Human robot Interaction

# Introduction

Recent advancements in Artificial Intelligence (AI), particularly in the domain of Natural Language Processing and Large Language Models (LLMs), have demonstrated remarkable progress across various AI-related fields. The convergence of increased computing power and comprehensive datasets has led to impressive capabilities in LLMs, including their ability to generate computer code. While writing code has traditionally been considered a complex task due to its syntactical and logical requirements, LLMs have shown a surprising aptitude for code generation.

However, the tendency of LLMs to occasionally produce non-factual content (a phenomenon known as "hallucination") raises questions about their reliability in generating functional code. Given the professional demeanor and confidence with which LLMs provide responses, it can be challenging to discern the true effectiveness of LLM-generated code without rigorous testing.

This project investigates the efficacy of LLMs in generating Java code for an abstract robot class using one-shot learning techniques. We developed a comprehensive "system instruction" document that detailed the environment, requirements, and expectations for the LLMs. This document also provided examples of expected inputs and outputs to guide the LLMs in generating appropriate code.

# Project Overview

## Objectives

The primary objectives of this study were to:

1. Assess the ability of LLMs to generate functional Java code for robot control tasks.

2. Compare the performance of three different LLMs in this specific domain.

3. Evaluate the effectiveness of one-shot learning techniques in guiding LLMs for code generation.

## Methodology

We employed a set of 20 carefully crafted prompts, each designed to elicit code generation for specific robot control tasks. These prompts were presented to three distinct LLMs:

1. ChatGPT 3.5

2. Gemini 1.5 Pro

3. Claude 3.5 Sonnet

The generated code was then subjected to a standardized testing framework to evaluate its functionality and adherence to the specified requirements.

# Environment and System Design

## Abstract Robot Class

We implemented a closed environment centered around an abstract robot class. This class defined the capabilities of robot instances and methods for manifesting these abilities. Key features of the robot class include:

- Movement along x, y, and z axes in 3D space

- A gripper mechanism for manipulating objects

- Collision detection to prevent logical errors

## Object Class

An abstract object class was created to represent items that the robot could interact with in the 3D space. This class included:

- Coordinates (x, y, z) to represent the object's position

- Methods for updating and retrieving the object's position

## Collision Detection

To ensure logical consistency and prevent errors, we implemented a collision detection system. This system includes:

- A `boolean gripperIsOpen` flag to track the gripper's state

- A `checkCollision()` method that verifies the robot's intended movement against existing object positions

- Exception throwing when a collision is detected, allowing for more robust error handling and testing

# System Instructions and Prompt Engineering

## System Instruction Document

A crucial component of this project was the development of a comprehensive system instruction document. This document served as a guide for the LLMs, providing:

- Detailed descriptions of the robot and object classes

- Specifications for movement and object manipulation

- Examples of expected input prompts and corresponding output code

- Guidelines for error handling and edge cases

The system instruction document was designed to facilitate one-shot learning, allowing the LLMs to generate appropriate code based on a single, detailed set of instructions.

## Prompt Engineering

Significant effort was invested in crafting effective prompts that would elicit the desired code generation from the LLMs. Key aspects of our prompt engineering approach included:

- Varying levels of complexity in the tasks described

- Clear and concise language to avoid ambiguity

- Inclusion of specific requirements (e.g., "do not return to origin") to test the LLMs' attention to detail

# Testing Methodology

## Test Suite Design

We developed a comprehensive test suite using JUnit to evaluate the generated code. The test suite included:

- Assertions to verify the final positions of both the robot and manipulated objects

- Checks for adherence to specific instructions (e.g., not returning to origin when specified)

- Collision detection tests to ensure the generated code respected the physical constraints of the system

## Standardized Evaluation

To ensure a fair comparison between the LLMs, we:

- Used identical prompts for all three models

- Applied the same test suite to all generated code

- Evaluated both the robot's final position and the positions of any manipulated objects

# Results and Analysis

## Performance Comparison

The results of our evaluation revealed significant differences in performance between the three LLMs:

1. Claude 3.5 Sonnet:

- Passed: 19 tests

- Failed: 1 test

- Notable for its high success rate and adherence to instructions

2. Gemini 1.5 Pro:

- Passed: 12 tests

- Failed: 8 tests

- Showed improved performance over older models but struggled with complex object manipulations

3. ChatGPT 3.5:

- Passed: 4 tests

- Failed: 16 tests

- Demonstrated difficulty in following the detailed instructions provided

## Analysis

- ChatGPT 3.5 often failed to adhere to the specific instructions outlined in the system instruction document.

- Gemini 1.5 Pro struggled with more complex scenarios, particularly those involving multiple object manipulations or precise positioning.

- Claude 3.5 Sonnet's single failure was related to a task involving object manipulation at different heights, suggesting a potential area for improvement in spatial reasoning.

# Discussion

## Implications for LLM-assisted Robotics

The results of this study suggest that recent LLM models, particularly Claude 3.5 Sonnet, show promise in generating functional code for robot control tasks. This capability could significantly accelerate the development of robotic systems by providing programmers with a powerful tool for rapid prototyping and problem-solving.

## Limitations and Considerations

While the results are encouraging, it's important to note several limitations:

- The study focused on a specific, abstract robot environment and may not fully generalize to all robotic systems.

- The effectiveness of the generated code relies heavily on the quality of the system instructions and prompts provided.

- Real-world robotic applications often involve more complex environments and safety considerations not addressed in this study.

## Future Directions

Future research in this area could explore:

- The performance of LLMs on more complex robotic tasks and environments

- The impact of different programming languages on LLM code generation capabilities

- Strategies for improving LLM performance in areas where they currently struggle, such as spatial reasoning in 3D environments

# Conclusion

This study demonstrates the potential of Large Language Models in generating functional code for robot control tasks. The significant performance differences observed between LLM versions highlight the rapid pace of advancement in this field.

Claude 3.5 Sonnet's impressive performance, with a 95% success rate, suggests that LLMs are becoming increasingly reliable tools for code generation in specific domains. However, the variability in results across different LLMs underscores the importance of choosing the right model and providing clear, comprehensive instructions.

While LLMs show promise as aids in robotics programming, it's crucial to emphasize that they should be viewed as tools to augment human programmers rather than replace them. The complexity of real-world robotic systems, safety considerations, and the need for human oversight in critical applications mean that human expertise remains essential.

As LLM technology continues to evolve, we can expect further improvements in their code generation capabilities. This progression may lead to more efficient development processes in robotics and other fields, potentially accelerating innovation and reducing the time required to bring new robotic solutions to market.

In conclusion, while LLMs have shown significant potential in generating code for human-robot interaction, their application should be approached with a clear understanding of their capabilities and limitations. As the field advances, continued research and rigorous testing will be crucial in harnessing the full potential of LLMs in robotics and beyond.