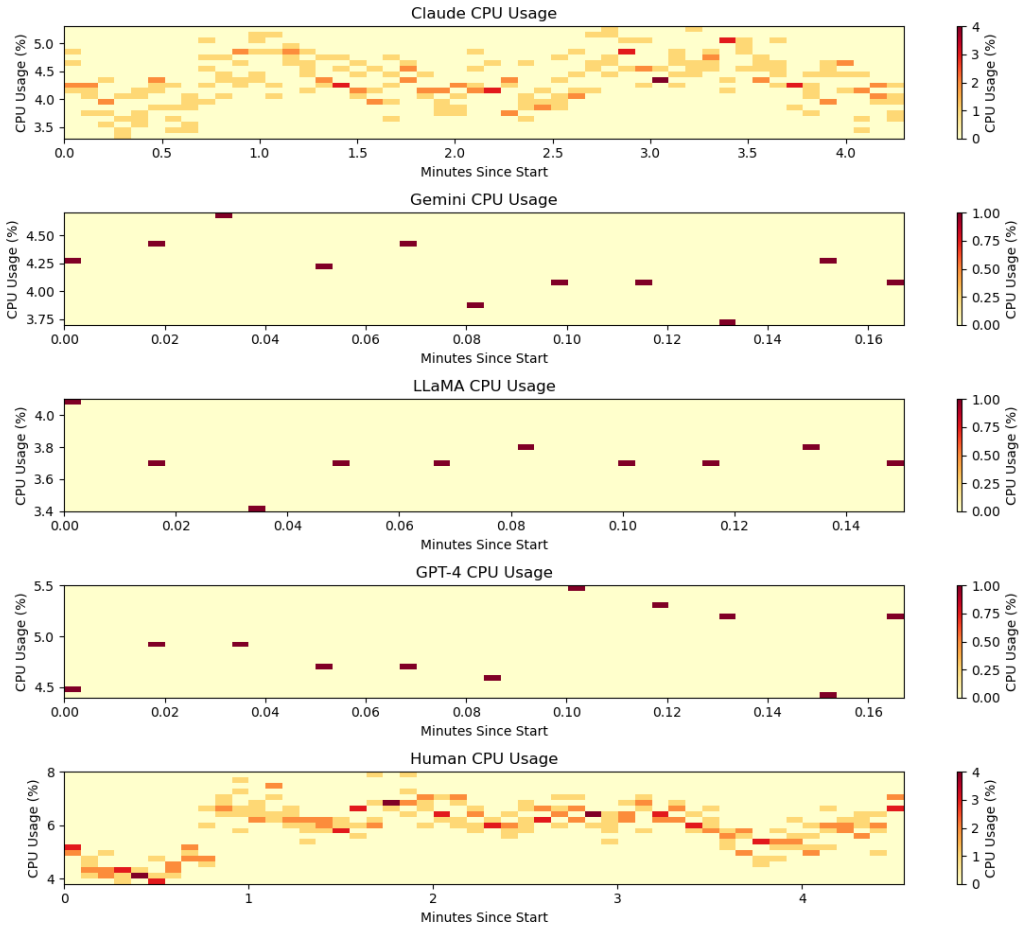

Figure 4.6 CPU usage heatmap

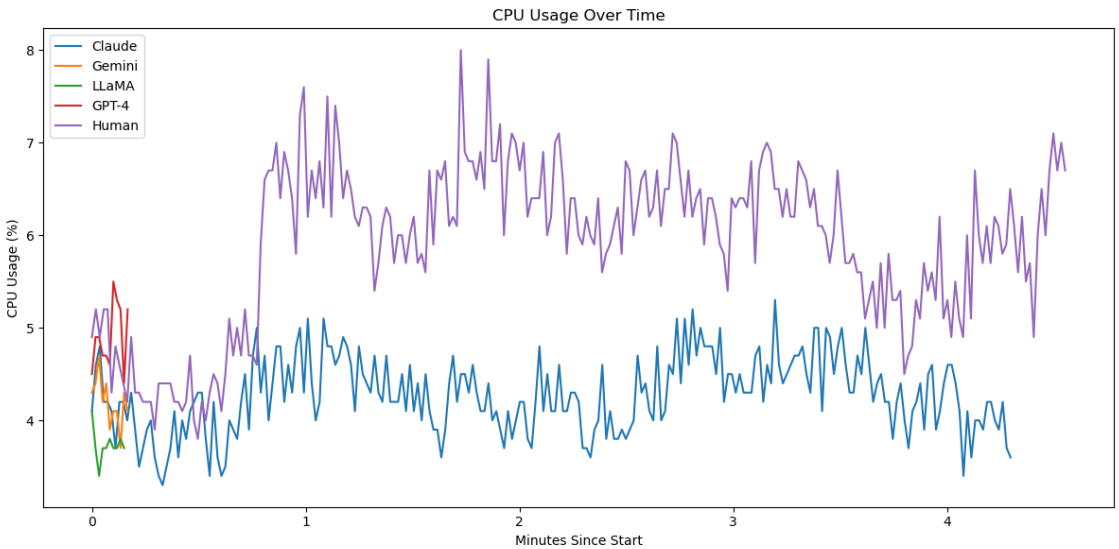

Figure 4.7 CPU usage over time

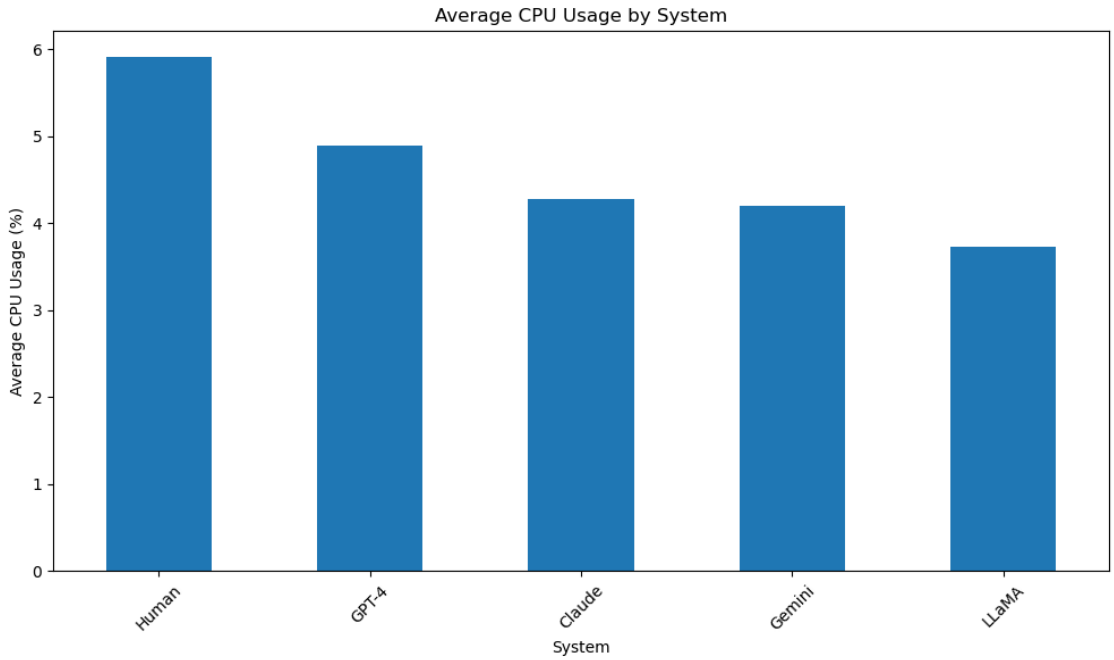

Figure 4.8 Average CPU usage

Memory usage patterns were remarkably consistent across all systems, with minimal standard deviations. The human-generated code showed the highest mean memory usage (19.90% ± 0.05%), marginally exceeding the LLMs. Among the LLMs, on the other hand, GPT-4o utilized slightly more memory (19.59% ± 0.03%) compared to Claude 3.5 Sonnet and Gemini 1.5 Flash (both at 19.33% ± 0.05%). Llama 3 70B displayed the lowest and most consistent memory usage (19.00% ± 0.00%), indicating a potentially more memory-efficient architecture or implementation.

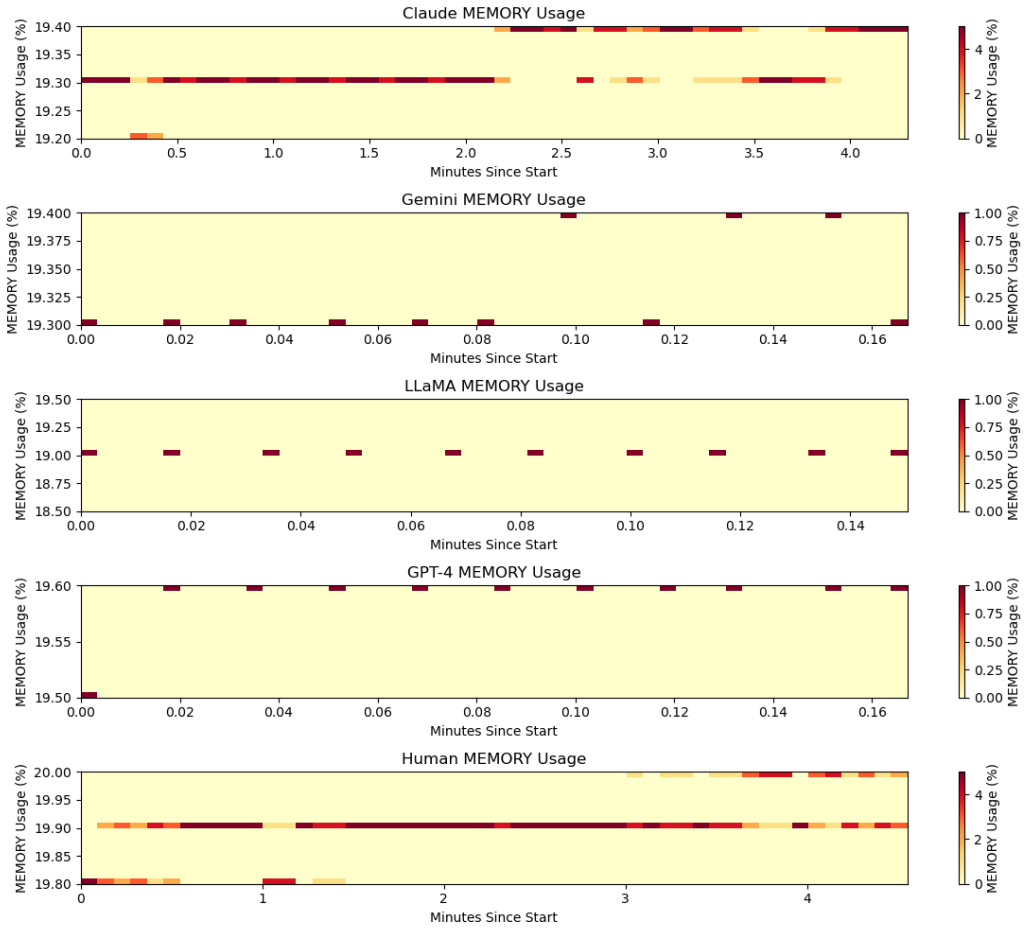

Figure 4.9 Memory usage heatmap

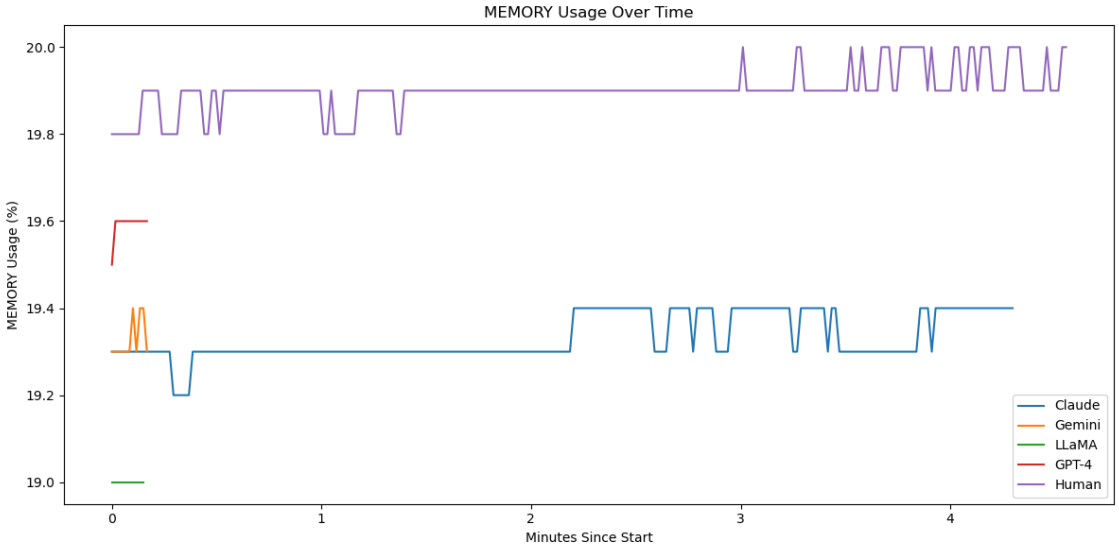

Figure 4.10 Memory usage over time

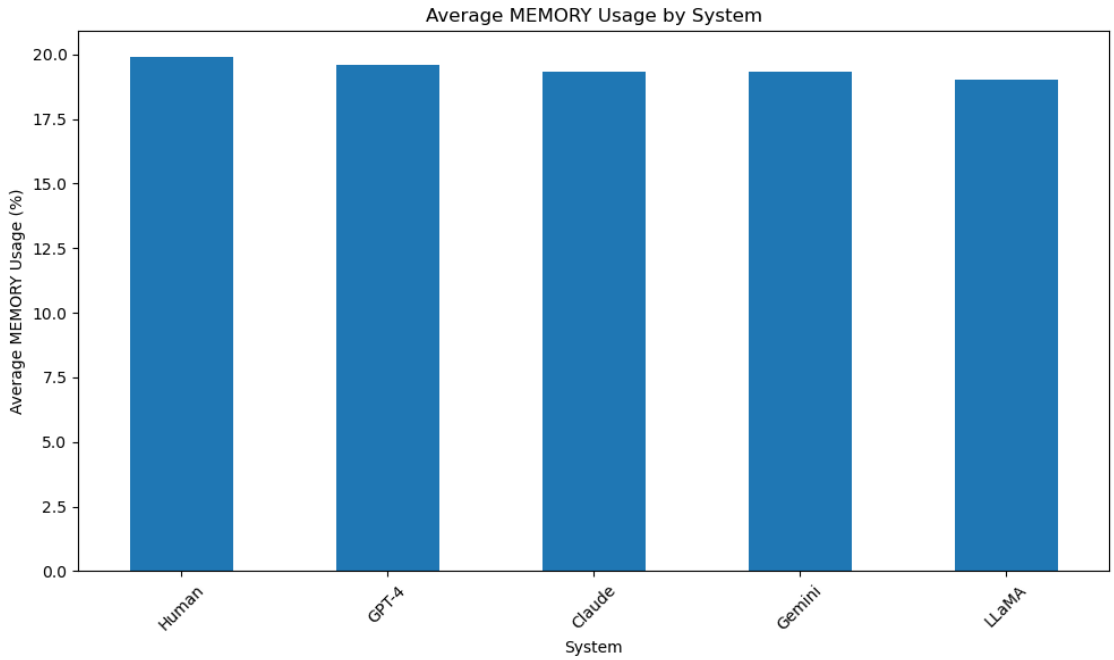

Figure 4.11 Average memory usage

4.2.2 Code complexity analysis

The charts indicate interesting patterns in code complexity across different AI models and human implementation. For the main function, Claude 3.5 Sonnet shows the highest cyclomatic complexity (7), significantly higher than others (2-3). GPT-4o and Llama 3 70B produced the longest code (54 and 52 lines respectively), while human generated and Gemini 1.5 Flash code had the shortest (27 lines each).

For the helper function, all implementations show similar complexity metrics, with nearly identical cyclomatic complexity (1) and parameter count (4). Gemini 1.5 Flash's code implementation is slightly more concise at 12 lines, compared to 13 for human generated and Claude 3.5 Sonnet code.

These differences highlight varying approaches to problem-solving and code generation among LLMs and the human-generated code.

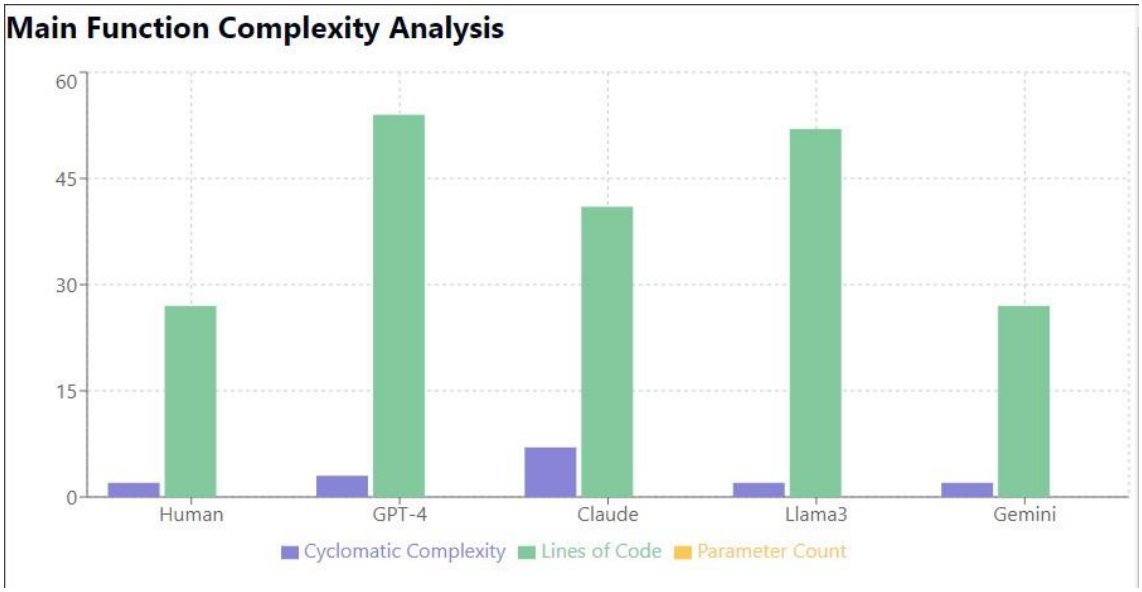

Figure 4.12 Main function complexity analysis

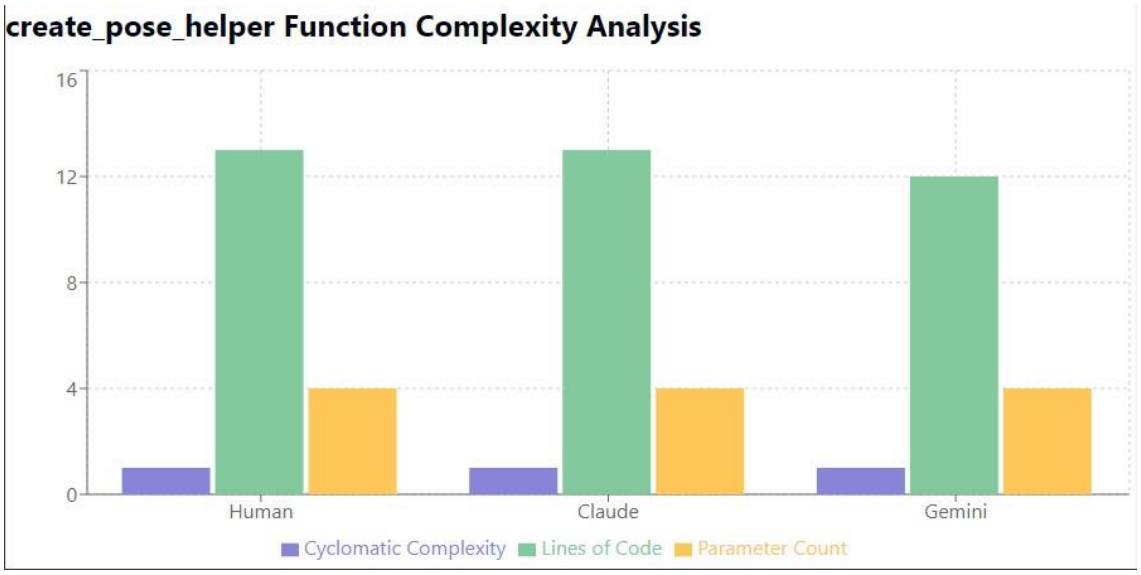

Figure 4.13 Helper function complexity analysis

4.3 Robot Description Generation Task

In this task, each LLM is given the same prompt of a request to generate a URDF file of a simple robot, which is a box-shaped construct on three wheels coupled with a LiDAR (Wang et al., 2020). The details for the robot's physical characteristics regarding its kinematics and dynamics are included in the prompt.

Each LLM except Gemini, the URDF file of which raised a parsing error and hence no robot showed up on RViz, was able to generate a URDF file that is eligible to run and provide a display on RViz.

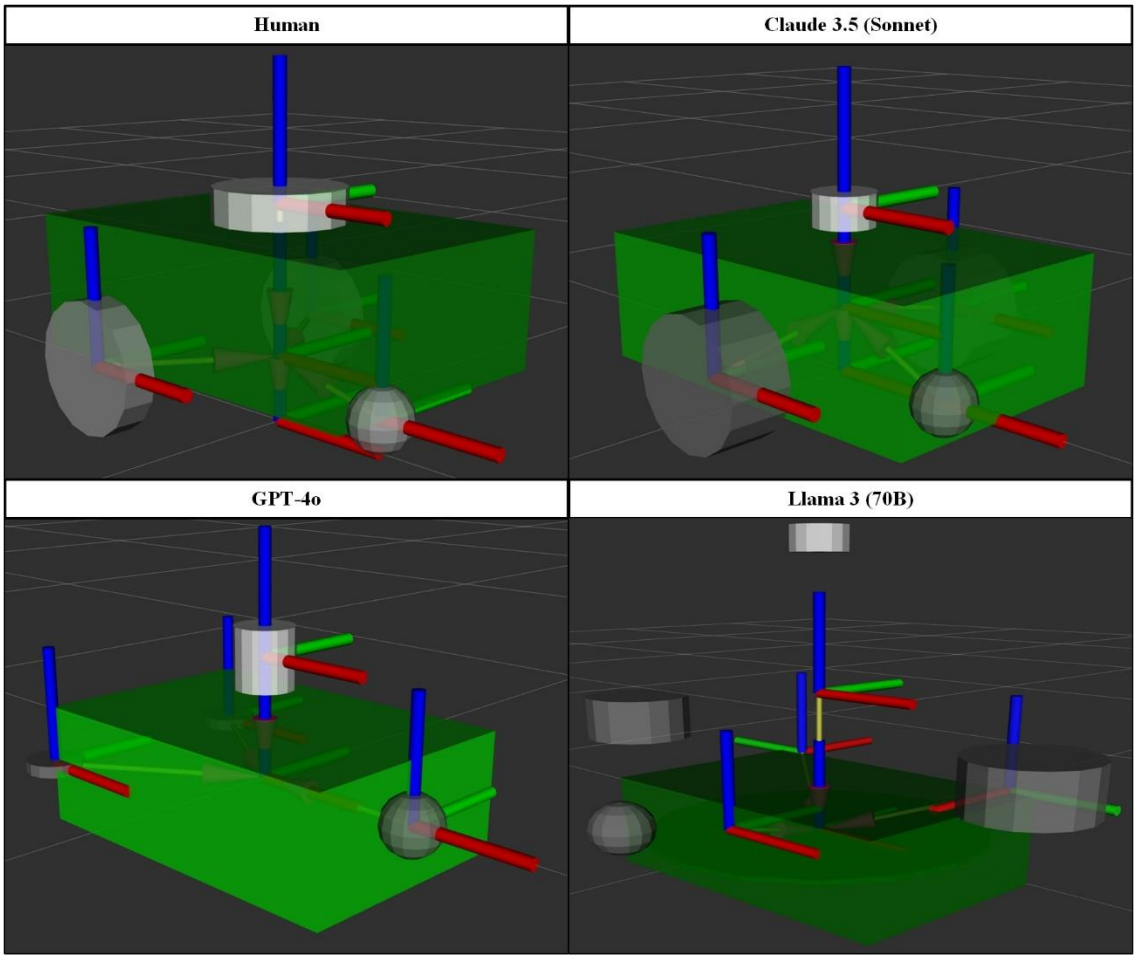

Figure 4.14 URDF generation task RViz demonstrations

The key parameters for evaluation in this task encompass several critical aspects of URDF file generation. These include the accuracy of robot dimensions, ensuring the generated file closely matches the specified physical characteristics. The correctness of joint types and configurations is crucial, particularly for the wheels and LiDAR components. Proper implementation of robot dynamics, including mass, inertia, and other physical properties as specified in the prompt, is essential for an accurate representation. The inclusion and correct positioning of the LiDAR sensor within the robot model is another vital factor. Syntactical correctness is paramount, as the ability to generate a URDF file that can be

parsed without errors directly impacts functionality. The visual representation in RViz serves as a key indicator of success, demonstrating how accurately the robot is displayed compared to the intended design. The number of attempts required by each LLM to produce a functioning URDF file provides insight into their efficiency and understanding of the task. Lastly, adherence to URDF file structure and conventions, including proper use of XML tags, attributes, and overall file organization, is necessary for creating a standard-compliant and easily interpretable robot description.

Claude 3.5 Sonnet, competent enough to provide the robot description file in the first try, had only the LiDAR's size different compared to that given in the URDF file written by a human. Since, there is no explicit mention for the size of the LiDAR, it is acceptable. GPT4o and Llama 3 70B, on the other hand, did take more than one trials to have a URDF file that can be run and display the robot of interest on RViz. It is also worth mentioning that out of these trio, only Claude 3.5 Sonnet was able to generate a URDF file that is acceptable. Others' URDF files was eligible to run; however, failed to meet expected robot kinematics and dynamics.

5. DISCUSSION

The integration of LLMs into robotics and software engineering has proved its worth, particularly in code generation support. On the other hand, their application to a thorough pipeline guidance, especially for complex simulations in environments like Gazebo, remains as a challenge. The results of this study, combined with recent research findings, highlight both the potential and limitations of LLMs in robotics application development.

Recent reviews of LLM-based code generation emphasize the models' strong code understanding and writing abilities, which have boosted developer productivity across various software engineering tasks (Wang & Chen, 2023). Be that as it may, experiences with ROS-based projects indicate a significant gap between LLMs' theoretical knowledge and the practical implications on robotics applications. This divergence is especially evident in LLMs' incompetence to provide error-free, end-to-end guidance for complex setups such as Gazebo simulations.

The experience with ROS-based projects introduced challenges similar to those confronted in ClassEval benchmark, which showed that LLMs are less performant on class-level tasks as to method-level generation (Du et al., 2024). While resolving errors, fixing one issue often led to a series of new problems. This marks the current limitations of LLMs in fully grasping the contextual subtleties and interdependencies within robotics systems. The intricacy of robotics applications, which oftentimes involve multiple interdependent code units and broader contextual dependencies, seems to be making things harder for LLMs, so they usually fail to fulfill expectations.

Although there are limiting factors, a hybrid approach of combining human expertise with LLM capabilities has produced promising results. The idea has been reinforced by first manually implementing and understanding Gazebo simulations for basic robotics tasks and subsequently using this knowledge to guide LLM-based script generation. By this way a more effective workflow could be established. Leveraging the strengths of LLMs in code generation while mitigating their weaknesses in system-level understanding resonates with the literature (Chang et al., 2023). Their framework, a self-iteration code

generation framework incorporating different software development roles, demonstrated significant improvements in code generation quality across various LLMs.

Regarding LLMs' varying performance for generating scripts based on basic instructions, it aligns with the literature findings. Experiences make it prominent that there is a need for strict human supervision and validation when using LLMs for specialized domains like robotics. Likewise, ChatGPT's performance in solving competition-level programming problems varied significantly, with a tendency to copy existing code for more challenging tasks (Yan et al., 2023).

Progress made in LLM applications in particular domains, like electronics system design, offer intuition for future developments in robotics. Improvements in Verilog code generation through fine-tuning LLMs with domain-specific dataset (Thakur et al., 2023), and a benchmarking framework for evaluating LLMs in Verilog code generation (Liu et al., 2023) suggest that similar strategies could be employed to boost LLMs' performance in code generation tasks for ROS-based projects.

Furthermore, the integration of specialized knowledge into the LLMs' reasoning process, the knowledge-aware code generation techniques enabled compelling progress in dealing with generalization problems (Huang et al., 2024). The very same idea could be valuable in robotics as well, where domain-specific knowledge is essential for effective code generation and problem-solving.

All in all, although LLMs have great potential as assistant tools in robotics development in the context of code generation and ideation, their current limitations put a stop to sole reliance on these systems for complex robotics applications. The development of a true AI assistant for robotics application development remains a challenging goal. More specialized LLMs trained on robotics-specific datasets to improve integration with simulation environments for real-time testing and feedback coupled with enhanced contextual understanding and error prediction capabilities may be took under advisement. Notwithstanding, the complexity of robotics systems calls for the understanding of the fact that human supervision will remain bottom-line in system design, troubleshooting,

and overall project management for the foreseeable future. A balanced approach, leveraging LLMs' strengths while compensating for their weaknesses with human expertise, appears to be the most effective strategy in the current technological landscape. As the field of AI in robotics continues to evolve, ongoing research into specialized training, knowledge integration, and iterative development processes may lead to more comprehensive and reliable AI assistants, potentially revolutionizing the process of robotics application development.

5.1 Conclusion

5.1.1 Object manipulation task

The findings reveal several interesting patterns:

Efficiency vs. Complexity: Human generated code has the best performance in terms of planning and execution time despite of not being the most concise. This implies that human expertise may trade off performance over code brevity.

AI Model Variations: Among AI models, Claude 3.5 Sonnet demonstrates a balanced performance, with competitive planning and execution times, and moderate code conciseness.

Trade-offs: Although it has slower planning times, GPT-4o still achieves success in a single trial, which indicates a potential trade-off between initial planning and overall reliability.

Resource Efficiency: All models demonstrated excellent resource management, with negligible CPU usage and minimal memory consumption, suggesting well-optimized implementations across the board.

The results do indicate that there is a potential for AI to assist developers in their work to design object manipulation algorithms. The performance data of the human generated code implies the significance of human expertise while the competent AI models, particularly Claude 3.5 Sonnet and Llama 3 70B is likely to enable promote the software development process. Yet, it is important to note that the object navigation task might have been such that it had more complexity and hence longer duration.

5.1.2 Navigation task

The resource utilization patterns reveal interesting distinctions between the AI models. GPT-4o, while showing higher resource requirements, may offer capabilities that justify this increased utilization. Claude and Gemini exhibited remarkably similar resource profiles, suggesting comparable computational approaches or optimizations. Llama 3 70B's consistently lower resource utilization across both CPU and memory metrics is noteworthy and warrants further investigation into its architectural efficiencies.

It is crucial to note the disparity in sample sizes between the systems. Claude 3.5 Sonnet and human interactions had substantially more data points (235 and 249, respectively) compared to GPT-4o, Gemini 1.5 Flash, and Llama 3 70B (11, 11, and 10, respectively). This difference is due to the fact that the codes generated by these LLMs do not result in a successful simulation run.

5.1.3 Robot description generation task

The performance of several LLMs in generating URDF files for a simple box-shaped robot with three wheels and a LiDAR sensor was evaluated in this study. The assessment criteria were established to include accuracy of robot dimensions, correct joint configurations, proper implementation of robot dynamics, accurate LiDAR positioning, syntactical correctness, visual representation in RViz, efficiency in producing a functioning file, and adherence to URDF conventions.

A relatively better performance was seen in the performance of Claude 3.5 Sonnet, by which an acceptable URDF file was generated on the first attempt with only minor deviations in LiDAR size. On the other hand, GPT4o and Llama 3 70B required multiple attempts to produce the corresponding files that could be run and displayed in RViz. Still, the specified robot kinematics and dynamics were not accurately represented in their outputs. As another example of varying capabilities of different LLMs in handling specialized tasks like URDF file generation, a parsing error was produced by Gemini's output, by which the robot could not be displayed in RViz.

These results highlight the potential of LLMs in automating complex tasks in robotics. They also show that there is room for improvement. Claude 3.5 Sonnet's performance is promising since the process of creating robot descriptions can be streamlined. However, it is indicated by the inconsistent results across different models that further refinement and specialized training may be necessitated to achieve reliable performance across a range of LLMs for such technical tasks.

5.2 Future Work

This study is focused on a comparative analysis of LLMs for code generation in specific ROS tasks; and expanding the scope to build a pipeline for the entire robotics development process is aimed as a future work. It will be such that natural language processing faculties of LLMs are utilized throughout the development lifecycle, from world creation and robot description to simulation setup and task execution. The main goal is to democratize robotics related studies by lowering the barriers for entry in the sense of both education and development. Future research should also explore the creation of specialized LLMs or fine-tuning existing models to better understand robotics-specific concepts and generate more accurate and comprehensive ROS code and configurations. Furthermore, investigating ways to improve LLMs' ability to provide contextually relevant instructions for simulation setup and execution would be crucial. More accessibility to newcomers and streamlining the process for experienced developers are expected to revolutionize how robotics projects are conceptualized and implemented.

REFERENCES

Al-Rashid Agha, R., Mahdi, Z., Sefer, M., & Hamarash, I. (06 2021). A ROS-Gazebo Interface for the Katana Robotic Arm Manipulation. *UKH Journal of Science and Engineering*, *5*, 26–37. doi:10.25079/ukhjse.v5n1y2021.pp26-37

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., … Wu, Y. (2023). PaLM 2 Technical Report. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2305.10403

Anthropic. (2024). The Claude 3 Model Family: Opus, Sonnet, Haiku

Austin, J., Odena, A., Nye, M. I., Bosma, M., Michalewski, H., Dohan, D., … Sutton, C. (2021). Program Synthesis with Large Language Models. *CoRR*, *abs/2108.07732*. Retrieved from https://arxiv.org/abs/2108.07732

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., … Kaplan, J. (2022). Constitutional AI: Harmlessness from AI Feedback. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2212.08073

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., … Liang, P. (2022). On the Opportunities and Risks of Foundation Models. *arXiv [Cs.LG]*. Retrieved from http://arxiv.org/abs/2108.07258

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020). Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1877–1901).

Chang, T., Chen, S., Fan, G., & Feng, Z. (2023). A Self-Iteration Code Generation Method Based on Large Language Models. *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, 275–281. doi:10.1109/ICPADS60453.2023.00049

Coleman, D., Sucan, I., Chitta, S., & Correll, N. (2014). Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. *arXiv [Cs.RO]*. Retrieved from http://arxiv.org/abs/1404.3785

D'Hollander, E. H., Danneels, E., Decorte, K.-B., Loobuyck, S., Vanheule, A., Van Kets, I., & Stroobandt, D. (2024). Exploring Large Language Models for Verilog Hardware Design Generation. *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 111–115. doi:10.1109/IPDPSW63119.2024.00034

Ding, Y., Zhang, X., Paxton, C., & Zhang, S. (2023). Task and Motion Planning with Large Language Models for Object Rearrangement. *arXiv [Cs.RO]*. Retrieved from http://arxiv.org/abs/2303.06247

Du, X., Liu, M., Wang, K., Wang, H., Liu, J., Chen, Y., … Lou, Y. (2024). Evaluating Large Language Models in Class-Level Code Generation. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. Presented at the <conf-loc>, <city>Lisbon</city>, <country>Portugal</country>, </conf-loc>. doi:10.1145/3597503.3639219

Girden, E. R. (1992). *ANOVA: Repeated measures*. Sage.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. doi:10.1038/s41586-020-2649-2

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2020). Measuring Massive Multitask Language Understanding. *CoRR*, *abs/2009.03300*. Retrieved from https://arxiv.org/abs/2009.03300

Huang, T., Sun, Z., Jin, Z., Li, G., & Lyu, C. (2024). Knowledge-Aware Code Generation with Large Language Models. *2024 IEEE/ACM 32nd International Conference on Program Comprehension (ICPC)*, 52–63.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

Iso/iec 25010. (2011). *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*.

Kam, H. R., Lee, S.-H., Park, T., & Kim, C.-H. (2015). RViz: a toolkit for real domain data visualization. *Telecommunication Systems*, *60*, 337–345. Retrieved from https://api.semanticscholar.org/CorpusID:6697099

Khokar, K., Beeson, P., & Burridge, R. R. (2015). Implementation of KDL Inverse Kinematics Routine on the Atlas Humanoid Robot. *Procedia Computer Science*, *46*, 1441–1448. Retrieved from https://api.semanticscholar.org/CorpusID:62376226

Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, *3*, 2149–2154 vol.3. doi:10.1109/IROS.2004.1389727

Li, Y., Shi, J., & Zhang, Z. (2024). An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering. *IEEE Access*, *12*, 53074–53087. doi:10.1109/ACCESS.2024.3385682

Liu, M., Pinckney, N., Khailany, B., & Ren, H. (2023). VerilogEval: Evaluating Large Language Models for Verilog Code Generation. *arXiv [Cs.LG]*. Retrieved from http://arxiv.org/abs/2309.07544

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, *7*(66), eabm6074. doi:10.1126/scirobotics.abm6074

Macenski, S., Martin, F., White, R., & Clavero, J. G. (2020, October). The Marathon 2: A Navigation System. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. doi:10.1109/iros45743.2020.9341207

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320. doi:10.1109/TSE.1976.233837

McKinney, W. (2011). *pandas: a Foundational Python Library for Data Analysis and Statistics*. Retrieved from https://api.semanticscholar.org/CorpusID:61539023

Mengacci, R., Zambella, G., Grioli, G., Caporale, D., Catalano, M. G., & Bicchi, A. (2021). An Open-Source ROS-Gazebo Toolbox for Simulating Robots With Compliant Actuators. *Frontiers in Robotics and AI*, *8*. doi:10.3389/frobt.2021.713083

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., … Zoph, B. (2024). GPT-4 Technical Report. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2303.08774

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., … Lowe, R. (2022). Training language models to follow instructions with human feedback. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2203.02155

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., … Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, *21*(1).

Sobell, M. G. (2015). *A Practical Guide to Ubuntu Linux (4th Edition)* (4th ed.). USA: Prentice Hall Press.

Stella, F., Della Santina, C., & Hughes, J. (06 2023). How can LLMs transform the robotic design process? *Nature Machine Intelligence*, *5*. doi:10.1038/s42256-023-00669-7

Thakur, S., Ahmad, B., Fan, Z., Pearce, H., Tan, B., Karri, R., … Garg, S. (2023). Benchmarking Large Language Models for Automated Verilog RTL Code Generation. *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–6. doi:10.23919/DATE56975.2023.10137086

Tola, D., & Corke, P. (2023). Understanding URDF: A Dataset and Analysis. *arXiv [Cs.RO]*. Retrieved from http://arxiv.org/abs/2308.00514

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., … Scialom, T. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2307.09288

Töberg, J.-P., & Cimiano, P. (2023). Generation of Robot Manipulation Plans Using Generative Large Language Models. *2023 Seventh IEEE International Conference on Robotic Computing (IRC)*, 190–197. doi:10.1109/IRC59093.2023.00039

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 30). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi:10.1038/s41592-019-0686-2

Wang, J., & Chen, Y. (2023). A Review on Code Generation with LLMs: Application and Evaluation. *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*, 284–289. doi:10.1109/MedAI59581.2023.00044

Wang, J., Wu, Z., Li, Y., Jiang, H., Shu, P., Shi, E., … Zhang, S. (2024). Large Language Models for Robotics: Opportunities, Challenges, and Perspectives. *arXiv E-Prints*, arXiv:2401.04334. doi:10.48550/arXiv.2401.04334

Wang, X., Pan, H., Guo, K., Yang, X., & Luo, S. (06 2020). The evolution of LiDAR and its application in high precision measurement. *IOP Conference Series: Earth and Environmental Science, 502*, 012008. doi:10.1088/1755-1315/502/1/012008

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software, 6*(60), 3021. doi:10.21105/joss.03021

Xie, Y., Yu, C., Zhu, T., Bai, J., Gong, Z., & Soh, H. (2023). Translating Natural Language to Planning Goals with Large-Language Models. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2302.05128

Yan, D., Gao, Z., & Liu, Z. (2023). A Closer Look at Different Difficulty Levels Code Generation Abilities of ChatGPT. *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1887–1898. doi:10.1109/ASE56229.2023.00096

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., … Wen, J.-R. (2023). A Survey of Large Language Models. *arXiv [Cs.CL]*. Retrieved from http://arxiv.org/abs/2303.18223