HAW HAMBURG

Masterarbeit

Michael Babic

Translating User Intent into Robot Instructions: Performance Analysis of Integrating LLMs in an Event-Driven Robotics Control System

Faculty of Engineering and Computer Science
Department Computer Science

Michael Babic

Translating User Intent into Robot Instructions: Performance Analysis of Integrating LLMs in an Event-Driven Robotics Control System

Master thesis submitted for examination in Master´s degree
in the study course *Master of Science Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Martin Becke
Supervisor: Prof. Dr. Bettina Buth

Submitted on: 10. März 2025

Michael Babic

Thema der Arbeit

Translating User Intent into Robot Instructions: Performance Analysis of Integrating LLMs in an Event-Driven Robotics Control System

Stichworte

Large Language Models, Robotik, Robotik-Steuerungssysteme, Integrationsprozess, LoRA

Kurzzusammenfassung

Neben den Fortschritten in der Robotiksteuerung durch Reinforcement Learning spielen Large Language Modelle (LLMs) eine zunehmend wichtige Rolle bei der Verbesserung der Interaktion zwischen Robotern und Nutzern. Die Aufgabe der LLMs hier besteht darin, eine ausführbare Liste von Anweisungen auszugeben, die die Absichten des Nutzers für den Roboter repräsentieren. Diese Anweisungen werden im Robotik-Steuerungssystem ausgeführt, sodass Nutzer Roboter über natürliche Sprache steuern können. Diese Arbeit schlägt ein Event-Basiertes Transaktionssystem vor, in dem Anweisungen an die Roboter in Form von Transaktionen ausgeführt werden. Diese integrieren Priorisierung und Ressourcen-Leasing, um eine sichere und parallelisierbare Ausführung zu ermöglichen. Das System umfasst eine benutzerzentrierte Oberfläche über einen Chatbot, wo der Nutzer zunächst generierte Anweisungen auswählt. Diese werden auf einem digitalen Zwilling evaluiert und getestet, um eine sichere Ausführung zu gewährleisten, wodurch sich das System besonders für den Einsatz von LLMs eignet.

Der Hauptbeitrag dieser Arbeit ist ein verallgemeinerter Integrationsprozess zur Einbindung von LLMs in ein solches Robotik-Kontrollsystem. Die Leistung dieses Ansatzes wird anhand mehrerer LLMs evaluiert, darunter Google's Gemma2, Meta's Llama3.1, Mistral's Nemo und Microsoft's Phi3. Der Prozess beginnt mit der Definition des Evaluationssets und der Evaluationsmetriken, gefolgt von der Definition und Bewertung von Systemprompts sowie der Verbesserung der Token-Generierung. Anschließend erfolgt das Feintuning der LLMs mittels Low-Rank Adaptation (LoRA), die Einführung eines Few-Shot-Prompting-Ansatzes sowie die abschließende Bewertung der Generierung mehrerer Antworten.

Die Ergebnisse zeigen, dass dieser neuartige Ansatz zu einer erfolgreichen Integration der genannten LLMs führt. Die durchschnittliche Bewertung aller Modelle auf dem gesamten Evaluationsset, einschließlich komplexer Aufgaben, konnte von 36% auf 70% des Maximalwerts gesteigert werden. In der Kategorie der grundlegenden Nutzung, die die typische Anwendung durch Nutzer widerspiegelt, wurde eine Verbesserung von 45% auf 86% erreicht.

Michael Babic

Title of Thesis

Translating User Intent into Robot Instructions: Performance Analysis of Integrating LLMs in an Event-Driven Robotics Control System

Keywords

Large Language Models, Robotics, Robotics Control Systems, Integration Process, LoRA

Abstract

Next to the advancements in robotics control through reinforcement learning, Large Language Models (LLMs) start to play an important role in improving robot-user interactions. The task for LLMs here is to output a parsable list of instructions, representing the user's intention for the robot, that will be executed in the robotics control system, allowing users to control robots through natural language. This thesis proposes an event-driven transaction system, where instructions to the robots are executed in the form of transactions that incorporate priority and resource acquisition to enable safe and concurrent access. It features a user-centric frontend over a chatbot, where the user must first choose translated instructions, which are evaluated and tested on a digital twin to ensure safe execution, making it suitable for the use of LLMs.

This thesis's main contribution is a novel and generalized integration process for integrating LLMs in such a robotics control system and evaluates the performance of this approach using multiple LLMs, including Google's Gemma2, Meta's LLama3.1, Mistral's Nemo, and Microsoft's Phi3. The process starts with defining the evaluation set and evaluation metric, defining and evaluating system prompts, and improving token generation, followed by fine-tuning the LLMs using Low Rank Adaptation (LoRA), proposing a few-shot prompting approach, and finishing with the evaluation of multi-response generation.

The findings show that this novel approach results in a successful integration of the mentioned LLMs, going from 36% to 70% of the maximum score averaged across all LLMs on the total evaluation set, including complex tasks, with an improvement from 45% to 86% on the basic usage category, which resembles the user's basic usage of the system.

Contents

List of Figures

1 Introduction

AI in robotics is a rapidly growing field, with research in the direction of autonomous robots, sensor improvements, and predictions through reinforcement learning. These directions tackle the sense and act paradigms in the traditional robotic architecture sense-plan-act [39], where the planning is done through task scheduling with given goals for the robot. For robotics in the home assistant and personal service fields, the user interacts with the robot to provide goals. This user-robot interaction is the main topic in said field. With the major growth of natural language processing through large language models (LLMs), research has been initiated in the direction of using LLMs as the interface for interacting with the robot by translating user intents into instructions.

In this work, an event-driven robotics control system is presented with an interface for controlling it over LLMs. The robot control problem is viewed as an automatization problem, where robot behavior is created through concatenating instructions. These instructions are managed through a transaction system, which allocates needed resources for them and manages their state and execution based on priority and attributes. A transaction abstracts the robot itself as a resource, leasing it for execution and then freeing it afterward. A trigger system allows for reactive behavior, making the robot react to environmental signals. The proposed robotics control system allows for parallel user interaction without interference, making it suitable for controlling robots over LLMs. A digital twin is included to validate instructions and test them based on the current state of the environment before execution in the real world.

A chat interface is provided in which the user can ask an LLM to perform a task, which takes the role of the robot and outputs selectable instruction lists. If one is accepted by the user, it is sent to the robotics control system, where transactions are derived from the instructions and executed.

A generalized integration guideline is provided for optimizing LLMs for the translation task, ensuring statistical soundness and reproducibility. This integration process can be easily replicated and used to evaluate other LLMs.

As a proof of concept, multiple LLMs from different vendors are evaluated. Microsoft's latest model, Phi3, Meta's LLaMA 3.1, Google's Gemma2, and Mistral's Nemo are assessed on the task of translating user intent to instructions for the robotics control system. These models are selected based on their performance in the Hugging Face Open LLM Leaderboards and their popularity [12].

Structure

The thesis is structured as follows:

Current research is presented in chapter 2, followed by an explanation of how it is extended with this approach. In chapter 3, the robotics control system is introduced, followed by an explanation of the approaches for the robotics control system and integration process in chapter 4. Chapter 5 presents the experiments, each containing their goal, an expectation, their structure, their execution containing the results, a discussion, and the conclusion. The experiments involve following the integration approach steps by identifying a system prompt, optimizing the models, developing and evaluating few-shot prompts, and conducting a multi-response temperature analysis. With this final result, the feasibility of the integration process for improving LLMs for user-robot interaction in a robotics control system is concluded in chapter 6.

2 Related Work

The rapid growth of LLMs results in increased usage of them for human interaction. While popular benchmark suits are used for evaluating their performance, work has to be done to measure their performance in user-centric human-AI collaboration tasks, as presented in [45].

Robot task planning is also the subject of extensive research. [25] uses ChatGPT-3.5 Turbo for real-time path planning in complex environments and parses the LLM's output as tasks to be allocated and executed.

[52] discusses the applicability of LLMs in generating goals from natural language by generating a planning domain definition language (PDDL [13]). A PDDL planning problem contains a domain that includes the environment, object relations, constraints, available actions, and the problem to be solved. The PDDL is then used by algorithms like Fast Downward [16] to generate an executable plan.

[21] uses LLMs for a multi-robot task allocation system, where the users instructions are decomposed into tasks that are then distributed through coalition formations of multiple robot groups and allocated to them.

[20] creates behavior trees for robotic tasks using LLMs. The generated behavior trees are validated in their XML format and executed on the robot, with a main focus on navigation and exploration tasks combined with an arm on the robot for picking up and transporting objects.

[27] presents a planning framework, where an LLM generates a sequence of executable actions for a robot to follow. They do so by making the LLM generate a full action sequence, which is checked given the environment state for validity. If the plan is valid, it gets executed. If not, they fall back to a step-by-step plan, where the LLM generates multiple possible actions for one step, and the most suitable by their environment model is executed. After that, they repeat the process until reaching the goal.

[50] proposes a safe and efficient task planning framework for generating constrained plans in a given environment using LLMs. While translating natural language into a plan, they make the LLM generate multiple explanations and translate each one into a semantic task plan. With their generated set of plans, they check for equality of them. They choose the most frequently repeated plan for execution. While creating each plan, a constrained algorithm checks the validity of the plan and corrects it so that each one is applicable to the current environment.

[46] considers task planning through constrained LLM prompt schemes that generate executable action sequences to deal with LLM hallucinations. They also propose an exception-handling module that checks that LLM-generated action sequences are admissible in the current environment. In contrast to the here presented approach, which requires the user's acceptance, this approach directly executes the user's request.

Other research also directly addresses the topic of hallucinations in LLM-powered code generation [28]. Their study gives five primary categories of hallucinations in LLM code generation and proposes a benchmark including hallucination recognition and mitigation tests. Their conclusion indicates the need for further research in this area and the current challenges faced by LLMs in detecting and mitigating hallucinations during code generation.

With this work, an expansion on previous research is aimed for by utilizing a robotics control system that incorporates reaction-based instructions and a digital twin. The list of tested LLMs is expanded, and their performance in creating instructions based on user input is compared using a multi-response chat platform that allows the user to accept or reject generated instructions. A structured integration guideline is provided to evaluate and improve LLMs, particularly by examining them in both their finetuned and non-finetuned states, with the goal of evaluating the integration effectiveness and the performance in translating user intent into robot instructions.

3 Robotics Control System

In this section, a general overview is presented of how LLMs interact with robots, with a particular focus on the papers from chapter 2 in which the generated plans are executed on robots to categorize their approaches and identify the problems associated with these approaches.

The robotics control system is then presented, along with a description of how these problems are addressed. The usage of LLMs is subsequently introduced in chapter 4.

3.1 Overview

[22] provides an overview of LLM usage in robotics. It groups these usages in 4 major categories:

- Communication, where the two tasks are converting natural language into semantic representations for the robots to process and language generation to communicate with the user.
- Perception, about building a perception of the environment for the robot to act in.
- Planning, about generating plans for the robot to execute.
- Control, where LLMs directly control the robots.

The main focus of this work is combining the planning and communication aspect to generate plans for the robot to execute through natural language from a user. While communication is about creating semantic representations, planning often integrates robot systems that handle the execution of these plans [22].

[21] has a simple task executor that sends commands to a simulation named ai2thor [23], which is created mainly for research.

[50] runs the generated plans step by step in the PyBullet simulation [8], which is a physics engine created mainly for research.

[25] uses the Robotic Operating System (ROS) [34] to execute the plan by the used LLM, using the ROS navigation stack.

[24] and [46] also use the ROS navigation stack. For [24], the generated path is converted to waypoints, which are checked in the ROS map for validity and then executed. [46] translates the user's input from the LLM into smaller tasks, which are functions that call ROS services. Their exceptional handling module to combat LLM hallucinations is called when translated tasks are not found in their task dictionary, and the LLM gets prompted with that hint to correct the output.

[27] executes their planned actions on a simulation of the Franka Panda robot, where its actions are implemented in ROS. Their validation checking is done by a geometric algorithm that, e.g., checks if the robot can reach another object that is included in the plan.

[20] executes actions with the BehaviorTree.CPP [5] library. The behavior tree contains sequential actions, state-dependent branching, fallbacks, parallel branches, and more. The actions are executed in ROS2 [29], the later version of ROS.

These ROS implementations concentrate on the operation of a single robot and build on the ROS navigation stack to validate their plans. The specific purpose of these approaches limits their scope, focusing either on only driving or only grabbing objects. As the output from the LLM is directly executed, they do not have a feedback loop or safety guards that check if the translated plan matches the user's intent and only rely on the validity of the plan by either geometric algorithms [27] or by checking if the plan has a valid path for navigation tasks using the ROS map.

Behavior trees add a layer of abstraction to allow building complex behavior out of smaller, simpler behaviors. But they are not suited for parallel executions of multiple trees for the same agent or controlling multiple robots, as it is designed for specifying the behavior of one agent [6].

Problems:

These are the identified challenges of the mentioned approaches:

- The plans are validated only logically, which may not fully capture environmental dynamics.
- The generated plan execution can not be done concurrently or changed while executing.
- The execution of translated plans is done directly after translation on the robot.

The first challenge is addressed by using a digital twin of the robot or robots that runs the generated plan fully in a simulation. Available resources needed for the execution are checked, and the plan is initially run in the simulation, capturing environment dynamics. Upon successful execution in the simulation, the plan is executed on the real robot.

The second challenge, concerning the execution of a plan, is approached by modeling the task execution problem in the form of a transaction system. This system allows for the same depth in creating complex behaviors from simpler ones but also manages the extension of behavior and resources in a concurrent and distributed environment, as opposed to the previously mentioned behavior trees.

To address the last challenge, a multi-response user interaction is proposed, which requires the user to select one choice specifically. In this way, a safety guard is provided to prevent unintended actions caused by LLM mistranslations. More on that is discussed in chapter 4.

3.2 Transaction System

In this section, the transaction system is presented, the core of this work's robotics control system. It is designed to manage shared resources in concurrent and distributed environments. It provides a flexible interface for resource access, such as for robots, allowing coordinated interaction among multiple programs, and is inspired by the automatization system Simatic S7 [15] in resource handling, resource access, and task execution.

On ROS, multiple programs can publish commands directly to a robot or other actors, but as more use cases arise, especially those requiring prioritization, this approach results in hard-coded, complex state machines. E.g., if one program navigates the robot to a destination while another prompts it to approach and greet a person, the robot would receive conflicting movement instructions, leading to unintended behavior. This system manages resource control to address these issues.

3.2.1 System Design

Each executable action is called a task. A task defines what resources it needs for execution, as in [15]. Driving to a location or rotating, e.g., are tasks that expect the availability of a robot base, or talking would need to use the audio playback. A resource system manages the available resources and their states, as well as providing a way to lease and free them for managing concurrent access.

In this proof of work, a resource is defined as the smallest unit that is used to create a behavior. For a robot that can drive and talk, e.g., the robot base and audio playback are resources. Resources themselves define if they can be shared or are exclusive. Resources from one robot are grouped as one robot, and when multiple parts are needed, the ones from one group are preferred. This design allows for two tasks to use both parts of a robot in parallel.

Tasks can be chained into a task list, which gets executed sequentially, passing through the state of the last task in the form of events.

Tasks are executed through transactions. A transaction contains lists of task lists, which are executed concurrently. A transaction can either be queued or running, visualized in fig. 3.1. Outside requests can remove a queued transaction, or the transaction system can start it automatically when enough resources are available. A running transaction can be aborted or removed, either internally if a task aborts its execution or when the

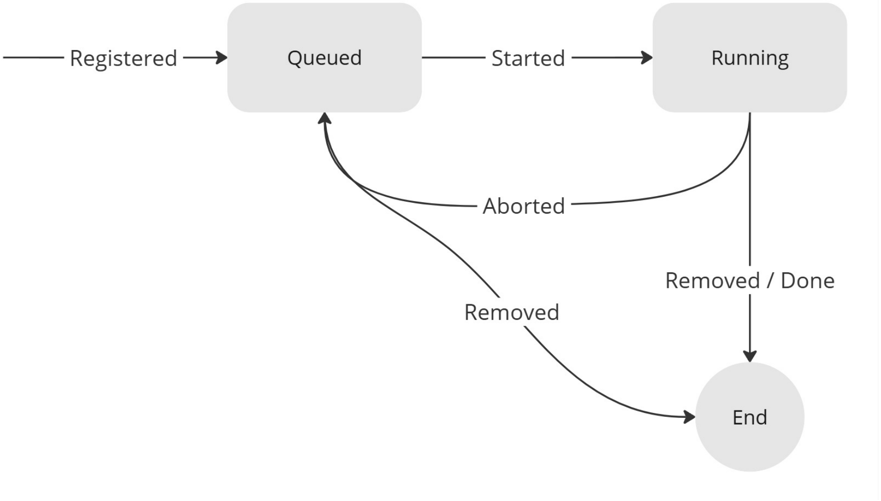


Figure 3.1: Transaction Lifetime.

transaction system stops it for taking over resources for a higher priority one. It can also finish cleanly if all tasks in each task list succeed. Transactions are aborted automatically when a leased resource becomes offline, meaning that a heartbeat was not received for too long a duration. To manage that, the transaction system caches the state of the resource system and subscribes to updates of it. It then sets abort signals to transactions that depend on a now offline resource. It also checks if a resource is leased by a transaction that is no longer available when, e.g., the communication of freeing a lease failed.

The transaction system manages the execution of transactions based on their priority.

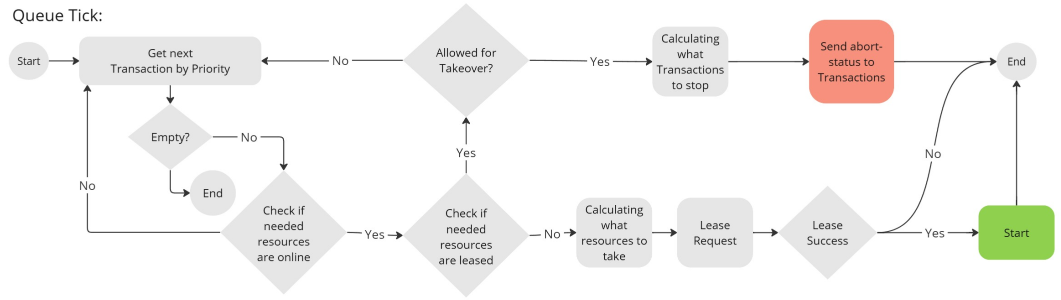

Queue Tick:
Start
Get next Transaction by Priority
No
Allowed for Takeover?
Yes
Calculating what Transactions to stop
Send abort-status to Transactions
End
Empty?
No
End
No
Check if needed resources are online
Yes
Yes
Check if needed resources are leased
No
Calculating what resources to take
Lease Request
Lease Success
No
Yes
Start

Figure 3.2: Transaction Queue.

Figure 3.2 shows how registered transactions are executed in the transaction queue. Queue ticks run only when no transaction is in an aborted or removed state; otherwise, the cleanup is awaited.

Transactions are sorted by their priority. The highest-prioritized transaction is then checked to see if its needed resources are currently online. If they are not, the next transaction is looked at. If some needed resources are leased, it is checked if they can be taken over and calculates the outcome of taking them over with the following priorities:

- Least amount of dangling resources.
- Lowest priorities.
- Least weight per attribute if set. An attribute for driving to a location, e.g., is the shortest path for each resource applicable to that distance.

This means that a cross product of transactions is made so that resources can be allocated to cover needs in a way that maximizes their utilization. If matching combinations are found, their priorities are checked, and if these priorities match, an attribute is checked if it is set. Otherwise, the first option is selected. When all required resources for a transaction are available and free, they are simply taken. A set attribute is used to decide between multiple choices.

A trigger system is implemented to build reactive behavior. Triggers are defined as functions that are called on specified events to initiate or manage transactions.

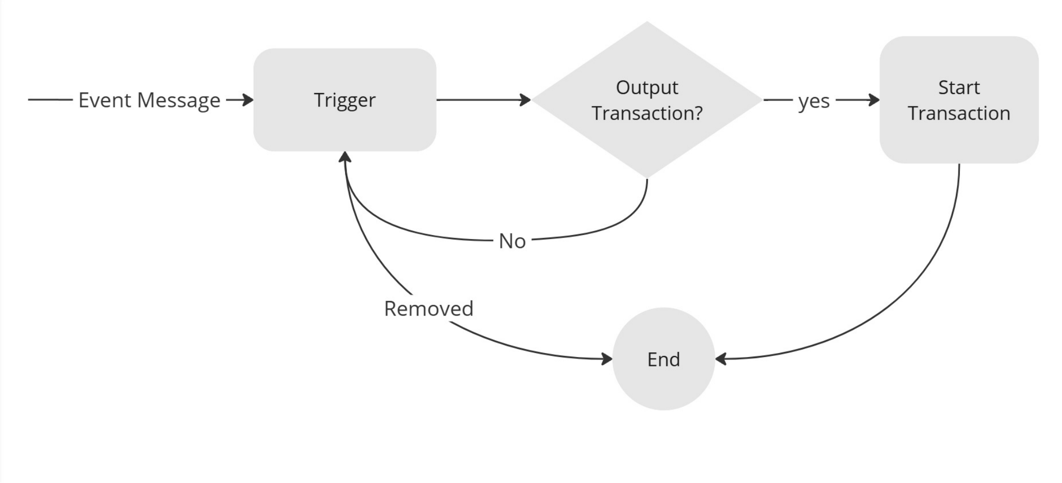


Figure 3.3: Trigger States.

Figure 3.3 shows how triggers work. They are registered to react to an event of a wanted type. The event serves as their input, and they have the option to return a transaction, which the transaction system will automatically register. It gets, after sending a transaction, removed from the trigger system but can be registered to be repeated any amount of times or stay permanently active.

As in [15], with the combination of transaction and trigger, tasks can run in a cycle, periodically, event-driven, and priority based.

The transaction system is designed for easy extensions of new tasks and resources in multiple programming languages using the Thrift IDL [2].

Figure 3.4 shows the structure of the system.

The whole system communicates over an internal publish and subscribe event bus. The parameter server handles setting and getting shared parameters. The resource system has an interface for leasing and freeing resources, as well as managing the registration

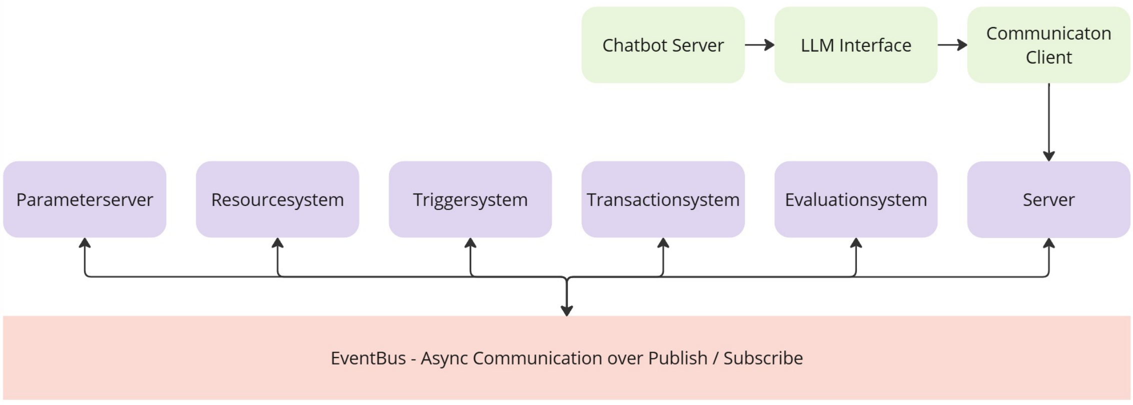


Figure 3.4: Robotic System Overview.

and connection to resources over a heartbeat. The trigger system manages the registration, deregistration, and execution of triggers. The transaction system does the same for transactions. The evaluation system has an interface for evaluating the validity of transactions and triggers through a simulation. The server is the entry point for outside access, implementing a REST API for accessing public functionalities of the other nodes and a publish-subscribe broker for events over a websocket. The communication client implements the counterpart of the server's API and is written in Rust and Python but can be implemented in each language that is supported by Thrift. The LLM interface and chatbot server will be presented in chapter 4.

The architecture of the system components ensures modularity, scalability, and asynchronous interaction between subsystems over the internal event bus. It allows for asynchronous execution, enabling decoupled components to react dynamically to new events. The resource system enforces structured access to resources, preventing conflicts and allowing safe concurrent operations for our transactions.

The transaction and trigger system allows for complex behavior out of small behaviors like the behavior tree but also allows for concurrent extension and usage of behaviors managed by the transaction system. When combined with the safety of execution provided by the evaluation system, it creates a system that is well-suited for use with LLMs.

3.2.2 Proof of Concept

In this section, the used robot in this work is shown with a visual example of how to create complex behaviors out of the building blocks, the tasks, and triggers.

Robot

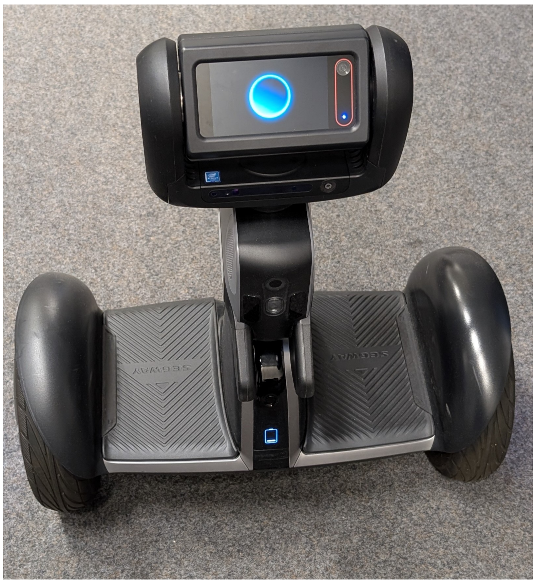


Figure 3.5: Picture of the Loomo Segway.

The robot used in this work is the Loomo Segway, as seen in fig. 3.5. It is a two-legged self-stabilizing Segway that can drive, rotate, talk, move its head, and has a touchscreen on its head. It also has multiple sensors, like an RGBD camera, a face cam, a fish eye cam, an IMU, and more [37].

In this proof of concept of the robotics control system, the Loomo provides the following resources for the resource system:

- LoomoBase: An exclusive resource for driving and rotating.
- LoomoVoice: An exclusive resource for talking.
- LoomoHead: An exclusive resource for rotating the head.
- LoomoSensors: A shared resource for using the sensors.

- LoomoDisplay: An exclusive resource for using the touchscreen.

This separation is mainly based on different API access points of the Loomo SDK [37], abstracting each part of the robot as a resource. Since sensors are published instead of used, these are abstracted as a shared resource. The last resource, the LoomoDisplay, abstracts the use of a touchscreen interface on the Loomo's head, which is implemented for user interaction.

The following tasks are implemented for the Loomo to be used in the transaction system again based on the Loomo SDK:

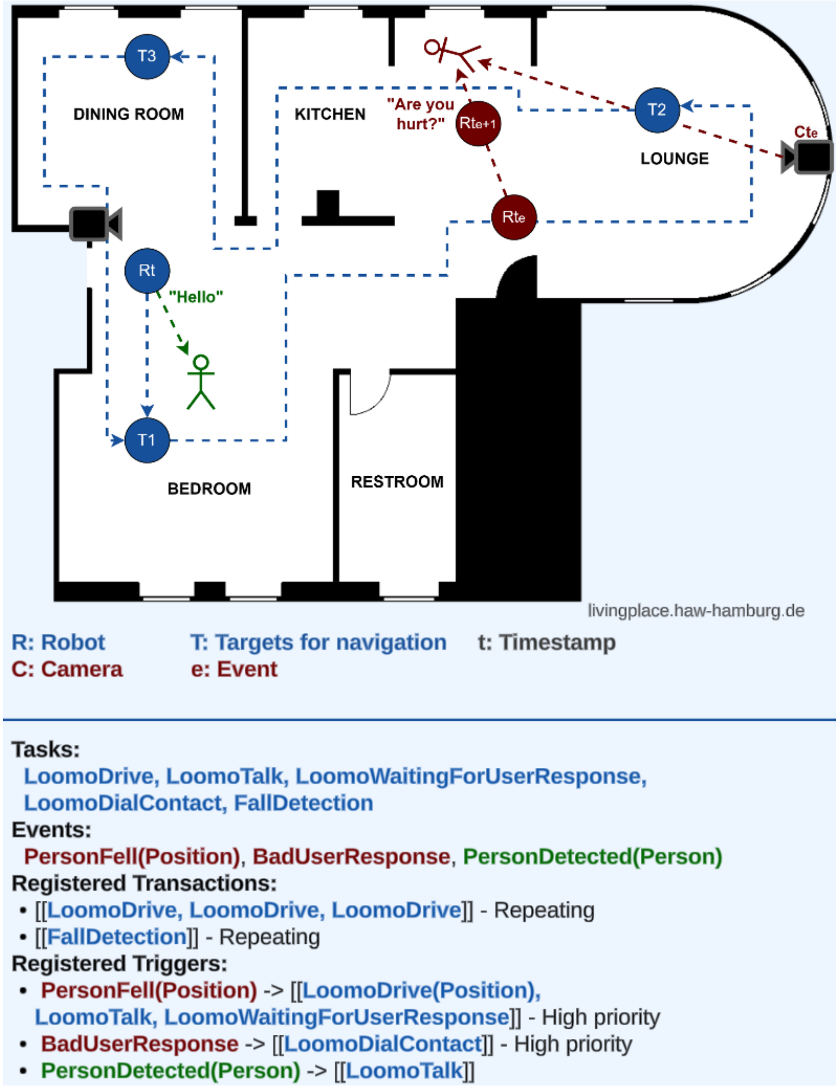
- LoomoDrive(Coordinate): Making the Loomo drive to a coordinate, either a named one, a local one or a global one. Needs the LoomoBase resource.
- LoomoRotate(Degree): Making the Loomo rotate a specific degree. Needs the LoomoBase resource.
- LoomoTalk(Text): Making the Loomo talk a specific text. Needs the LoomoVoice resource.
- LoomoRotateHead(pitch, yaw): Making the Loomo rotate his head to a specific pitch and yaw. Needs the LoomoHead resource.
- LoomoWaitingForUserResponse(AwaitedResponse): Making the Loomo wait on his touchscreen for user input. Needs the LoomoDisplay resource.

The behavior of the last task, waiting for a user input, is again based on a custom implementation of having buttons on the touchscreen of the Loomo head display for user interaction.

For each task, a ROS launch file is implemented, which defines a group of processes executed together. The transaction system then starts and closes these launch files for each task while executing a transaction.

Example

Here, an example use case is shown with added tasks of making the Loomo call for help and a fall-detection camera.



R: Robot
C: Camera
T: Targets for navigation
e: Event
t: Timestamp

Tasks:
LoomoDrive, LoomoTalk, LoomoWaitingForUserResponse, LoomoDialContact, FallDetection
Events:
PersonFell(Position), BadUserResponse, PersonDetected(Person)
Registered Transactions:
- [[LoomoDrive, LoomoDrive, LoomoDrive]] - Repeating
- [[FallDetection]] - Repeating

Registered Triggers:
- PersonFell(Position) -> [[LoomoDrive(Position), LoomoTalk, LoomoWaitingForUserResponse]] - High priority
- BadUserResponse -> [[LoomoDialContact]] - High priority
- PersonDetected(Person) -> [[LoomoTalk]]

Figure 3.6: Example automatization using the Transaction System.

Figure 3.6 shows an example behavior built with the transaction system. The Loomo starts on 'Rt' and sees a person through his face cam, which publishes a 'PersonDetected' event. A registered trigger responds to this event, initiating a transaction to greet the person. The robot itself is instructed to drive to the locations on 'T1', 'T2', and 'T3' in order in a repeating transaction. A camera running a fall detection sees a person falling in the kitchen and publishes a 'PersonFell' event containing the position of the fell person. The Loomo, now on 'Rte', reacts to that event with a high priority. Transaction to drive to that position, talk with the person, and wait for a response. If the person doesn't respond in time or signals help, a 'BadUserResponse' event instructs the Loomo to dial for help.

4 Approach

In this chapter, the multi-response approach is presented in detail and how it is used with the LLMs to control the robotics control system.

In this approach, the LLM generates one or multiple responses to a user query, on which he can choose one, addressing the challenge found in section 3.1 of directly executing LLM responses.

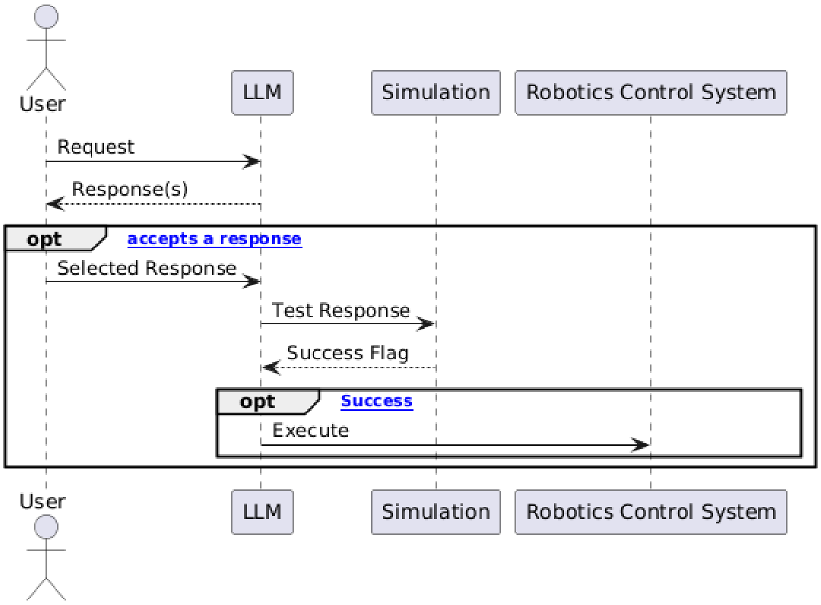


Figure 4.1: Robotics Control System Interface. The user's intent is translated into robot instructions through an LLM. If the user accepts a response, it gets sent to a simulation to test if it is executable. If it is, it gets executed on the real robot.

This feedback system, visualized in fig. 4.1, reduces the problem of having LLM hallucinations for the robotics control system since the user accepts a sequence instead of it being executed directly. To further address security concerns, the execution is first done in a simulation with a digital twin of the robot to check if it gets executed correctly, to take environment dynamics into account.

At the end of this section, what needs to be evaluated for this thesis to reason about the multi-response pattern will be listed, and an overview of the evaluation approach will be shown. The evaluation will be done in chapter 5.

4.1 LLM Interface

The task of the LLMs is to generate a list of instructions in the form of a JSON list to be executed by the robotics control system. JSON is used due to its support of the used programming language Rust [1] and the ease of parsing, generating, and reading it. The LLM Interface, seen in fig. 3.4, is the entry point for the LLM's text and handles the conversion from the JSON into trigger and transactions. It filters out non-valid JSON and checks if the instructions and trigger are applicable in the current environment.

An intermediate vocabulary is used for the LLM interface, which the LLMs need to generate JSON from, instead of generating transactions or triggers directly. This is done to keep a separation between the instructions that the LLMs generate and the tasks that the robotics control system uses. It allows creating instructions that do not map to only one task but to multiple tasks and enables extending or modifying the functions of the robotics control system without the need to modify the LLM inference.

The LLM interface also derives a more readable sentence from the parsed instructions for the user, from which he can choose, e.g.:

```
[{"EventBased":{"event":"PersonDetected",
"instructions":[{"LoomoDrivesTo":{"NamedLocation":"kitchen"}}]}}]
```

to:

If a person was detected: Loomo reaches the kitchen

The second functionality of the LLM interface is to execute a chosen generated sequence, which first gets evaluated by the evaluation system, and on success on the real robot.

4.2 Chatbot

The chatbot is a user interface for translating user requests into transactions and triggers.

For the website, the Chainlit framework is used [7], allowing for a user-friendly UI with a simple implementation. Here, the backend only needs to implement an optional on-start function and an on-message callback while having one state possible per user or chat. Multiple users can chat simultaneously with the chatbot and send instructions to the transaction system.

The chatbot can be configured with an environment containing information about the system prompt and a matched evaluation function, for example, evaluating the translated instructions on a digital twin or allowing them to be published directly to the transaction system.

The default environment uses a digital twin of two Loomo Segways in the living room apartment, as seen in fig. 3.6. The LLM takes the role of the Loomo Segway and converts user instructions to a list of event-based or direct instruction lists.

The chatbot server can be configured with a specific LLM, a local one, or an external one like ChatGPT.

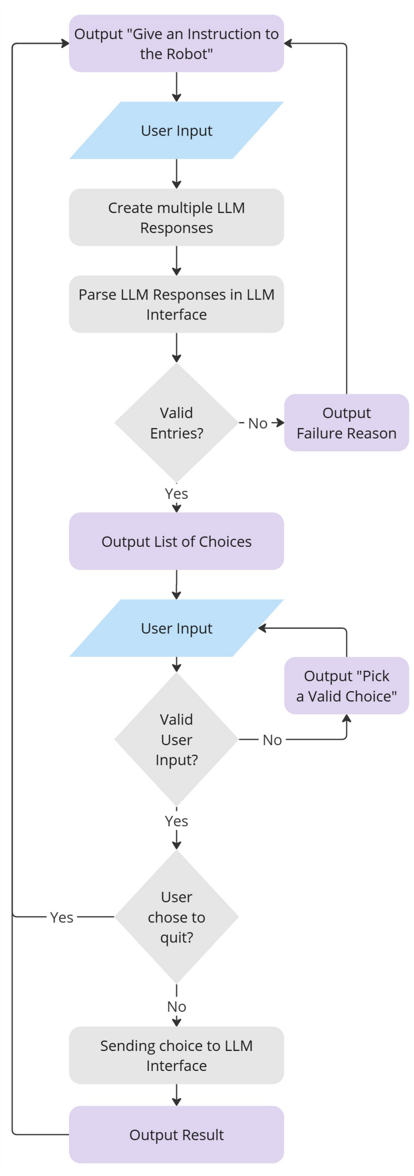

Output "Give an Instruction to the Robot"
User Input
Create multiple LLM Responses
Parse LLM Responses in LLM Interface
Valid Entries?
No
Output Failure Reason
Yes
Output List of Choices
User Input
Output "Pick a Valid Choice"
Valid User Input?
No
Yes
Yes
User chose to quit?
No
Sending choice to LLM Interface
Output Result

Figure 4.2: Chatbot Behavior Diagram.

It runs a simple state machine in the backend, using the LLM interface to communicate with the transaction system, seen in section 4.2. The initial state prints a simple message telling the user to give an instruction to the robot. The user input is then sent to the LLM multiple times with a configurable temperature variable to create multiple possible responses. The results are sent to the LLM interface to filter out inapplicable ones. When there are no valid LLM responses left, an error message will be printed out, and the state is set to the initial state. If there are choices, the user is instructed to pick one or go back to the initial state by interrupting with the 'q' choice. Wrong inputs will result in a short message that the input was false, followed by staying in the choosing state. When the user picks a choice, the result is sent to the LLM interface, which handles the evaluation and execution.

4.2.1 Chatbot Interaction

In this section, an interaction example with the chatbot is provided.

Preriquisites

The transaction system and all its components are running.

The two Loomo Segways are also started, and a node runs that registers them to the transaction system and sends a heartbeat for the active robots. For the map and navigation, a navigation stack is run for the two Loomos in the living room apartment. In this example, simulated Loomos are used, but the navigation map is a 1 to 1 mapping of the real apartment.

The chatbot is also started with a dual Loomo environment for evaluating the instructions on the digital twin and runs the LLM Llama 3.1 for translating the users requests in the 8B parameter version [11].

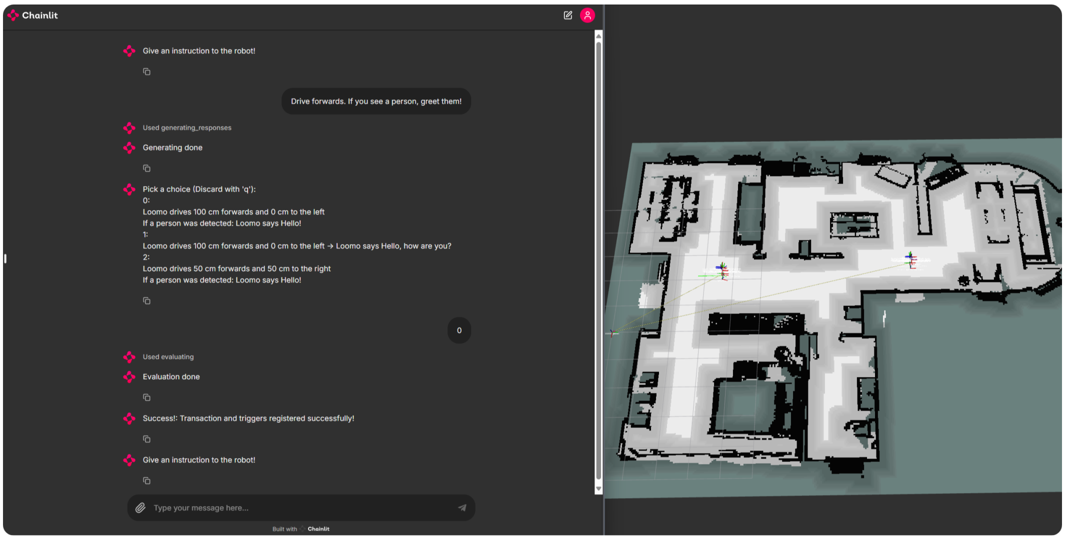
Experiment

The chatbot website is open, and the robot gets instructed to drive forward and, on seeing a person, to greet them.

This first instruction is visualized in fig. 4.3. On the left side, the chat UI is shown, and on the right side, the resulting visualized view of the robots states in the navigation map.

Here, a list of choices is generated to pick from. After choosing the first choice, it gets evaluated and executed, as seen by the green line in the navigation view. He drove a meter forward.

Another example is shown in fig. 4.4. Here, the chat was continued with asking him to go to the kitchen. The LLM now has only one distinct choice generated, which is picked.



Give an instruction to the robot!

Drive forwards. If you see a person, greet them!

Used generating_responses

Generating done

Pick a choice (Discard with 'q'):
0:
Loomo drives 100 cm forwards and 0 cm to the left
If a person was detected: Loomo says Hello!
1:
Loomo drives 100 cm forwards and 0 cm to the left → Loomo says Hello, how are you?
2:
Loomo drives 50 cm forwards and 50 cm to the right
If a person was detected: Loomo says Hello!

0

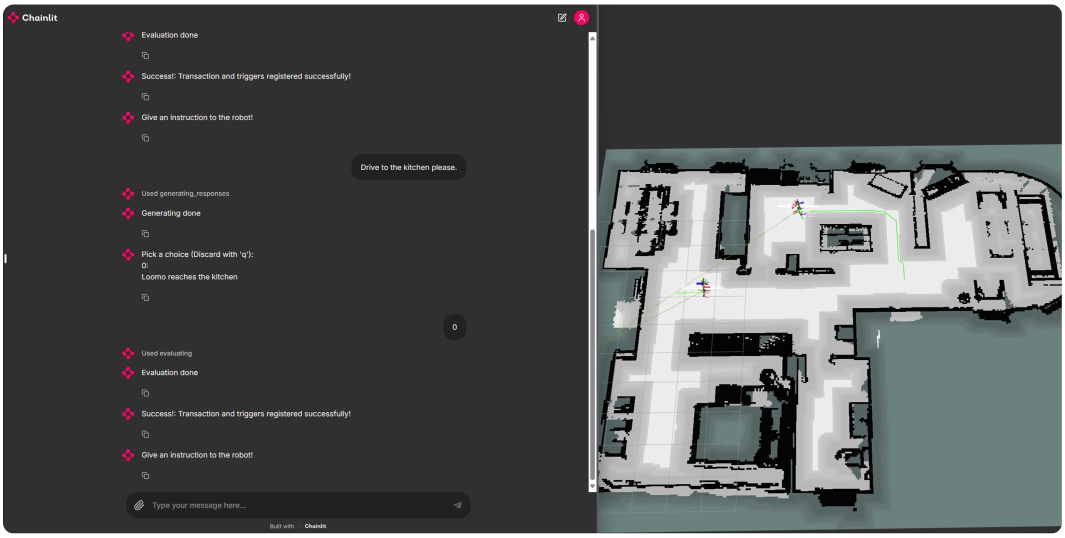
Used evaluating

Evaluation done

Success!: Transaction and triggers registered successfully!

Give an instruction to the robot!

Figure 4.3: Chatbot interaction example, first instruction.



Evaluation done

Success!: Transaction and triggers registered successfully!

Give an instruction to the robot!

Drive to the kitchen please.

Used generating_responses

Generating done

Pick a choice (Discard with 'q'):
0:
Loomo reaches the kitchen

0

Used evaluating

Evaluation done

Success!: Transaction and triggers registered successfully!

Give an instruction to the robot!

Figure 4.4: Chatbot interaction example, second instruction.

Now, the Loomo on the right drives to the kitchen, since he is closer to the location than the left one.

4.3 Integration Process

With the approach of using LLMs to generate instructions for robots, the three major problems found in section 3.1 are addressed, with the security concern of directly executing LLM-generated responses being addressed by the user-response pattern over the chat.

In this section, this workj's main contribution is presented, and the necessity of each process step is explained. LLMs have been shown to have great promise in understanding and responding to natural language, but the challenge of integrating them into structured robotics control systems remains. Current approaches are focused simply on translation and evaluate its correctness based on the LLM's output as it is, as discussed in chapter 2. However, a structured method for incorporating LLMs into a robotics control system is lacking.

Two main research points are addressed in this thesis:

- Defining a structured setup for integrating LLMs into a transactional robotics control system. There is no established approach that ensures deterministic, reliable, and logically sound translation of user instructions into executable transactions. This work aims to fill this gap by presenting a process that enables LLMs to function effectively within such a system in a generalized and reproducible way.

- Given this work's approach with using the transaction system for execution safety and security, another layer of safety is added by making the user choose between transactions generated by the LLM instead of the direct execution of the LLM output. The integrated LLMs will be evaluated by scaling the number of generated responses, evaluating their impact on quality, and reasoning about the effectiveness of this approach.

The effectiveness of these two research points is the research questions, which are evaluated through the experiments. It demonstrates both the necessity of a well-defined generalized integration process of LLMs for a robotics control system and the effectiveness of the proposed multi-response pattern in improving the reliability of the usage of LLMs in such systems.

The experiments evaluate the proposed process steps of creating a dataset and evaluation metric in the setup, followed by system prompt development, fine-tuning, few-shot prompting, and a multi-response research.

Process Steps

Here, the process steps for answering the main research questions are shown. The goal is to provide a generalized, structured guideline and evaluate it, finding out how to optimize LLMs in general for the translation and integration task and to show the impact of scaling to multiple choices.

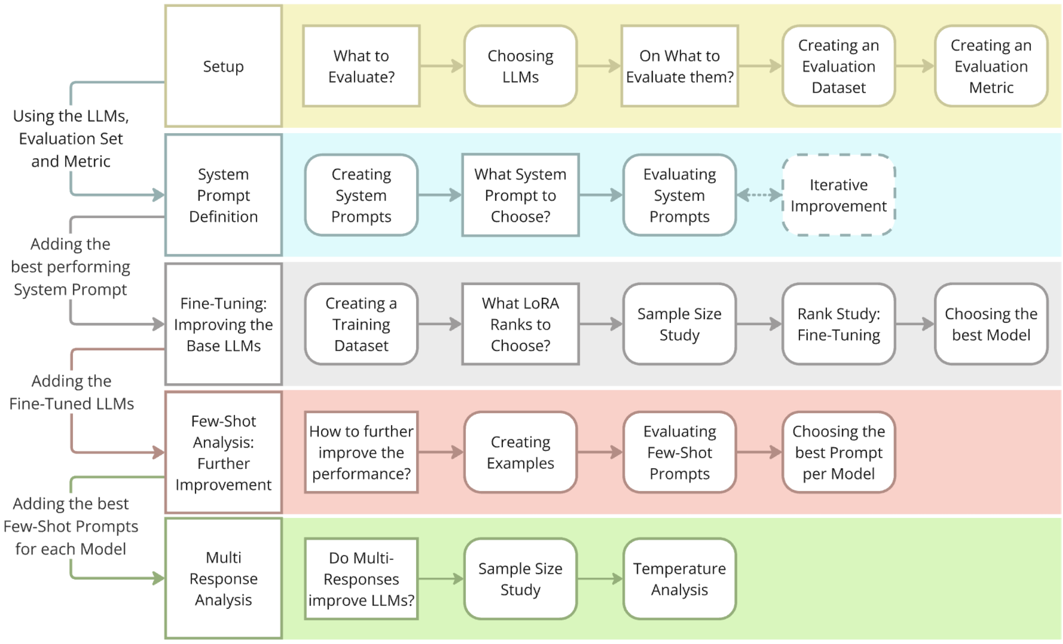

Setup
What to Evaluate?
Choosing LLMs
On What to Evaluate them?
Creating an Evaluation Dataset
Creating an Evaluation Metric
Using the LLMs, Evaluation Set and Metric
System Prompt Definition
Creating System Prompts
What System Prompt to Choose?
Evaluating System Prompts
Iterative Improvement
Adding the best performing System Prompt
Fine-Tuning: Improving the Base LLMs
Creating a Training Dataset
What LoRA Ranks to Choose?
Sample Size Study
Rank Study: Fine-Tuning
Choosing the best Model
Adding the Fine-Tuned LLMs
Few-Shot Analysis: Further Improvement
How to further improve the performance?
Creating Examples
Evaluating Few-Shot Prompts
Choosing the best Prompt per Model
Adding the best Few-Shot Prompts for each Model
Multi Response Analysis
Do Multi-Responses improve LLMs?
Sample Size Study
Temperature Analysis

Figure 4.5: Approach Overview.

In fig. 4.5, a visual representation of the approach that will be followed in the experiments is provided.

In the first step, it presents the setup. Here, the first question is what to evaluate—meaning what LLMs to choose, followed by how to evaluate them. This section requires an evaluation dataset and an evaluation metric. The dataset needs to represent the task that LLMs are tested to solve on, and the metric must be able to give a score on how well the LLM performs on the dataset.

The next step begins with the evaluation dataset, metric, and selected LLMs. Example system prompts are developed here, providing the LLM with the necessary context and guidance to generate a response that solves the wanted task. After creating the example system prompts, the question is which one to use or if one is already good enough. To answer that, they need to be evaluated on the evaluation dataset using the metric.

Upon evaluation, an iterative process starts, improving the system prompts and evaluating them again until the results are satisfactory. To keep the generalizable approach, it is especially important to find a stable prompt for each LLM to generate parsable instructions.

Adding the best system prompt to the setup, the guide follows up with the fine-tuning step to optimize the LLMs for the task. Since fine-tuning the LLMs needs to be done with a system prompt to associate the data with the task, this section follows after choosing a system prompt. Starting with the section, training data needs to be created that resembles a broad range of possible user inputs and the corresponding wanted output labels. Since full-model fine-tuning is very expensive, Low Rank Adaptation (LoRA) is used [18] to add weights to the LLMs, and only these are fine-tuned on the training data. This leads to the question of what rank size, a parameter determining the number of weights to fine-tune in LoRA, to choose. Rank sizes are largely dependent on the task and training data, so a rank study is proposed to find the optimal size. Since fine-tuning is a process that introduces noise, a sample study is conducted to find the optimal number of samples required per rank size and fine-tune all LLMs with the found sample size. The proposal is that the run-to-run variance for fine-tuning LLMs with LoRA is so small that one LLM with one rank size is suitable to find a sample size for each LLM on each rank size. Continuing with the fine-tuning for all LLMs in their different rank configurations, it is tested that the sample size stays valid given the statistical assumptions; otherwise, another sample study is needed. The best models for the optimal rank size are chosen to be the fine-tuned LLMs for the following sections.

Since having fine-tuning training data is a time-consuming process, this work keeps the base LLMs and the fine-tuned LLMs separate to show that the continued integration improvements are also feasible for the base LLMs.

In the next step, given the base and fine-tuned LLMs, each LLM's evaluation score on the dataset is further improved by updating the system prompt with examples. This process is called few-shot prompting, where examples added to the prompt are used to guide the LLM to generate better responses without changing the model weights. Since each LLM performs differently, there is no general solution for all LLMs, meaning that this section has to be done after having the base and fine-tuned LLMs ready. Examples are first created for categories in the evaluation dataset that most LLMs struggled with. With the differences in performance per LLM, an algorithm is proposed to find good example sets for each one by first evaluating each example on their own on the LLM and putting them into a descending sorted list, ranked by their performance. A broader search space of combinations is then explored using that sorted list, e.g., by taking the first, the third, and the fifth-best examples, and these combinations are evaluated. The search space includes 24 different variations. The best combination is then chosen to be the few-shot prompt for that LLM.

Now that the LLMs have undergone all these improvements, the second research question can be addressed. How well the LLMs scale with multiple responses for the task, especially when used in the proposed chatbot approach, is answered. For that, a temperature study is conducted. The randomness of LLMs is controlled by the temperature, and how it affects the quality of the generated responses is examined by two metrics: how does it improve the evaluation score on the evaluation dataset when multiple responses are generated and only the best one is chosen, and how does it score when averaging the results? Meaning that, if the user sees multiple responses, do they contain a good response, and what is the quality on average? Since temperature adds randomness to the evaluation, another sample study is suggested. This time, again one LLM is chosen, with the highest temperature that will be tested. It is argued that the variance between the LLMs' scores on those metrics is similar enough and that this variance decreases as the temperature is lowered. Following this, a temperature analysis is conducted, the validity of the sample size is ensured based on the statistical assumptions, and the research is concluded.

The integration process is evaluated in chapter 5 by following the steps to answer the research questions.

5 Experiments

In this chapter, the experiments are presented to analyze the performance of large language models (LLMs) in translating user intent to goals for the robotics control system following the integration process. The LLM's task is to generate a specified format that will be parsed in the robotics control system to execute the given instructions.

The first process is the setup, as seen in section 4.3, to define which LLMs to integrate and create an evaluation dataset and metric.

The experiments start by creating and optimizing a system prompt, included at the top of each prompt. A fine-tuning analysis of the LLMs is followed up on using the dataset and chosen prompt, and the system prompt is updated with a few-shot examples for the base models and fine-tuned ones. The experiments are concluded by providing a temperature analysis for the LLMs, analyzing the trade-off between determinism and creativity for controlling the robotics control system.

The created dataset represents popular benchmark tasks and specific tasks for controlling the robotics system. The goal of the experiments is to provide a structured approach in optimizing the performance of arbitrary LLMs for the robotics control system and to analyze and improve their performance.

For the experiments, the goals, expectations, structure, and execution containing the results and discussion, and a conclusion for each part of the experiments are presented.

5.1 Setup

A maximum evaluation time of two weeks is aimed for all setup steps over the chosen LLMs, and a 24GB VRAM GPU with BFloat16 support is used. For inferring the LLMs, the transformers library [49] is used to download and utilize models wrapped in a custom library to provide a unified interface for all LLMs, as well as to easily configure inference parameters such as the temperature, system prompt, and token generation.

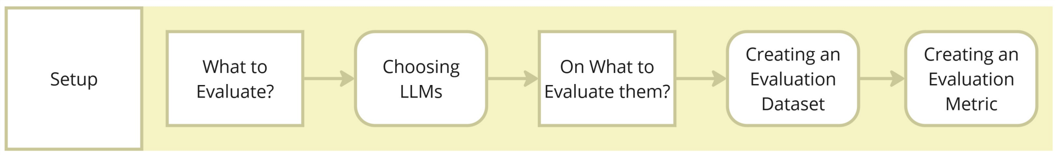

Setup
What to Evaluate?
Choosing LLMs
On What to Evaluate them?
Creating an Evaluation Dataset
Creating an Evaluation Metric

Figure 5.1: Step one of the Experiment Approach from fig. 4.5.

As in fig. 5.1, the first question by the proposed approach is what LLMs to evaluate.

5.1.1 Large Language Models

This study evaluates Microsoft's latest model, Phi3; Meta's LLama3.1; Google's Gemma2; and Mistral's Nemo on the task of translating user intent to goals for the robotics control system. The 8B parameter version of LLaMA 3.1 is used. For Phi3, the 14B model is used; for Mistral Nemo, the 12B model; and for Gemma2, the 27B model. To fit the ones over 10B parameters into the GPU VRAM, 4-bit quantization is used. These choices are made based on their performance in the Hugging Face Open LLM Leaderboards and their popularity [12]. A clear structure for the experiments with our diverse set of LLMs from different vendors is provided. The experiments can be easily replicated and extended to other LLMs.

LLama3.1 is currently one of the best-performing LLMs for code generation [51] and is also at the top of LLM benchmarks with its 70B parameter version. Phi3 fits into the same space and presented a 1.3B version of its model, which still holds up to the mentioned LLM's [30]. Mistral's new Nemo model improves upon the older versions and compares itself to the LLama model favorably, [3]. Gemma2 is Google's new state-of-the-art model. They claim its smaller models outperform Mistral, LLaMA, and Phi3, and the 27B model comes close to LLaMA's 70B model and GPT-4 [42]. For said models, their instruct version is chosen. Instruct models are fine-tuned to follow instructions and answer questions, typically used for chatting [33]. The instruction templates are parsed in the custom library from this thesis.

These models will be evaluated after fine-tuning and in their non-fine-tuned state to see their in-context performance.

5.1.2 Evaluation Dataset

Following with on what to evaluate the chosen LLMs from fig. 5.1, the goal is to create a set of evaluation data suitable to evaluate the performance in translating user intent into robotics control goals.

This work inspects the benchmark datasets from the open LLM Huggingface leaderboard, the most popular platform for benchmarking open source models [12] and evaluate the tasks included in them to find applicable ones for the robotics control system. It includes IFEval [54], an instruction-following evaluation dataset, BBH [41] benchmarking mathematics, reasoning, and question answering, MATH LvL 5 [17], containing textbook math problems, GPQA [35], multiple choice tasks about biology, physics, and chemistry, MMLU-PRO [47], similar questions about more fields including psychology, law, and more, and MUSR [38], solving textbook questions with multiple steps of reasoning required.

For this work, only the IFEval and BBH are applicable, since the rest is about textbook questions and answers, which are not suitable for the here-presented use case. The first two contain direct reasoning and instruction tasks, the main task for the robotics control system.

From IFEval, the following task categories for the evaluation dataset are chosen:

- Include Keywords, where the output has to include a specific keyword.
- Keyword Frequency, where the output has to include a specific keyword multiple times.
- Two Responses, where the LLM needs to respond twice.
- JSON Format. This category is implicitly included in the dataset since JSON is the expected format.

Other categories are about the number of words and sentences, language, Markdown formats, and similar text-based manipulation tasks that would alter the wanted JSON format.

From BBH, the following task categories are chosen:

- Geometric Shapes, reasoning about geometric shapes.

- Logical Deduction, reasoning about the environment.
- Navigation, reasoning about the position.
- Temporal Sequencing, reasoning about the order of actions.
- Word Sorting, sorting words alphabetically.

These task categories are used for this work's own benchmark, keeping their logic but adapting them to the robotics control system and context. Multiple task categories are added to cover multiple aspects of the system and to evaluate the reasoning capabilities of the LLMs.

The whole vocabulary of the system needs to be covered, based on the proof of concept with the Loomo Segway, to test if the LLM understands the transaction system logic, if it can infer and recognize patterns, if it can infer instructions and events, if it can logically deduce information from context provided by the user, if it can recognize text-based patterns, if it can handle abstract requests, and if it can handle multilingual tasks.

For that, top-level categories are created to group specific behavior, each containing multiple subcategories:

- Basic Evaluation
  - Basic evaluation prompts covering the whole vocabulary of the system.
  - Values for Event Topics, extending the tasks before with a focus on events.
- Multilingual Test
  - German Basic Evaluation, where the LLM gets prompted in German.
- Transaction System Logic
  - Event Sequencing, where the LLM has to reason about the order of actions. E.g., changing the order of reaching wanted locations in comparison to the text input, like going to place B after A, expecting to go to A first.
  - Multiple Events to react to, where the task contains multiple events to react to.
  - Parallel Instructions, where the LLM has to create parallel instruction lists.
- Inferring and Pattern Recognition

  - Referring to and Repeating things already done, where the LLM needs the robot to repeat actions.
  - Inferring tasks to do after calling them out, where the robot needs to say something and then do it.
  - User Interaction Logic, where the LLM has to reason about the users in the environment.

- Inferring Instructions and Events
  - Inferring User Responses, where the LLM has to infer user responses.
  - Inferring Danger Events, where the LLM has to infer reacting to a dangerous state.
- Logical Deduction
  - Environment Deduction. In this context, the deduction capabilities of the LLM are tested by providing it with information about the environment. For example, directing it to identify the coldest place.
  - Navigation Deduction, where the LLM has to reason about its position. E.g., if it ends up in a position after multiple driving instructions, it has to act based on that state.
- Text Based Patterns
  - Include Keywords. The LLM has to append keywords before locations or text output, e.g., appending the keyword 'LIVING-PLACE' when instructed to navigate to a position.
  - Keyword Frequency. The LLM has to repeat instructions multiple times.
  - Two Responses. The LLM has to provide two responses to the wanted request, making it a parallel instruction to, e.g., control two robots at the same time.
  - Word Sorting, driving to locations sorted alphabetically or making multiple movements in a sorted order.
- Abstraction of Movement

  - Geometrics. In this context, the LLM is instructed to create geometric movements for the robot, such as driving in a triangle.
  - Head Movements, where the LLM has to abstract the movement of the robot's head, e.g., answering by nodding.

For the basic evaluation, a set of 25 labels are created and for the rest of the tasks 6, resulting in a total of 133 handwritten prompts for evaluating the LLM's performance.

5.1.3 Evaluation Metric

For the basic evaluation, a set of 25 labels is created, and for the rest of the tasks, 6, resulting in a total of 133 handwritten prompts for evaluating the LLM's performance.

The goal is to satisfy the following criteria:

- Having a normalized score of 0 to 1, 0 meaning no similarity and 1 meaning full logical similarity.
- Independence of order between the lists, since instruction lists are executed in parallel.
- Loose Comparisons. In this context, instructions are given to the robot that convey meaning rather than exact values, such as directing it to drive a tick forward.
- Independence of where bad instructions lie, e.g., if the robot is instructed to say hello and drive forwards, but the LLM-generated say hello, say hello again, and drive forwards, a result of 1 - 1/3 is expected, since 2/3rds of the logic are correct to the label.
- Punish generating more than needed instructions or parallel instruction lists.

Given a set of instructions and events:

$$\begin{aligned}\text{Let } \mathcal{I} = \{&\texttt{StartingEmergencyStop},\\ &\texttt{LoomoSetsHeadRotationPitchYaw}(pitch, yaw),\\ &\texttt{LoomoDrivesTo}(position),\\ &\texttt{LoomoRotates}(rotation),\\ &\texttt{LoomoSpeaks}(text),\\ &\texttt{LoomoWaitsForUserResponse}(yes|no)\}.\end{aligned}$$

$$\begin{aligned}\text{Let } \mathcal{E} = \{&\texttt{DangerousStateDetected},\\ &\texttt{PersonDetected}(pitch, yaw),\\ &\texttt{LoomoBadUserResponseReceived}(position),\\ &\texttt{Topic}(text)\}.\end{aligned}$$

Each instruction and event has a type(a) and optionally associated parameters (specified in the brackets next to the type).

Instruction Similarity:

The instruction similarity is based on how closely two instructions of the same type match each other. For example, for geometric instructions, the instructions are similar if their target uses the same sign and is within a similar distance. The similarity is defined as being within 50 degrees for angles and 0.5 meters for positions. These values are chosen by observing the outputs of the used LLMs when tasking them with indirect wordings, e.g., drive a bit forward or rotate a bit to the right, as well as the reasoning that the robot's navigation stops being precise at under half a meter or smaller angles.

Comparing two instructions:

$$S(a,b) = \begin{cases} 0 & \text{if type}(a) \neq \text{type}(b), \\ S_{\text{matchedType}}(a,b) & \text{if type}(a) = \text{type}(b). \end{cases}$$

$$S_{\text{matchedType}}(a,b) = \begin{cases} 1 & \text{if } \tau = \texttt{StartingEmergencyStop}, \\[2ex] 1 & \text{if } \tau = \texttt{LoomoSetsHeadRotationPitchYaw} \\ & \quad \wedge\ |\text{pitch}(a) - \text{pitch}(b)| \leq 50^\circ \\ & \quad \wedge\ |\text{yaw}(a) - \text{yaw}(b)| \leq 50^\circ \\ & \quad \wedge\ (\text{sign}(\text{pitch}(a)) = \text{sign}(\text{pitch}(b))) \\ & \quad \wedge\ (\text{sign}(\text{yaw}(a)) = \text{sign}(\text{yaw}(b))), \\[2ex] 1 & \text{if } \tau = \texttt{LoomoDrivesTo} \\ & \quad \wedge\ \|pos(a) - pos(b)\| \leq 0.5 \text{ m}, \\[2ex] 1 & \text{if } \tau = \texttt{LoomoRotates} \\ & \quad \wedge\ |\text{rotation}(a) - \text{rotation}(b)| \leq 50^\circ \\ & \quad \wedge\ (\text{sign}(\text{rotation}(a)) = \text{sign}(\text{rotation}(b))), \\[2ex] S_{\text{DL}}(w(a), w(b)) & \text{if } \tau = \texttt{LoomoSpeaks}, \\[2ex] 1 & \text{if } \tau = \texttt{LoomoWaitsForUserResponse} \\ & \quad \wedge\ (\text{label}(a) = \text{label}(b)), \\[2ex] 0 & \text{otherwise.} \end{cases}$$

For τ = `LoomoSpeaks`, let $w(a), w(b)$ be the text parameters and d_{DL} the Damerau–Levenshtein distance to measure the similarity of words:

$$S_{\text{speaks}}(a,b) = 1 - \frac{d_{\text{DL}}(w(a), w(b))}{\max(|w(a)|, |w(b)|)}.$$

Instruction List Similarity:

Since the output of the LLM contains multiple instruction lists that are either direct or event-based, first, how to compare two arbitrary ones is defined.

Consider two instruction lists A and B:

$$A = \{a_1, \ldots, a_m\}, \quad B = \{b_1, \ldots, b_n\},$$

where $m = |A|$ and $n = |B|$.
Each $a_i \in \mathcal{I}$ and each $b_j \in \mathcal{I}$.

Each list can be either `direct` or `event-based`(e) with an event type $e \in \mathcal{E}$. Defining a type-compatibility function $C(A, B)$:

$$C(A,B) = \begin{cases} 1 & \text{if listType}(A) = \texttt{direct} \wedge \text{listType}(B) = \texttt{direct}, \\ 1 & \text{if listType}(A) = \texttt{event-based}(e_A) \wedge \text{listType}(B) = \texttt{event-based}(e_B) \wedge e_A = e_B, \\ 0 & \text{otherwise.} \end{cases}$$

Given this, the similarity between two instruction lists A and B is:

$$S_{A,B} = C(A,B) \cdot S'_{A,B}.$$

To define $S'_{A,B}$, first consider a set Σ of mappings from the indices of A to the indices of B:

$$\sigma : \{1, \ldots, m\} \to \{1, \ldots, n\} \cup \{\text{none}\}.$$

These mappings must satisfy:

1. (Ascending)

$$\forall i < j : \ \sigma(i) \neq \text{none}, \ \sigma(j) \neq \text{none} \implies \sigma(i) < \sigma(j).$$

2. (Injective)

$$\forall i \neq j : \ \sigma(i) \neq \text{none}, \ \sigma(j) \neq \text{none} \implies \sigma(i) \neq \sigma(j).$$

3. (Optional Unmatched) Elements of $\{1, \ldots, m\}$ that cannot be matched injectively and in ascending order are assigned none:

$$\exists i : \sigma(i) = \text{none allowed}.$$

Define:

$$S'_{A,B} = \max_{\sigma \in \Sigma} \left(\frac{\sum_{i=1}^{m} S(a_i, b_{\sigma(i)})}{m} \cdot \min\left(1, \frac{m}{n}\right) \right),$$

where $S(a_i, b_{\sigma(i)})$ is the similarity between individual instructions a_i and $b_{\sigma(i)}$.

Set-of-lists similarity:

Multiple instruction lists are expected, since these can run in parallel. So, considering two lists of instruction lists to compare a label and generated lists:

$$\mathcal{A} = \{A_1, \ldots, A_p\}, \quad \mathcal{B} = \{B_1, \ldots, B_q\},$$

where $p = |\mathcal{A}|$ and $q = |\mathcal{B}|$.

Defining a set $\mathcal{T}$ of mappings:

$$\tau : \{1, \ldots, p\} \to \{1, \ldots, q\} \cup \{\text{none}\}$$

such that:

1. (Injective)

$$\forall k_1 \neq k_2 : \; \tau(k_1) \neq \text{none}, \; \tau(k_2) \neq \text{none} \implies \tau(k_1) \neq \tau(k_2).$$

2. (Optional Unmatched)

$$\exists k : \tau(k) = \text{none allowed.}$$

Define:

$$S_{\mathcal{A},\mathcal{B}} = \max_{\tau \in T} \left(\frac{\sum_{k=1}^{p} S_{A_k, B_{\tau(k)}}}{p} \cdot \min\left(1, \frac{p}{q}\right) \right).$$

Here, the order does not matter, since the instruction lists are executed in parallel.

This final function is used to compare the lists of instruction lists, with finding the optimal projection from the generated list to the label list.

Evaluation Examples

To illustrate the various evaluation metric scenarios, examples are presented where the label corresponds to the desired label and the generated output serves as a benchmark for comparison. For readability, abstract code is used.

The metric is associative for instruction lists, since they are executed in parallel:

```
label:
        - [LoomoRotates(90)]
        - [LoomoSpeaks("Hello")]
reversed:
        - [LoomoSpeaks("Hello")]
        - [Instruction::LoomoRotates(90)]
similarity == 1.0
```

The metric is order-sensitive for instructions when comparing lists, since the order of instructions matters:

```
label:
        - [LoomoRotates(90), LoomoSpeaks("Hello")]
reversed:
        - [LoomoSpeaks("Hello"), LoomoRotates(90)]
similarity == 0.0
```

Adding one bad instruction results in a decrease in the similarity:

```
label:
        - [LoomoRotates(90), LoomoSpeaks("Hello")]
generated:
        - [LoomoSpeaks("beep"), LoomoRotates(90),
                LoomoSpeaks("Hello")]
similarity == 1.0 - (1.0 / 3.0)
```

The same applies to missing an instruction. Here, the similarity is 0.5, meaning that only 50% of the logic is correct:

```
label:
        - [LoomoRotates(90), LoomoSpeaks("Hello")]
```

```
generated:
    - [LoomoSpeaks("Hello")]
similarity == 0.50
```

The metric allows for subtle differences in matched instructions, since indirect wordings are expected for some instructions, e.g., drive a bit forwards:

```
label:
    - [LoomoRotates(90),
          LoomoDrivesTo(LocalCoordinateInCm(x=100, y=0)),
          LoomoDrivesTo(NamedLocation("Bedroom")),
          LoomoSpeaks("Hello")]
generated:
    - [LoomoRotates(70),
          LoomoDrivesTo(LocalCoordinateInCm(x=80, y=0)),
          LoomoDrivesTo(NamedLocation("bedroom")),
          LoomoSpeaks("hello")]
similarity == 1.0
```

If the instructions differ too much, the similarity is 0:

```
    label:
        - [LoomoRotates(90)]
    generated:
        - [LoomoRotates(-90)]
similarity == 0.0
```

5.2 Experiments

In this section, the following approach steps, as seen in fig. 4.5, are presented, to conclude the thesis.

5.2.1 System Prompt

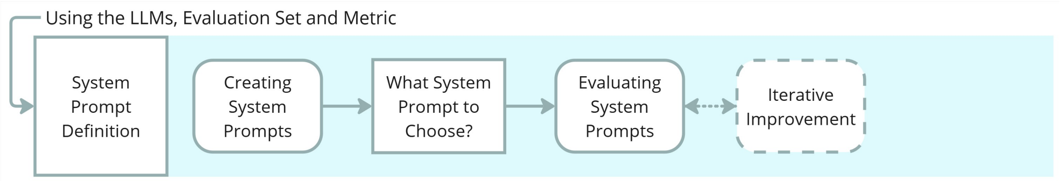

Using the LLMs, Evaluation Set and Metric
System Prompt Definition
Creating System Prompts
What System Prompt to Choose?
Evaluating System Prompts
Iterative Improvement

Figure 5.2: Step two of the Experiment Approach from fig. 4.5.

Following the next guideline step, as seen in fig. 5.2, the created evaluation dataset, the similarity score as the evaluation metric, and the picked LLMs Gemma2, Llama3.1, Mistral Nemo, and Phi3 are added for this experiment.

A system prompt is used to guide the output of the LLMs by being appended to each input [36]. A template structures the instruction prompts, filling in the placeholders with user input and system context. [36]. For the used instruction models, model-specific instruction templates are used per model that contain a specific header tag for user input and assistant output. If the instruction template does not contain a separate system tag, where the system prompt resides, it is appended to the user tag and separated with a Context:' and Request' block.

The goal for this section is to create a system prompt that first results in valid output for each LLM and secondly has a higher than 0.5 evaluation score for the basic evaluation set.

The expectation is that the first system prompts will have difficulties providing a valid output format for some models but can be fixed by improving them in an iterative process. It is also expected that the evaluation goal for the basic evaluation set will be reached, since it is a simple set of instructions. For the more complex tasks, a lower score is expected.

First, two baseline system prompts are defined, and each LLM is evaluated using those. Iteratively, the prompts are improved until a conclusion is reached based on the goal by analyzing the outputs of the LLMs.

OpenAI's prompting guide [32] provides the following strategies. They are presented here and shown how they are integrated into this work's prompts:

- Ask the model to adopt a persona: The model is asked to take the role of the robot.

```
You are a robot named eoomo and respond to requests from a user
in the form of an instruction system. Instructions can be based
on events. These instructions  form the goal for the robot. The
data layouts are in JSON format. Variables are highlighted with
a $ symbol before them, following their type.
```

- Provide reference text for its vocabulary: A list of words that the system prompt uses is provided.

```
The different types of events are:
- "DangerousStateDetected"
      - An event that indicates a dangerous state.
- "LoomoBadUserResponseReceived"
      - An event that indicates that the robot named Loomo
        received a wrong user response or a timeout waiting on
        a user response occurred.
- "PersonDetected"
      - An event that indicates that a person was detected.
- {"Topic": $STRING}
      - An event that indicates that an event occurred on the
        topic named by the value string.

The different types of instructions are:
- "StartingEmergencyStop"
      - An instruction to start an emergency stop.
- {"LoomoDrivesTo":{"LocalCoordinateInCm":{"x":$int,"y":$int}}}
      - An instruction for the robot named Loomo to drive to a local
        coordinate (x, y) in centimeters in front of the robot.
- {"LoomoDrivesTo":{"GlobalCoordinateInCm":{"x":$int,"y":$int}}}
      - An instruction for the robot named Loomo to drive to a global
        coordinate (x, y) in centimeters on the map.
- {"LoomoDrivesTo":{"NamedLocation": $STRING}}
      - An instruction for the robot named Loomo to drive to a
        globally named location, represented by the argument.
```

```
- {"LoomoRotates": $int}
      - An instruction for the robot named Loomo to rotate, in degrees.
        Positive means rotating to the left, negative to the right.
- {"LoomoSetsHeadRotationPitchYaw":{"pitch": $int,"yaw": $int}}
      - An instruction for the robot named Loomo to rotate his head to a
        specified position. (pitch,yaw) in degrees. Negative pitch means
        moving the head down, positive pitch means moving the head up.
        Negative yaw means moving the head to the right, positive yaw
        means moving the head to the left.
- {"LoomoSpeaks": $String}
      - An instruction for the robot named Loomo to say the text given as
        a value.
- {"LoomoWaitsForUserResponse": $"yes|no"}
      - An instruction for the robot named Loomo to wait on a user input
        and expect the given answer. Can either be "yes" or "no". If the
        Loomo awaits a "yes", but a "no" answer is given or a timeout
        occurs, then an event of the type LoomoBadUserResponseReceived
        will be automatically thrown.
```

- Write clear instructions of what the model needs to do: Clear instructions of how it needs to respond are provided.
- Specify the output: The output format is directly specified in the system prompt.

```
Your task is to return a response in the following JSON format,
returning a list containing direct and/or event-based instruction
lists. You have to translate the user's intention into instructions.

A direct entry looks like this:
{"Direct":{"instructions":[$Instruction...]}}
      - If the user wants something to happen that is not based
        on external events.

An event based entry looks like this:
{"EventBased":{"event":$EventType,"instructions":[$Instruction...]}}
      - If the user wants something to happen based on an external event.
            The event argument can only be of the types given in the
            events list. The second value is an instruction list.

You need to respond ONLY (no extra text, and as a one liner) with a list
containing either of them:
[$Direct|$EventBased ...]
Meaning that if you only need to return one Direct Instructionlist, you
```

```
still return it in a list.
ONLY use the types provided and ONLY Change variables with a Dollar sign
before them.
```

- Test changes systematically: The system prompt is iteratively improved.
- Use delimiters: Instruct models are used, and delimiter tokens are directly added into the text.
- Providing examples: A short example of the output structure is provided. Few-shot examples are added later in section 5.2.3.

To have a second baseline prompt, ChatGPT is used to generate an improved system prompt out of the first one. The final version was slightly altered, and mistakes were fixed to align with the correct vocabulary and structure. It mainly separated each section by a markdown block, added markdown enumerations, and added headers for each section.

The sections are about the context, event types, instruction types, response requirements, and rules to follow.

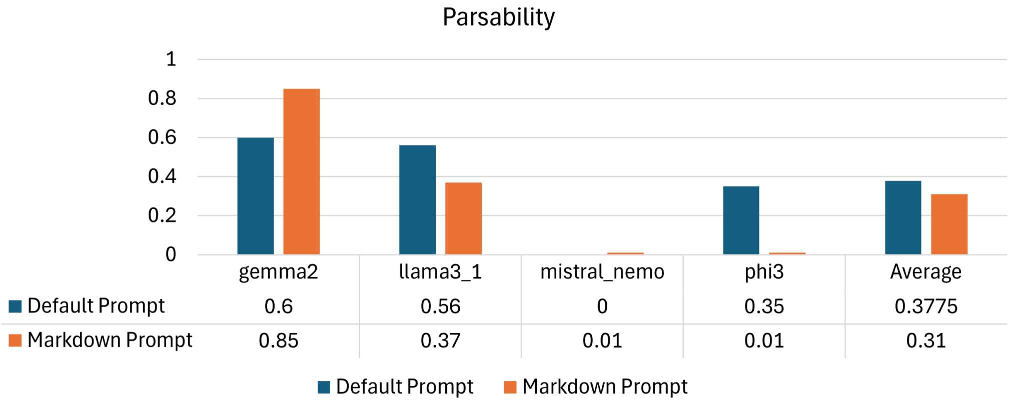
Results:

First, the percentage of parsable outputs is calculated for each model for the whole evaluation set. For these benchmarks, the temperature value for each LLM is set to 0, meaning that it behaves deterministically and always picks the most likely token.

As seen in fig. 5.3, both prompts failed to get the LLMs to output valid JSON. The default prompt did slightly better on average, since phi3 and llama3.1 had a higher percentage of valid outputs using it, while the markdown prompt achieved the highest score for gemma2 but was worse for the other models. Mistral Nemo failed to output correct JSON on either prompt.

With these results, an analysis is first conducted on the failed outputs for both prompts:

- Mistral Nemo, Llama 3.1, and Phi 3 failed to produce correctly formatted JSON, often with missing brackets or wrong characters at the beginning of the JSON.
- Gemma2 sometimes added markdown JSON symbols, more often in the default prompt.

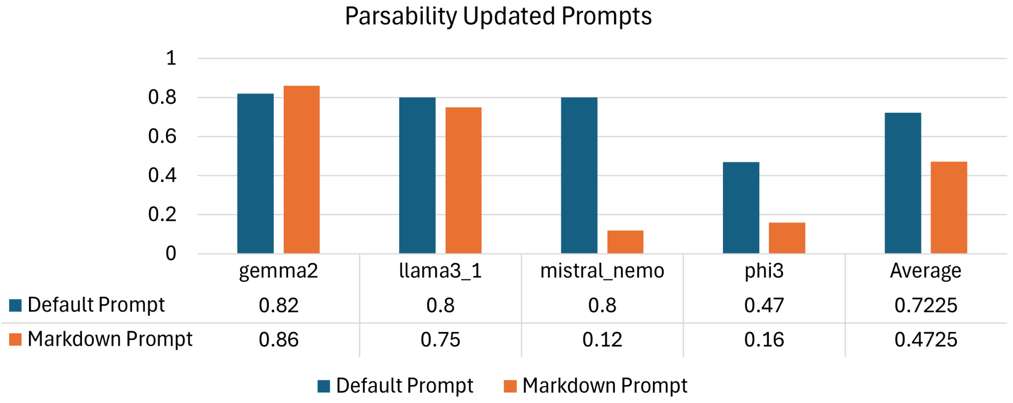


	gemma2	llama3_1	mistral_nemo	phi3	Average
■ Default Prompt	0.6	0.56	0	0.35	0.3775
■ Markdown Prompt	0.85	0.37	0.01	0.01	0.31

Figure 5.3: Parsability of the LLM's outputs for the evaluation set with the first system prompts.

To fix these issues, an iterative improvement is made, as seen in the approach in fig. 5.2, updating both prompts by adding the JSON schema to the prompt. A JSON schema represents a structural contract for the JSON output; following it ensures that the output is correctly formatted. Also, the following text after the added JSON schema is appended:

````
ONLY give a response in the mentioned JSON format, no additional text and only
exactly one response. Do NOT add ```json and ``` around the response.
````



	gemma2	llama3_1	mistral_nemo	phi3	Average
■ Default Prompt	0.82	0.8	0.8	0.47	0.7225
■ Markdown Prompt	0.86	0.75	0.12	0.16	0.4725

Figure 5.4: Parsability of the LLMs outputs for the evaluation set with the updated prompts.

Figure 5.4 shows the updated prompt's parsability score. This update improved the default prompt by a lot, going from an average of 37.75% correct parsing of the LLM's outputs to 72.25%. The markdown prompt only improved by about 16%, but still had under 20% performance for Mistral Nemo and Phi3.

These were the issues with the updated prompts:

- On the updated default prompt, Phi3 hallucinated types into the JSON output that were not contained in the JSON schema and sometimes still had bad bracket placements. On the updated Markdown prompt, JSON markdown symbols were still often added.
- Mistral Nemo had only in the updated Markdown prompt the same issues as before, adding bad characters or wrong brackets.

Concluding the results, although the updated prompts improved the parsability of the LLM's outputs, the issues still appear to be too variable between the models and not suitable for the experiment structure. This is particularly true since a generally applicable guideline is desired for use on any LLM.

Guided LLM Generation:

With this conclusion, the next iteration of improvement from the approach in fig. 5.2 is started by adding a structured text generation tool. Structured text generation means that the LLM's output tokens are constrained to a specific format.

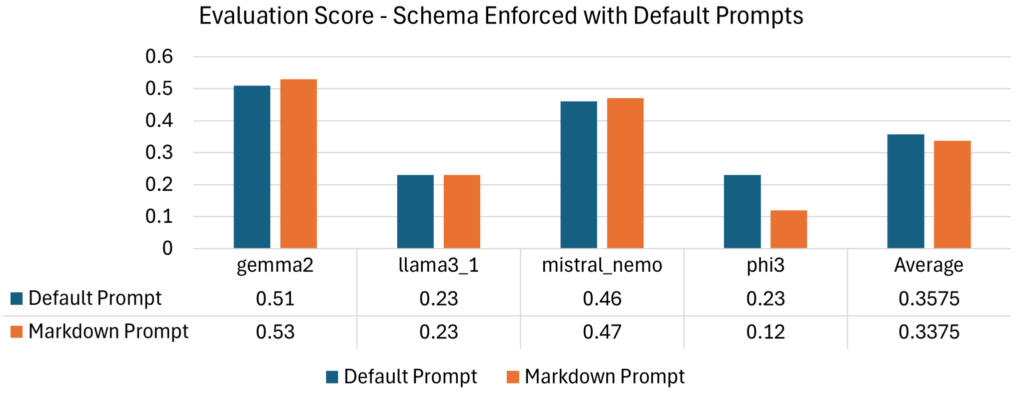
For LLMs, the next token is sampled from a categorical distribution over each token in the vocabulary. In guided generation, before the token is sampled, a mask is applied to only allow certain tokens to be sampled.

Most implementations create a mask after each new token by re-evaluating the whole vocabulary, which adds huge computational overhead for generating tokens [48].

Outlines the guided LLM generation tool used in this work [48], converting the schema into a Finite State Machine that is used to mask the vocabulary. This approach means that for each token, the mask is already included in the current state and can be applied directly.

With the JSON schema enforced by the tool, a parsability of 100% is achieved for each model, except when it generates too much output, which is restricted to a set maximum.

With that, the LLMs are evaluated on the evaluation dataset using the evaluation metric defined in section 5.1.3.



	gemma2	llama3_1	mistral_nemo	phi3	Average
■ Default Prompt	0.51	0.23	0.46	0.23	0.3575
■ Markdown Prompt	0.53	0.23	0.47	0.12	0.3375

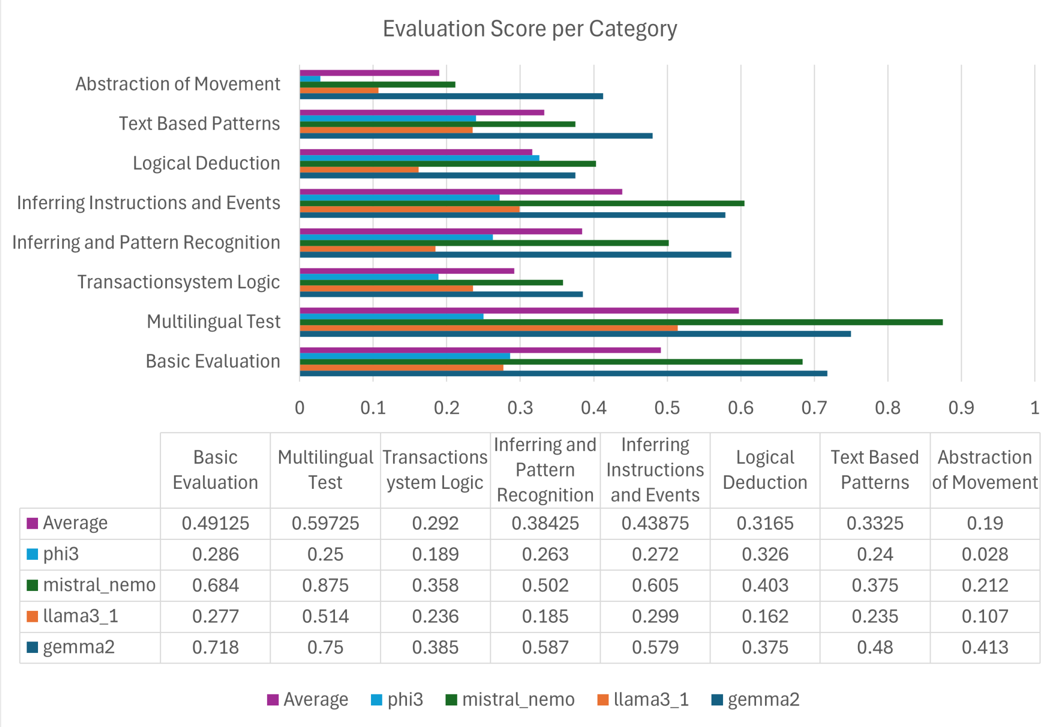
Figure 5.5: Evaluation Score of the LLMs outputs for the evaluation set with the first system prompts, using guided LLM generation to enforce the JSON schema.

Figure 5.6 shows the evaluation score for each LLM using the guided LLM generation, meaning that each output was valid JSON. Both prompts achieved very similar results, with the default prompt averaging slightly higher due to better performance with Phi3. For this work, the default prompt is chosen. Looking at the results, Gemma2 already achieved a slightly above 0.5 evaluation score for the evaluation set, with Mistral Nemo being close behind it with 0.46. Phi3 and LLama3.1 share the same score of around 0.23.

Looking at fig. 5.6, the evaluation score for each LLM per category is seen, using the default prompt with guided LLM generation. The 'Basic Evaluation' and 'Multilingual Test' categories have the highest scores, as the tasks are basic and simply test vocabulary. The other, more complex tasks have lower scores, with the 'Abstraction of Movement' only reaching around 0.19 on average.

Conclusion

Concluding this section, two system prompts were provided for the LLMs, following the strategies from OpenAI's prompting guide [32]. Looking at the expectations with a focus on iteratively improving the prompts, it is seen that for the use case of needing a specific JSON format, updating the prompts alone did not fully work for all LLMs. For this reason, it is concluded that guided LLM generation is needed to have a stable output



	Basic Evaluation	Multilingual Test	Transactionsystem Logic	Inferring and Pattern Recognition	Inferring Instructions and Events	Logical Deduction	Text Based Patterns	Abstraction of Movement
🟪 Average	0.49125	0.59725	0.292	0.38425	0.43875	0.3165	0.3325	0.19
🟦 phi3	0.286	0.25	0.189	0.263	0.272	0.326	0.24	0.028
🟩 mistral_nemo	0.684	0.875	0.358	0.502	0.605	0.403	0.375	0.212
🟧 llama3_1	0.277	0.514	0.236	0.185	0.299	0.162	0.235	0.107
■ gemma2	0.718	0.75	0.385	0.587	0.579	0.375	0.48	0.413

Figure 5.6: Evaluation Score of each LLM per Category using the Default Prompt with guided LLM generation.

and build a pipeline that works for arbitrary LLMs. The LLM choices have different performances, with Gemma2 already achieving around a 0.5 evaluation score for the evaluation set, while Mistral Nemo is close behind it with 0.46. Phi3 and LLama3.1 share the same score of around 0.23. The expectation of achieving more than a 0.5 evaluation score for the basic category was only met for Gemma2 and Mistral Nemo, with only Mistral Nemo achieving more than 0.5 in two of the other complex categories.

5.2.2 Finetuning

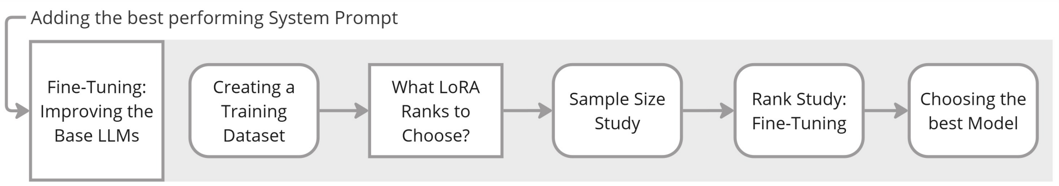

Adding the best performing System Prompt
Fine-Tuning: Improving the Base LLMs
Creating a Training Dataset
What LoRA Ranks to Choose?
Sample Size Study
Rank Study: Fine-Tuning
Choosing the best Model

Figure 5.7: Step three of the Experiment Approach from fig. 4.5.

Here, the experimental approach continues with the next step, as seen in fig. 5.7, by adding the system prompt with the guided LLM generation to the LLMs and fine-tuning them to improve their performance on the evaluation set.

An LLM is trained on a specific dataset during fine-tuning to enhance its performance for that set. Since full-model finetuning is very costly and time-consuming [18], LoRA, Low-Rank Adaptation, is used, as shown in fig. 5.8. LoRA adds extra weights to the LLM layers and freezes the rest of it so that it can be fine-tuned. This means that only the new weights are trained, and their sizes are determined by the rank size A'. A' is given a random Gaussian initialization, and B' is set to 0.r'. B' are the weights of these two layers, where The output of one layer with an added LoRA adapter is simply the addition of the old weights and the LoRA weights.

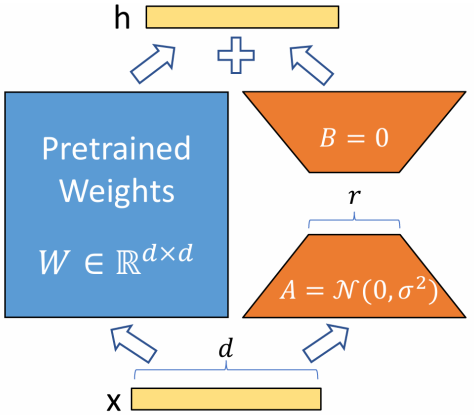


Figure 5.8: LoRA reparametrization, image from [18].

The goal here, next to improving the LLM's performance, is to measure how much of an impact the fine-tuning has, as well as give a systematic approach to choosing the size of the added weights.

It is expected that each LLM will be improved after being fine-tuned, especially Llama 3.1 and Phi 3, which shared the worst performance, and expect the size of the added weights to not matter much for fine-tuning, since the vocabulary is small.

First, the training data is presented, followed by a LoRA rank analysis to find out what rank size to choose as seen in fig. 5.7. The LLM's finetuning and evaluation losses while finetuning are shown and analyzed, followed by an evaluation of the finetuned LLMs.

Training Data Generation:

In this section, the training data used to fine-tune the LLMs is presented as a simple proof of concept to provide examples over the whole vocabulary. First, parts of a sentence indicating a specific instruction or event are created.

Examples for the instruction ‘LoomoSpeaks’:

```
Hey Loomo, say hello.
Greet me, Loomo!
```

For each flat instruction and event, a list of 100 examples is provided. Around 10–25 examples per instruction and event were written by us, and the rest were generated by ChatGPT 4.0, ChatGPT 3.5 Turbo, and Microsoft Copilot. A program is created that reads in the example sentences and concatenates them, creating random sentence combinations. The resulting concatenation is then cleaned up with random connection words and saved with the paired instruction and event values as a training data prompt.

A full example:

```
Input Text:
Loomo, go to the bedroom, then please navigate to the kitchen, and
    retrieve a yes response.
Upon identifying a dangerous scenario, say alert.

Label:
[{
    "Direct": {
      "instructions": [
        {
          "LoomoDrivesTo": {
            "NamedLocation": "bedroom"
          }
        },
        {
          "LoomoDrivesTo": {
            "NamedLocation": "kitchen"
          }
        },
        {
          "LoomoWaitsForUserResponse": "Yes"
        }
      ]
    }
  },
  {
    "EventBased": {
        "event": "DangerousStateDetected",
        "instructions": [
          {
            "LoomoSpeaks": "alert"
          }
        ]
      }
}]

Human readable text for LLM interaction:
Loomo reaches the bedroom -> Loomo reaches the kitchen -> Loomo
    awaits a yes from the user
If a dangerous event happened: Loomo says alert
```

The input text is an example of randomly chosen sentences. The LLM gets it as an input, and the label is the expected output. The human-readable text for LLM interaction is what the user sees after sending his request to the LLM and what he can choose to accept or not on the Chatbot front end.

A total of around 700 training prompts are created this way. For validating the generated dataset, to test if it has enough distinct samples and no duplicates, [4] is used, a data profiling tool. The size is rather on the smaller side according to [44], but it is argued that, with the small vocabulary, it is sufficient.

LoRA Rank Analysis:

For fine-tuning with LoRA, the rank size ‘r’ needs to be chosen, the main parameter to tune from the authors of LoRA [18], meaning the size of the added weight matrices to the LLM’s layers.

The fine-tuning library used is the Supervised Fine-tuning trainer from Hugging Face with the default settings from LoRA [19]. Only the rank size is varied for this experiment, and the models are trained for one full episode.

The goal here is to find out if one rank size performs best for all models or if each model needs a different optimal rank size.

Since fine-tuning depends on many random variables, such as the randomized initialization of the LoRA weights [18], the random sampling of the training data, and the randomness in the optimizer, a suitable sample size needs to be determined first.

For that, the following assumptions are made to save computational resources:

- The variance in performance of 50 fine-tuned Llama 3.1 with rank size 16 is smaller or within Levene's test for homogeneity of variances of the same or other models of varying ranks. The Levene test measures if multiple samples have the same variance [26].
- To determine the required sample size for comparing different LLMs and ranks, collecting 50 samples from Llama 3.1 with rank size 16 is sufficient. Using these samples, a sample size analysis is made to estimate the number of observations needed for detecting a performance difference of 1% (Cohen's d = 0.1 [9]) with 90% statistical power and a significance level of 0.05. These values are commonly used in statistical analysis [31]. This means that only 50 runs of fine-tuning and evaluating Llama 3.1 with rank size 16 are needed to find out the sample size. As long as the variance of each LLM and rank size variation is not significantly different from Llama 3.1 with rank size 16, according to Levene's test, the sample size can be used for comparing LLMs in their rank sizes.

Following the above two points, t-tests [40] can be conducted for each model with their different rank sizes with the same sample size to find significant differences in performance, given the presented statistical power and significance level.

Sample Size Analysis:

Here, 50 separate fine-tuning and evaluation runs are conducted for Llama 3.1 with rank size 16 to calculate an appropriate sample size for further fine-tuning experiments.

The results after 50 runs, shown in fig. 5.9, show an average evaluation score of the fine-tuned models of 0.52, with q1, the 25th percentile at 0.515, and q3, the 75th percentile at 0.525. There is only a small variance, with 50% of the results being within 1% of the performance difference.

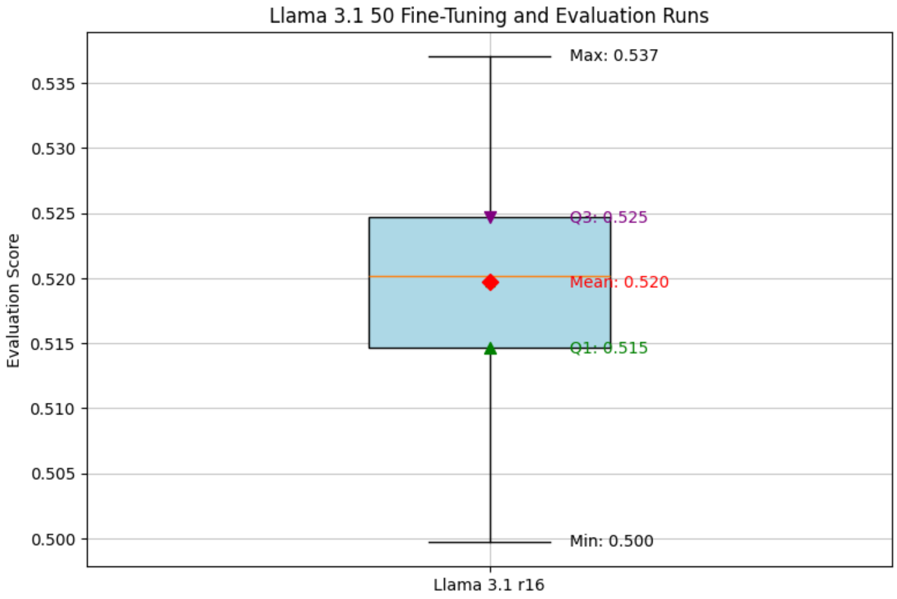


Figure 5.9: Box Plot for fine-tuning Llama 3.1 50 times and evaluating its performance with rank size 16.

Calculating the sample size needed for detecting a performance difference of 1% with 90% statistical power and a significance level of 0.05, a sample size of 7 is calculated, since the variance is very low for the fine-tuned models.

LoRA Rank Results:

The sample size of 7 is now used to find the optimal rank size for each LLM, meaning that each LLM is fine-tuned with rank sizes 8, 16, and 32 and evaluated on the evaluation set 7 times. Note that a higher rank size was not used, as it did not fit on the GPU's VRAM for the 27B Gemma2 model.

For each following run, the variances of the evaluation scores were tested using Levene's test, and each one passed, making the sample size of 7 sufficient for the given assumptions.

Figure 5.10 shows the results for fine-tuning gemma2 with different rank configurations. In the t-test, no significant difference in performance was observed between the rank sizes. The average performance was highest for rank size 8, with the highest single value

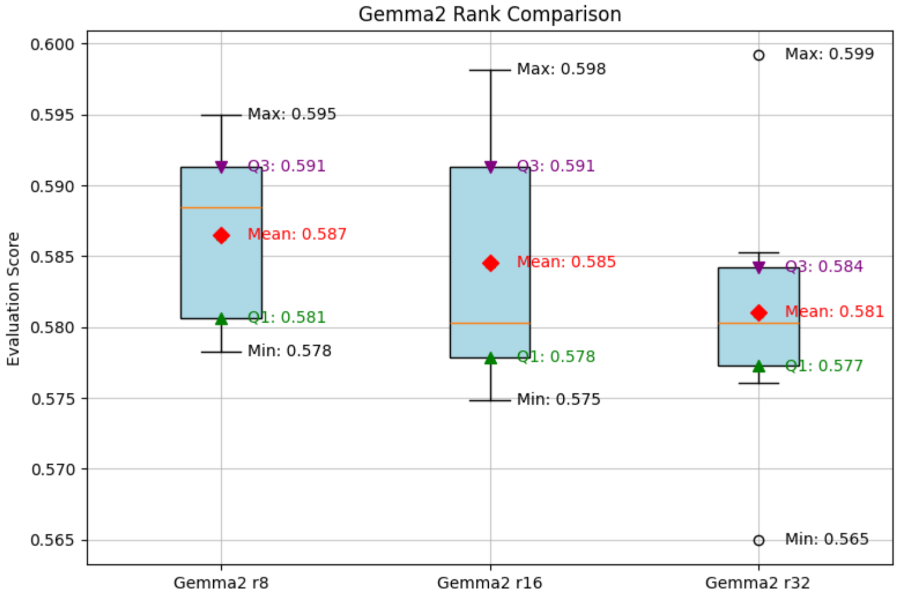


Figure 5.10: Box Plot for fine-tuning Gemma2 7 times and evaluating its performance with different rank sizes.

being an outlier for rank size 32. The conclusion for this model is that each of the chosen rank sizes is applicable.

For Llama 3.1, shown in fig. 5.11, the mean for rank size 16 is almost identical for the sample size of 7 compared to the sample size of 50 in fig. 5.9. Conducting the t-test, a significant difference was found for rank size 8 compared to 16 and 32, with 16 and 32 having the highest average and maximum scores. The conclusion for this model is that rank size 8 is the best choice.

For Mistral Nemo, shown in fig. 5.12, the t-test also showed a significant difference for rank size 8 compared to 16 and 32, with it also having the highest average and maximum score, also concluding that rank size 8 is the best choice.

The same goes for Phi3, shown in fig. 5.13, where the t-test also showed a significant difference for rank size 8 compared to 16 and 32, with rank size 8 performing the best.

The overall result is that the rank size 8 is the best choice for all tested models given this works dataset, showing a significant difference in performance for all models but Gemma2, with the highest average score for all models based on the statistical assumptions and samples.

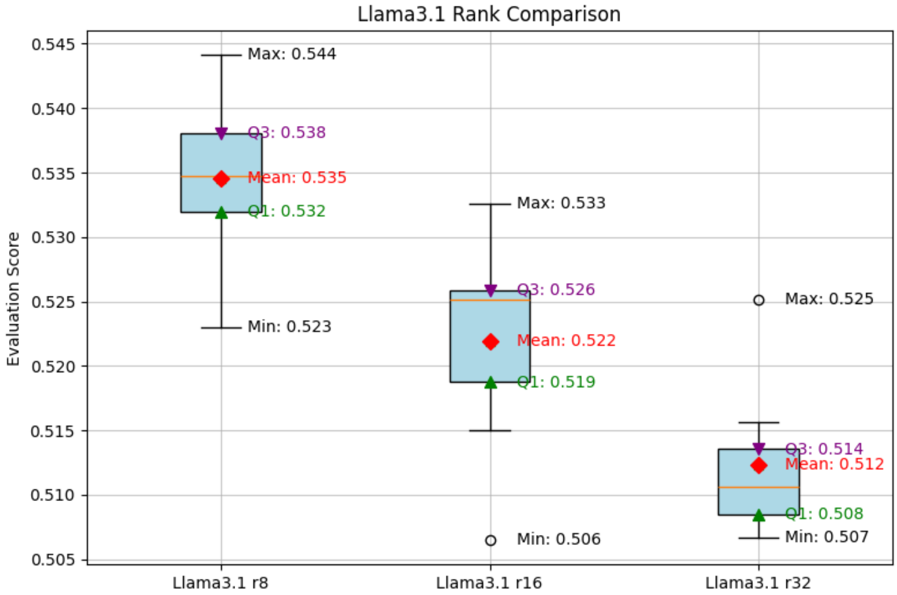


Figure 5.11: Box Plot for fine-tuning Llama3.1 7 times and evaluating its performance with different rank sizes.

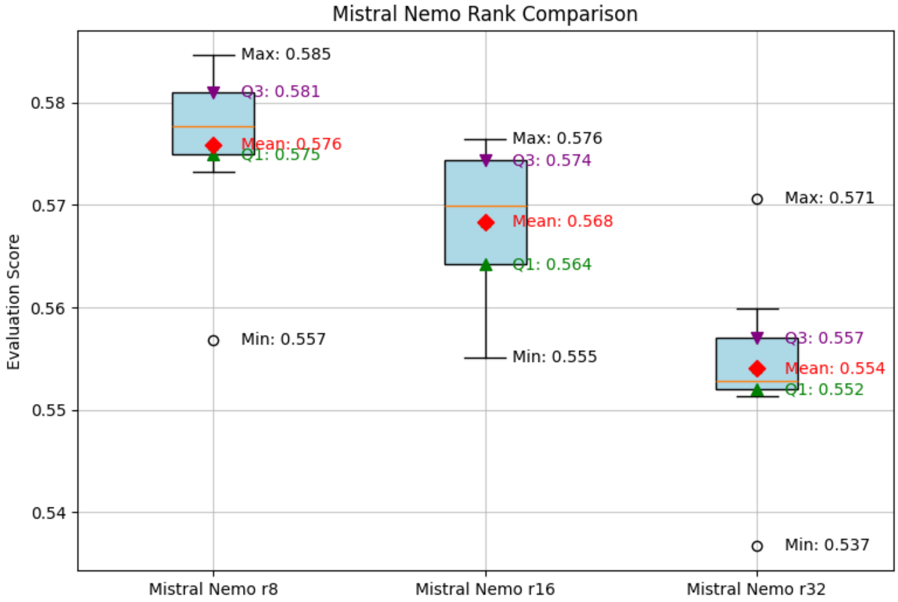


Figure 5.12: Box Plot for fine-tuning Mistral Nemo 7 times and evaluating its performance with different rank sizes.

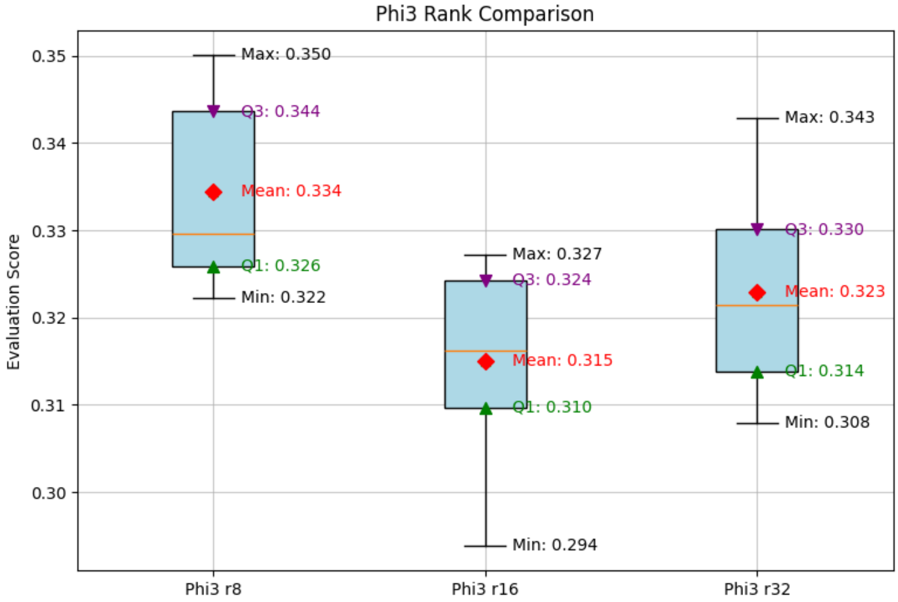


Figure 5.13: Box Plot for fine-tuning Phi3 7 times and evaluating its performance with different rank sizes.

Fine-Tuning Loss:

Here, the stability while training the LLMs with the chosen rank size of 8 is shown.

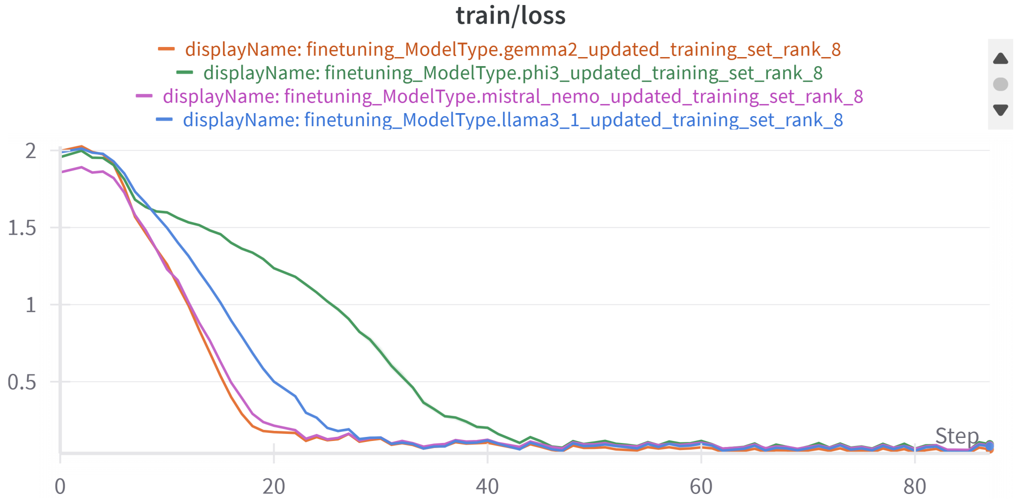
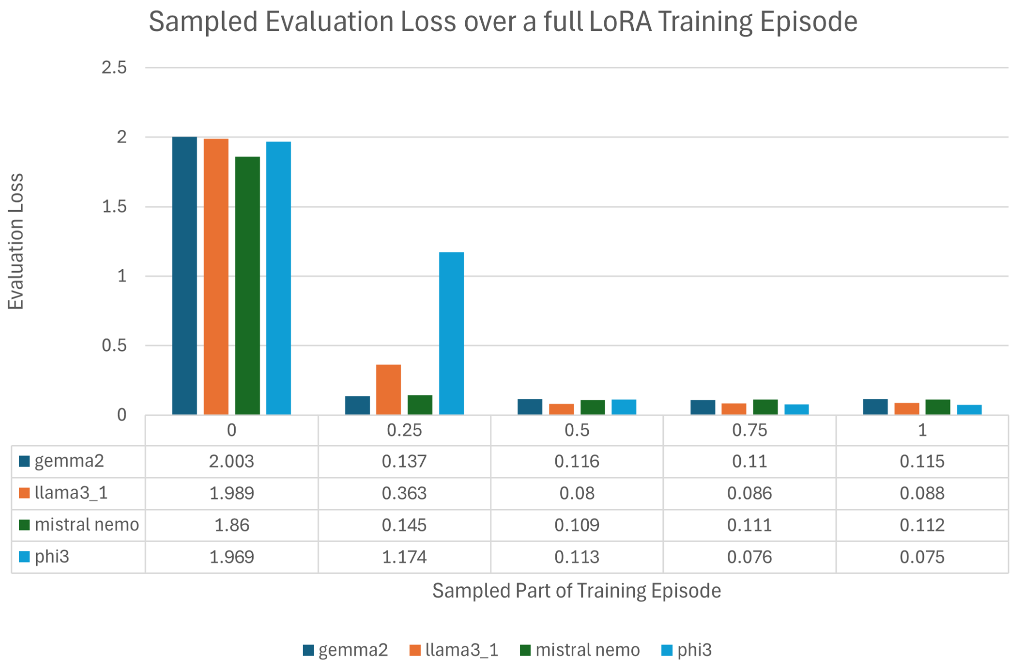


Figure 5.14: Training Loss for each LLM while finetuning on rank 8. Average of 7 runs.



	0	0.25	0.5	0.75	1
gemma2	2.003	0.137	0.116	0.11	0.115
llama3_1	1.989	0.363	0.08	0.086	0.088
mistral nemo	1.86	0.145	0.109	0.111	0.112
phi3	1.969	1.174	0.113	0.076	0.075

Figure 5.15: Evaluation Loss for each LLM while finetuning on rank 8. Average of 7 runs.

Figure 5.14 shows the training loss for each LLM while fine-tuning on rank 8. The average of 7 runs is shown per LLM. The y-axis shows the loss, and the x-axis the training step. fig. 5.15 shows the evaluation set loss sampled 5 times across the full training episode, with 0 representing the start of the training and 1 the end. Each LLM converges in their training and evaluation loss, with Gemma2 converging on a similar rate with Mistral Nemo, closely followed by Llama3.1. Phi3 takes the longest to converge.

These results show that each LLM converges in their training and evaluation loss, resulting in a stable fine-tuning process.

Fine-Tuned Models:

For the rest of this work, the highest-performing trained model for each LLM is selected based on the given rank size of 8.

Figure 5.16 shows the evaluation score for each LLM before and after fine-tuning. Gemma2 improved the least by 16.70%, but still has the highest score of 0.595, closely followed by Mistral Nemo with an improvement of 27.17% and a score of 0.585. Phi3 improved a lot

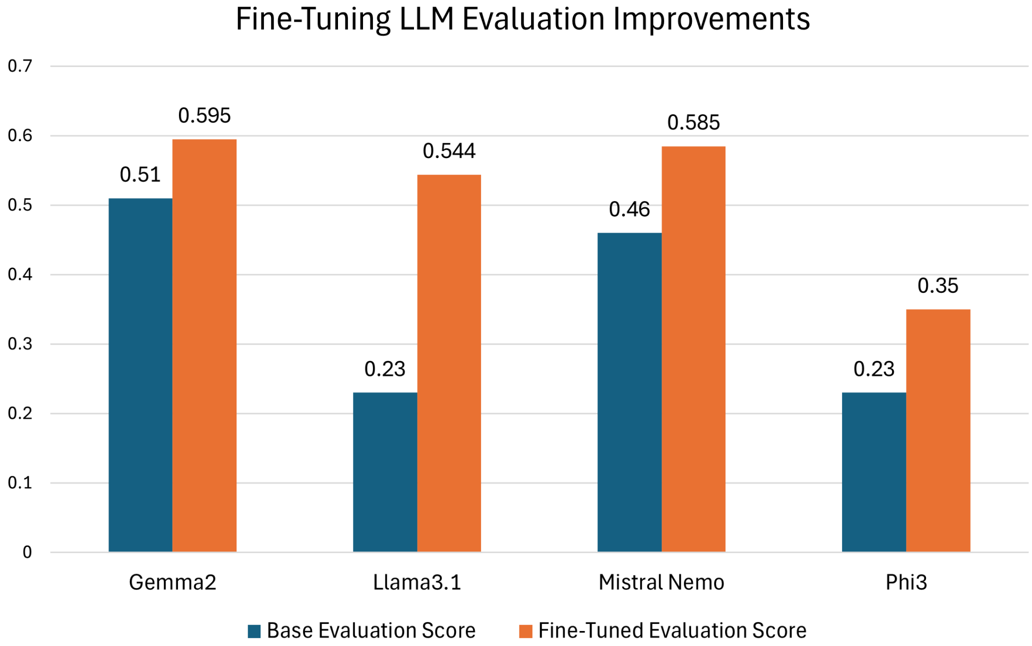
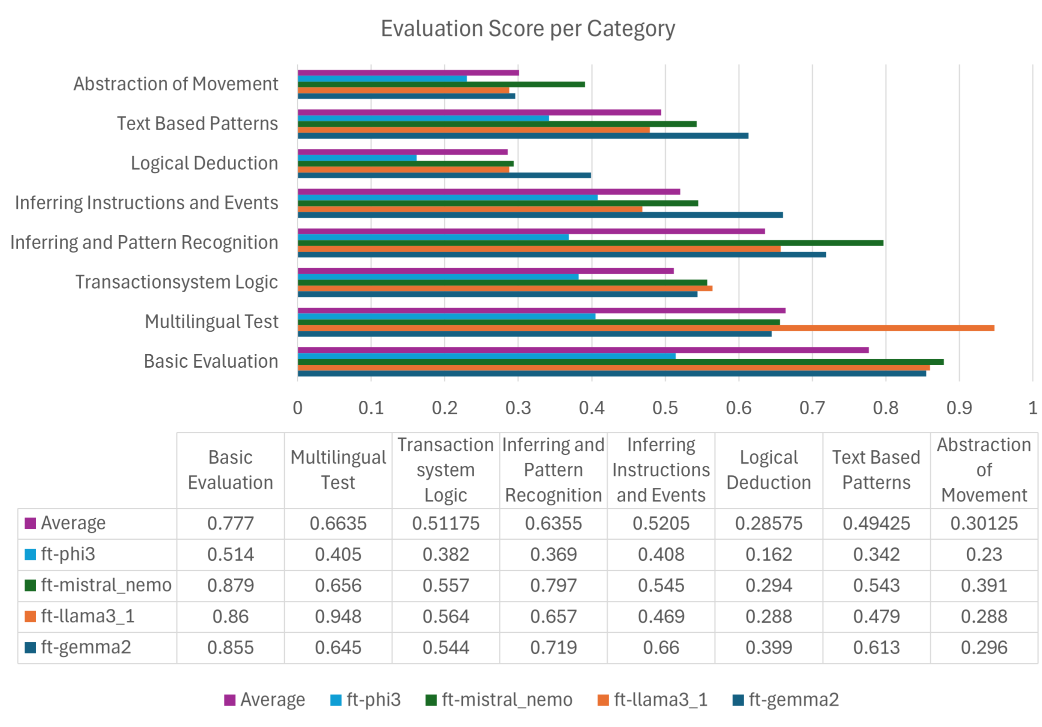


Figure 5.16: Evaluation Score of the LLM's outputs for the evaluation set before and after finetuning.

more by 52.17%, but still has the lowest score of 0.35. Llama3.1 improved the most by 136.50% and achieves a score of 0.544, closely behind Mistral Nemo and Gemma2.

Figure 5.17 shows the evaluation score for each LLM per category of the fine-tuned models, denoted by the 'ft' appendix. In fig. 5.18, the difference in evaluation score of the fine-tuned models to the base models per category can be seen.

The training data seems not to cover the logical deduction category well, with an average regression of around -0.03 for all models, with Phi3 and Mistral Nemo being impacted here by around -0.1. Gemma2 was impacted negatively in the abstraction of movement section, with around -0.166, and in the Multilingual Test section by -0.105. Mistral Nemo was also impacted here by -0.216. The category that resembles the training data the most, the Basic Evaluation set that includes the vocabulary, grew the most on average, with all models except Phi3 reaching above 0.85. This means that for simple requests, which are assumed to be the norm when the system is used, the models are performing well.

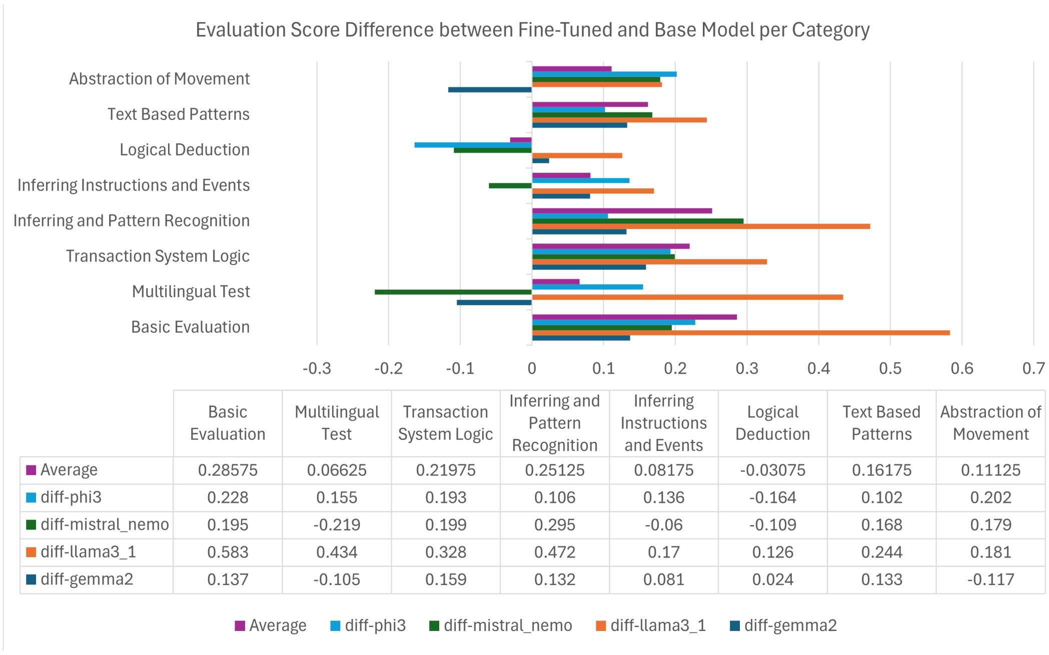


	Basic Evaluation	Multilingual Test	Transaction system Logic	Inferring and Pattern Recognition	Inferring Instructions and Events	Logical Deduction	Text Based Patterns	Abstraction of Movement
■ Average	0.777	0.6635	0.51175	0.6355	0.5205	0.28575	0.49425	0.30125
■ ft-phi3	0.514	0.405	0.382	0.369	0.408	0.162	0.342	0.23
■ ft-mistral_nemo	0.879	0.656	0.557	0.797	0.545	0.294	0.543	0.391
■ ft-llama3_1	0.86	0.948	0.564	0.657	0.469	0.288	0.479	0.288
■ ft-gemma2	0.855	0.645	0.544	0.719	0.66	0.399	0.613	0.296

Figure 5.17: Evaluation Score of each Fine-Tuned LLM per Category using the Default Prompt with guided LLM generation.

Conclusion

In this experiment, the training data for fine-tuning the LLMs and a structure for choosing the rank size for LoRA were presented. A stochastic approach was used to reduce computational resources by finding a suitable sample size for the statistical assumptions. With the assumption of being 95% confident in the results, with 90% statistical power in finding a 1% performance difference, it was determined that a sample size of 7 is sufficient for the needs. Levene's test confirmed the homogeneity of variances between LLM and rank size variations for the samples. Based on these assumptions, rank size 8 was found to be the best choice for all tested models, with a significant difference in performance for all models except Gemma2, which performed well on each rank setting. The fact that a higher rank size performed worse indicates that the models were quick in learning the training data and tended to overfit, leading to worse performance on the evaluation set.



	Basic Evaluation	Multilingual Test	Transaction System Logic	Inferring and Pattern Recognition	Inferring Instructions and Events	Logical Deduction	Text Based Patterns	Abstraction of Movement
■ Average	0.28575	0.06625	0.21975	0.25125	0.08175	-0.03075	0.16175	0.11125
■ diff-phi3	0.228	0.155	0.193	0.106	0.136	-0.164	0.102	0.202
■ diff-mistral_nemo	0.195	-0.219	0.199	0.295	-0.06	-0.109	0.168	0.179
■ diff-llama3_1	0.583	0.434	0.328	0.472	0.17	0.126	0.244	0.181
■ diff-gemma2	0.137	-0.105	0.159	0.132	0.081	0.024	0.133	-0.117

Figure 5.18: Evaluation Score Difference of each Fine-Tuned LLM to the Base LLM per Category using the Default Prompt with guided LLM generation.

The expectation of each LLM being improved after fine-tuning was met, with the worst models in the benchmark, Phi3 and Llama3.1, showing the most improvement. Surprisingly, Llama3.1 improved by more than 100%, while Phi3, the larger model, only improved by around 50%, despite both having almost the same evaluation score before fine-tuning. Mistral Nemo, the model with the same parameter size as Phi3, is now only 1% behind Gemma2, the largest model in the benchmark.

What was found by analyzing the results per category is that the training dataset caused small regressions for some models in the Multilingual Test, Abstraction of Movement, and Logical Deduction categories. This indicates that the training data, which is based on providing basic examples, does not adequately cover these advanced tasks. A follow-up will focus on further improving the models in the next section.

For the best integration advice for LLMs to robotics control systems, it is concluded that fine-tuning gives a beneficial impact on the performance of the LLMs.

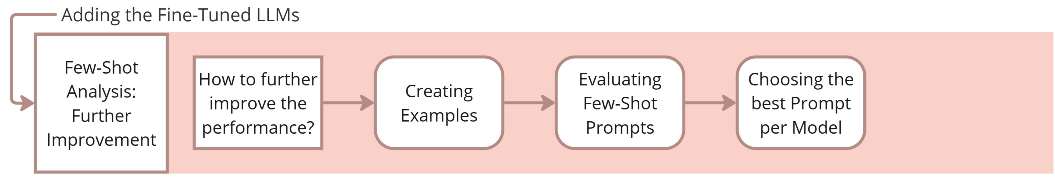

Adding the Fine-Tuned LLMs
Few-Shot Analysis: Further Improvement
How to further improve the performance?
Creating Examples
Evaluating Few-Shot Prompts
Choosing the best Prompt per Model

Figure 5.19: Step four of the Experiment Approach from fig. 4.5.

5.2.3 Few-Shot Examples

Few-shot examples mean that examples are appended to the system prompt. In this section, the fine-tuned models are added to the experiment process, examples are created, and their impact on the LLM's performance is evaluated, following the next step, as seen in fig. 5.19.

Given the fine-tuned and base LLMs, the goal is to further improve their performance by adding examples to the system prompt, as mentioned in the OpenAI prompting guide [32]. It is also important to determine whether LLMs need different examples or if one set works best for all.

The performance of each LLM is expected to improve by adding a few-shot examples, with more examples leading to more performance. It is also expected that each example has a different impact on each LLM, based on their individual performance in the evaluation categories.

First, the examples are presented, followed by a structural analysis of their impact in a single-shot and multi-shot setting, to determine if the goal and expectations were met.

Single-Shot Examples:

For each category, except the two best-performing ones, the basic evaluation and multilingual set, one example per subcategory is created, resulting in 16 examples. These are the more complex tasks that need improvement.

The examples are indexed for each main category, based on the sub-category in the itemization on section 5.1.2.

Single-Shot Analysis:

The experiment begins with analyzing the impact of each single-shot example on the fine-tuned and base LLMs. Each model is evaluated on the evaluation set in its base and fine-tuned state, with the system prompt containing the example at the end.

Figure A.1 in the appendix chapter in appendix A.1 shows the difference in the evaluation score for each model using each single-shot example in the system prompt. Only the Abstraction of Movement 0 example, meaning the driving in geometric forms, had a positive impact on all models, which was the worst performing category for the fine-tuned and base models. It still was not the most impactful example for all models, with each model having a different best-performing example.

These were the best performing examples for each model with fig. 5.20 showing the performance increase:

	gemma2	llama3_1	mistral_nemo	phi3
Base	Transaction System Logic 1	Inferring and Pattern Recognition 0	Inferring Instructions and Events 0	Inferring Instructions and Events 0
Finetuned	Abstraction of Movement 1	Inferring and Pattern Recognition 0	Abstraction of Movement 0	Inferring Instructions and Events 1

Based on these results, it is concluded that each model requires different examples to increase its performance, and no single example fits all models.

Multi-Shot Analysis:

Since the goal is to improve the performance of the LLMs in general, the multi-shot case is now tested, with each model being evaluated with up to five examples for their system prompt. Five shots are considered a reasonable amount, often used in few-shot analysis [14]. This is also set as the maximum to reduce the search space.

One problem faced, as highlighted in [53], is that the selection of examples does not perform equally on all models, as discovered in section section 5.2.3. Additionally, when extending to multiple examples, issues like recency bias or majority bias arise. This means that when multiple examples share a lot of similarity, they are more likely to

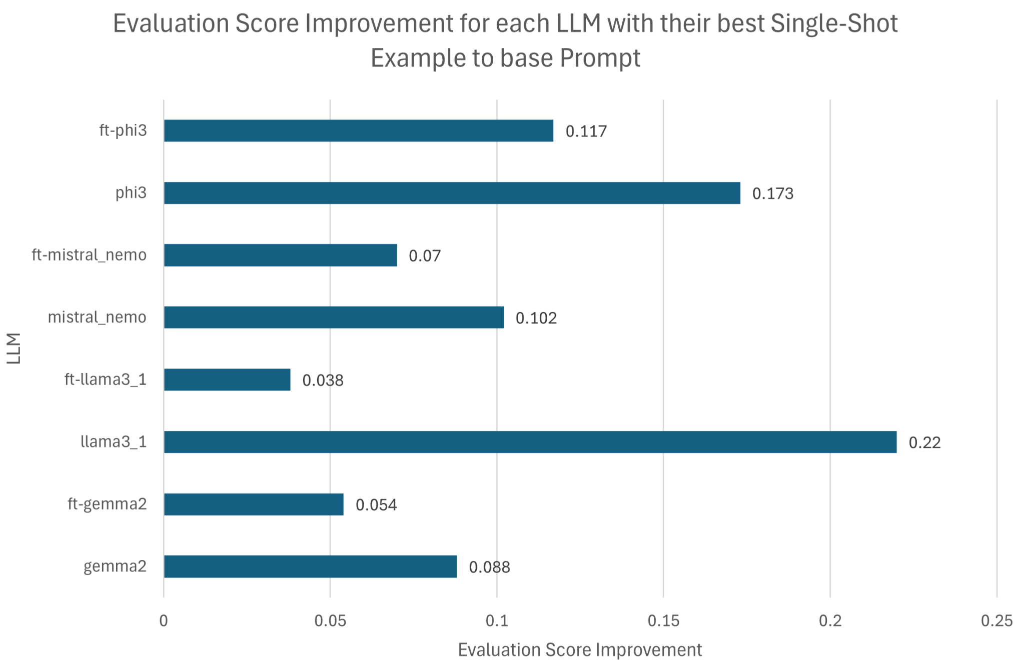


Figure 5.20: Performance increase with the best performing example for each model.

impact the model, but the position of the example—whether it is the first or last—can also influence the outcome. This further complicates the search for suitable example combinations.

A simplified approach to this problem is proposed to reduce the search space:

```
For each model, the examples are sorted by impact,
meaning that the best-performing example is placed
at index 0. The following few-shot example sets are
run for each model based on their sorted examples:
[0,1], [0,1,2], [0,1,2,3], and [0,1,2,3,4].
These examples are then re-run in multiple variations
to reduce the impact of recency bias and majority bias:
with an index offset of one, with an index offset of two,
and with every combination in reverse. For example,
the fifth best example, the third best example, and
the best example are combined, resulting in [4,2,0].
```

This gives a total of 24 multi-shot sets of examples per model, based on their individual results in the single-shot benchmark.

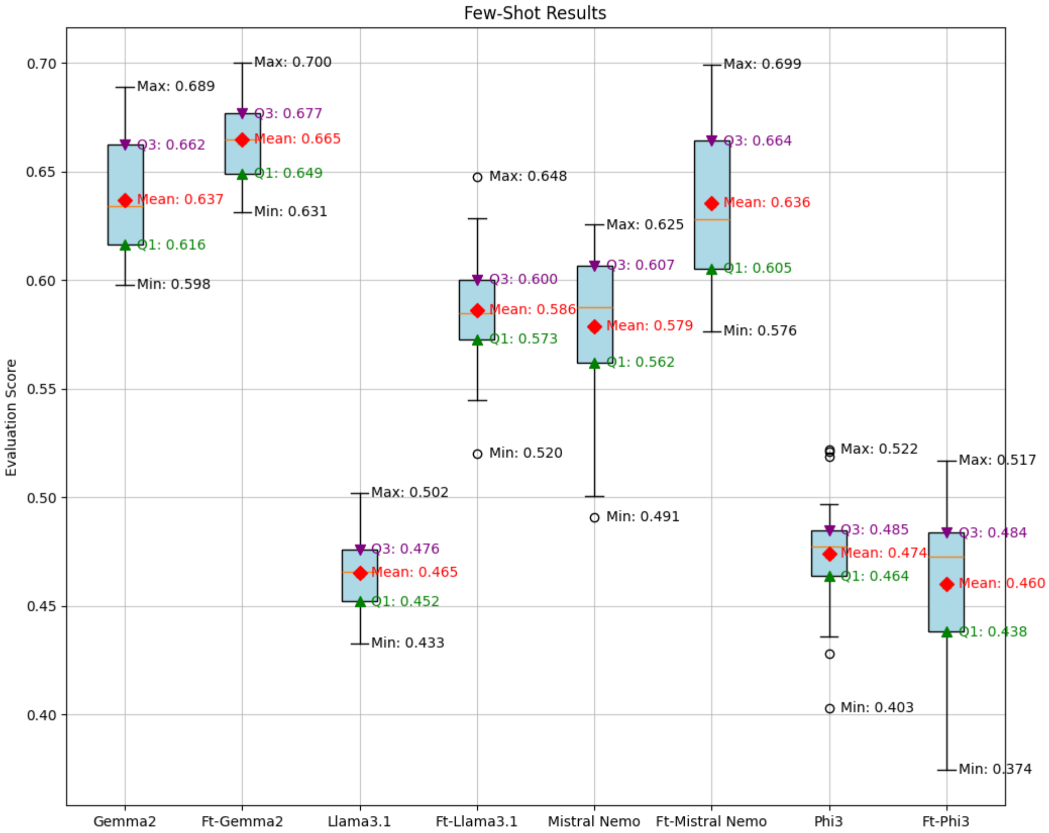
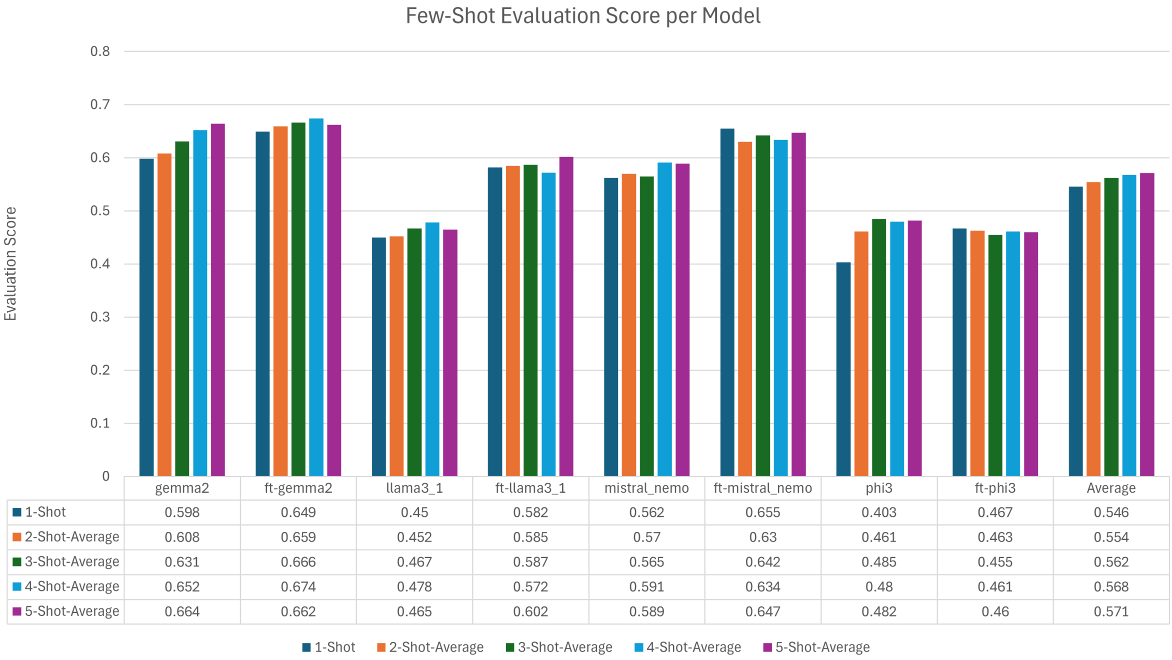


Figure 5.21: Box Plot Evaluation Scores of all Few-Shot Example Sets for each LLM.

Figure 5.21 shows the evaluation score in a box plot for each model in their base and fine-tuned state using each example set. The results show that the performance differs a lot for each model per example set with differences of up to 0.143 in evaluation score between the best and worst performing set. While the difference between fine-tuned and base models is very large, for Llama 3.1 and Mistral Nemo, Gemma2 and Phi3 had very similar results between the base and fine-tuned models. With such big performance impacts, it makes sense to have a broad search space for finding suitable example sets.

Figure 5.22 shows the evaluation score averages per few-shot example set size for each LLM. It shows that each LLM's results averaged perform better with more examples. When examining individual results, it is observed that this applies directly only to Gemma2. For other models, the performance has slight variations in what example size fits the best. This scenario further shows that a variation in example size needs to be included for the search space.



	gemma2	ft-gemma2	llama3_1	ft-llama3_1	mistral_nemo	ft-mistral_nemo	phi3	ft-phi3	Average
■ 1-Shot	0.598	0.649	0.45	0.582	0.562	0.655	0.403	0.467	0.546
■ 2-Shot-Average	0.608	0.659	0.452	0.585	0.57	0.63	0.461	0.463	0.554
■ 3-Shot-Average	0.631	0.666	0.467	0.587	0.565	0.642	0.485	0.455	0.562
■ 4-Shot-Average	0.652	0.674	0.478	0.572	0.591	0.634	0.48	0.461	0.568
■ 5-Shot-Average	0.664	0.662	0.465	0.602	0.589	0.647	0.482	0.46	0.571

Figure 5.22: Evaluation Score Averages per Few-Shot Set Length for each LLM.

Looking at fig. 5.23, the evaluation score of the ascending minus the descending few-shot example sets for each LLM is shown. For example, the score for Gemma2 with the example set [0,1,2] is compared to the score with the example set [2,1,0]. The results show an impact of up to 0.05 in the evaluation score just by changing the order of the examples, indicating a recency bias in the models. Since the results show that there is no consistent trend for each model, it is concluded that including this type of variation in the search space makes sense.

Figure 5.24 shows the evaluation score for the best found few-shot example set for each LLM. The blue bar shows the evaluation score with the example set under it. The orange bar shows the base performance without examples in the system prompt, and the green bar demonstrates the difference. As the fine-tuning already improved the models, the few-shot examples added gave a smaller improvement on these models in comparison to the base models. As previously stated, it is observed that no single order of sorted examples fits all models, with each model having a different best-performing set. For some models, the base models with examples performed even better than the fine-tuned models without examples, e.g., for Gemma2, Mistral Nemo, and Phi3. For Phi3, it is

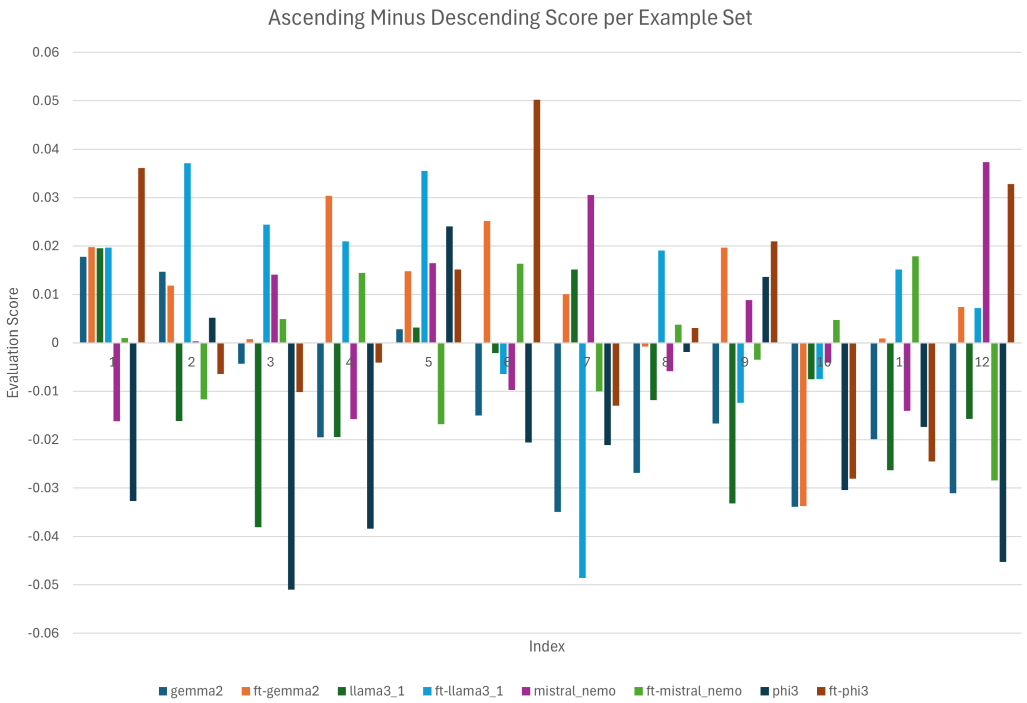
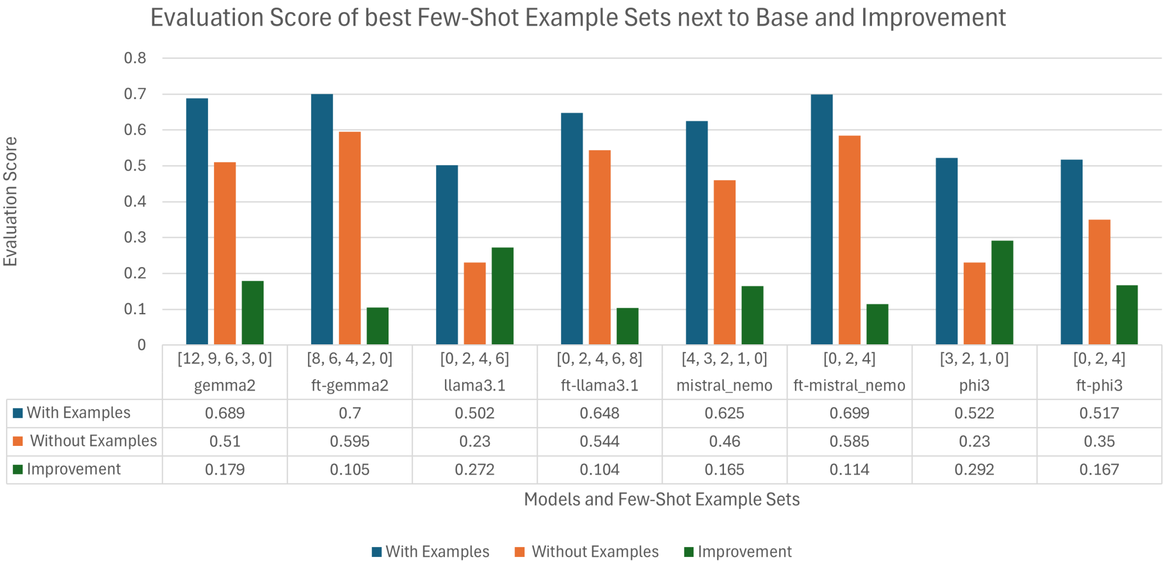


Figure 5.23: Evaluation Score of the Ascending minus Descending Few-Shot Example Sets for each LLM showing the Recency Bias.

even observed that the base model performed slightly better than the fine-tuned model with examples, although the difference is only around 0.05.

Figure 5.25 shows the evaluation score per category for each model with their best found few-shot example set to the base prompt without examples. Only Phi3 shows regressions in ’Logical Deduction’. The fine-tuned and base Mistral Nemo and the fine-tuned and base Gemma2 also show regressions in ’Multilingual Test’. Finally, the fine-tuned Llama3.1 and Mistral Nemo show a small regression in ’Basic Evaluation’. The later two categories were already performing well with the base prompt and were excluded in the search space, with only the ’Multilingual Test’ showing concerning regressions. Since this category is not crucial for the robotics control system, no further optimization is made to improve these regressions. It is concluded that adding examples in one language may negatively impact performance when prompting in another language, as observed in the fine-tuning to base category regressions in fig. 5.18. The other categories improved a lot for each model, especially the worst-performing ones from before, like Abstraction of



	[12, 9, 6, 3, 0] gemma2	[8, 6, 4, 2, 0] ft-gemma2	[0, 2, 4, 6] llama3.1	[0, 2, 4, 6, 8] ft-llama3.1	[4, 3, 2, 1, 0] mistral_nemo	[0, 2, 4] ft-mistral_nemo	[3, 2, 1, 0] phi3	[0, 2, 4] ft-phi3
🟦 With Examples	0.689	0.7	0.502	0.648	0.625	0.699	0.522	0.517
🟧 Without Examples	0.51	0.595	0.23	0.544	0.46	0.585	0.23	0.35
🟩 Improvement	0.179	0.105	0.272	0.104	0.165	0.114	0.292	0.167

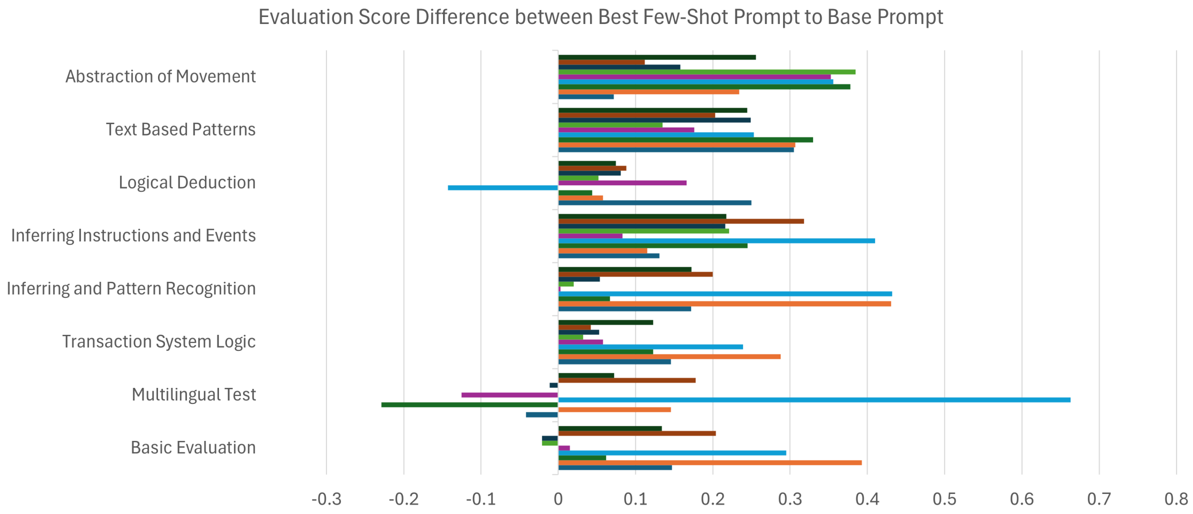
Figure 5.24: Evaluation Score of the best found Few-Shot Example Set for each LLM. The blue bar shows the evaluation score with the example set under it. The orange bar shows the base performance without examples in the system prompt, and the green bar the difference.

Movement. Having few-shot examples for bad-performing contexts helps a lot with the LLMs to improve in these areas.

Conclusion

Concluding this experiment, it was observed that few-shot examples improved the performance of all tested LLMs. The degree of improvement varied significantly depending on the model, the type of example, and the order in which examples were presented. The results met the expectations: more examples generally led to better performance, each example impacted each model differently, and the models were influenced by the order of the examples.

Base models benefit more from the examples than fine-tuned models, with the fine-tuned models having a smaller improvement but still achieving a higher score in the end, except Phi 3, where the base model slightly outperformed the fine-tuned version.



	Basic Evaluation	Multilingual Test	Transaction System Logic	Inferring and Pattern Recognition	Inferring Instructions and Events	Logical Deduction	Text Based Patterns	Abstraction of Movement
■ Average	0.13425	0.0725	0.122625	0.172375	0.217375	0.0745	0.24475	0.256
■ diff-ft-phi3	0.204	0.178	0.042	0.2	0.318	0.088	0.203	0.112
■ diff-ft-mistral_nemo	-0.021	-0.011	0.053	0.054	0.216	0.081	0.249	0.158
■ diff-ft-llama3_1	-0.021	0	0.032	0.02	0.221	0.052	0.135	0.385
■ diff-ft-gemma2	0.015	-0.125	0.058	0.003	0.083	0.166	0.176	0.353
■ diff-phi3	0.295	0.663	0.239	0.432	0.41	-0.143	0.253	0.356
■ diff-mistral_nemo	0.062	-0.229	0.123	0.067	0.245	0.044	0.33	0.378
■ diff-llama3_1	0.393	0.146	0.288	0.431	0.115	0.058	0.307	0.234
■ diff-gemma2	0.147	-0.042	0.146	0.172	0.131	0.25	0.305	0.072

■ Average ■ diff-ft-phi3 ■ diff-ft-mistral_nemo ■ diff-ft-llama3_1 ■ diff-ft-gemma2
■ diff-phi3 ■ diff-mistral_nemo ■ diff-llama3_1 ■ diff-gemma2

Figure 5.25: Evaluation Score Differences per Category for the best found Few-Shot Example Set for each LLM to the Base Prompt without Examples.

Additionally, this experiment highlights challenges associated with optimizing few-shot prompts. The regressions here, even though there are only a few, are bigger than the regressions found after the fine-tuning, especially in the Multilingual Test.

Overall, the proposed approach of sorting examples based on their impact and testing them in different orders and combinations provides an efficient way to optimize few-shot prompting while reducing the search space. Finally, each model surpasses the 0.5 evaluation score, with Gemma2 reaching 0.7, closely followed by fine-tuned Mistral Nemo with 0.699. Surprisingly, the fine-tuned Llama3.1 comes close to the larger models with 0.648, even though it has only 8B parameters.

When comparing the performance gains from fine-tuning, it was found that adding examples to the system prompt had an even greater impact on overall LLM performance than fine-tuning alone. The result suggests that few-shot prompting is a more efficient way to improve model performance for this use case, particularly for larger models. However, the best results were achieved by combining both fine-tuning and few-shot examples, showing the significance of combining the approaches.

Looking at the research questions from section 4.3, a novel systematic approach to integrate LLMs into a robotics control system was successfully presented. The following section concludes the second part of this work's contribution.

5.2.4 Multi-Response Analysis

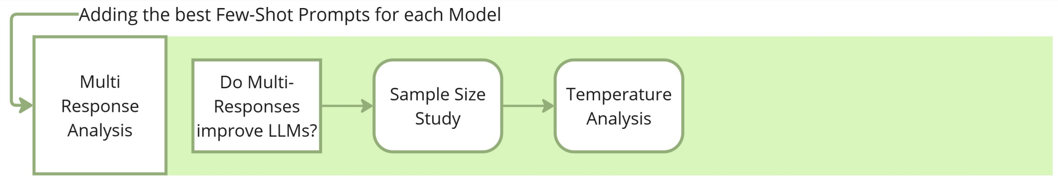

Adding the best Few-Shot Prompts for each Model
Multi Response Analysis
Do Multi-Responses improve LLMs?
Sample Size Study
Temperature Analysis

Figure 5.26: Step five of the Experiment Approach from fig. 4.5.

Figure 5.26 shows the final step of the integration approach, conducting the multi-response analysis. Here, the best-found few-shot example set is added to each LLM and used as their new system prompts.

The impact of the temperature parameter for the integrated LLMs is analyzed to further improve their performance and to reason about the use of the multiple-choice pattern in a robotics control system. The temperature parameter is a hyperparameter that controls the randomness of the LLM's output. A higher temperature leads to more random responses, while a lower temperature leads to more deterministic responses [10], with 0 being deterministic and 1 being a common balance between randomness and determinism.

As another basic condition, the number of multiple responses per LLM is set to three to further save resources. It is also argued that the number of responses should not grow too big, as a user should not be overwhelmed with too many choices.

The goal of this experiment is to analyze the impact of the temperature parameter on the performance of the LLMs to answer the second research question, how well the LLMs scale with multiple responses, and to answer how suitable that approach is for controlling robotic control systems. Two metrics are looked at:

- How good is the best response? Reasoning that the user will pick the best response when faced with multiple choices generated by the LLM in the chatbot.
- How good is the average response? Reasoning what the user will get on average.

Another goal is to determine if there is a suitable temperature setting for each LLM or if each one needs a different one.

The expectation is that the temperature parameter has a significant impact on the LLM's performance, with a growing temperature leading to possibly better answers while performing worse on average since the randomness increases.

This experiment starts with a sample size study to find a suitable sample size for each LLM and temperature combination, followed by the temperature analysis to conclude this work.

Sample Size Analysis:

Since the temperature parameter introduces additional randomness to the outputs of the LLMs, similar to the fine-tuning process, a sample size analysis is again proposed to determine a suitable sample size for the analysis, thereby saving computational resources.

In the following assumptions, the evaluation score is calculated by taking the best response of the three responses per LLM, as the variance is larger in this case compared to averaging the three responses:

- The variance in performance of 50 Llama 3.1 with temperature setting 1.0 and 3 responses is smaller or within Levene's test for homogeneity of variances of the same or other models with the same or lower temperature. The Levene test measures if multiple samples have the same variance [26].
- To determine the required sample size for comparing different LLMs and temperature configurations, collecting 50 samples from Llama 3.1 with the temperature set to 1.0 is sufficient. Using these samples, a sample size analysis is made to estimate the number of observations needed for detecting a performance difference of 2.5% (Cohen's d = 0.25 [9]) with 90% statistical power and a significance level of 0.05.

These values are commonly used in statistical analysis [31], with the detection size being chosen as a trade-off between significant performance differences and computational resources. This means that only 50 runs of evaluating Llama 3.1 with the temperature set to 1.0 are needed to find out the sample size. As long as the variance of each LLM and temperature variation is not significantly different from Llama 3.1 with temperature 1.0, according to Levene's test, the same sample size can be used for comparing LLMs in different temperature settings.

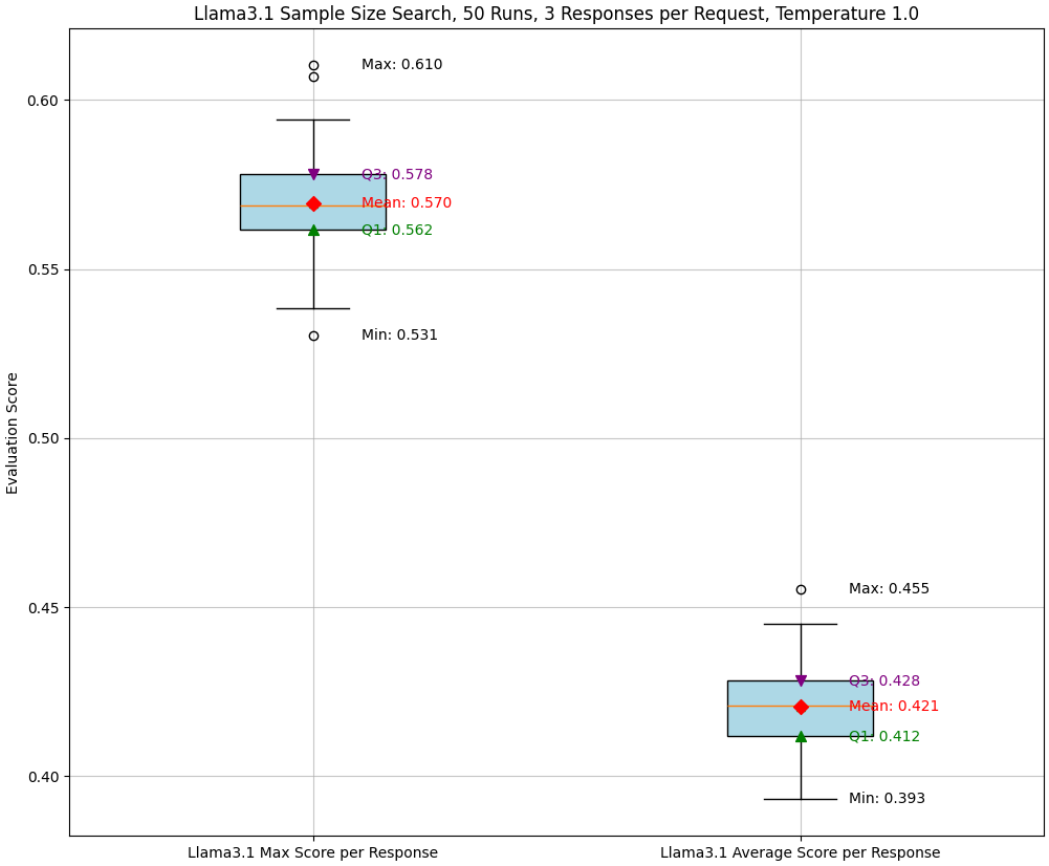


Figure 5.27: Box Plot for evaluating Llama 3.1 50 times on the temperature setting 1.0 by looking at the best response out of 3 and the average out of 3.

The results after 50 runs, shown in fig. 5.27, show for the best response of 3 an average evaluation score of 0.57, with q1, the 25th percentile at 0.562, and q3, the 75th percentile at 0.578. The variance is a bit higher than for the fine-tuning, with 50% of the results being within 1.6% of the performance difference and a minimum score of 0.531 and a maximum score of 0.61. For the average of 3, the average evaluation score is 0.421, with q1 at 0.412 and q3 at 0.428. 50% of the results here are also within 1.6% of the performance difference, with a minimum score of 0.393 and a maximum score of 0.455.

When the sample size needed to detect a performance difference of 2.5% with 90% statistical power and a significance level of 0.05 is calculated, a sample size of 10 is found to be required.

Temperature Analysis:

Here, the sample size of 10 is used to evaluate each LLM with different temperature settings. To further save computational resources, only the temperature settings 0.4, 0.7, and 1.0 are considered, with 1.0 being the trade-off between randomness and determinism, 0.7 being the common default value for LLMs like Chat-GPT [43], and 0.4 representing another step of the same size towards determinism.

For each following experiment, the variances of the evaluation scores were tested using Levene's test for both the average of 3 responses and the maximum of 3 responses. Each test passed, confirming that the sample size of 10 is sufficient for the given assumptions.

The box plots for all LLM results are added to the appendix in appendix A.2. They show the average per 3 answers in fig. A.2 and fig. A.3, and the maximum per 3 answers in fig. A.4 and fig. A.5. The resulting t-tests show a significant difference for the models between the 0.4 and 1.0 temperature, often also with the 0.7 temperature setting.

fig. 5.29 shows the results for the temperature analysis for each LLM, in their base and fine-tuned state, with the average of 3 responses and the maximum of 3 responses, averaged over all 10 samples per LLM and temperature setting. The title of each subfigure shows the LLM name, with the lines showing the maximum and average score for the base and fine-tuned model, denoted by the 'ft' appendix. The x-axis shows the temperature setting, and the y-axis the evaluation score.

Looking at the total average of all models, the average score gets worse per answer with a higher temperature, while the maximum score per answer does not improve after the temperature value of 0.7. Since Gemma2 performed better with a temperature of 1.0 than with 0.7, it is concluded that it may be worthwhile to test the temperature setting for each LLM individually. For the total performance, the fine-tuned models performed better than the base models after the full integration process.

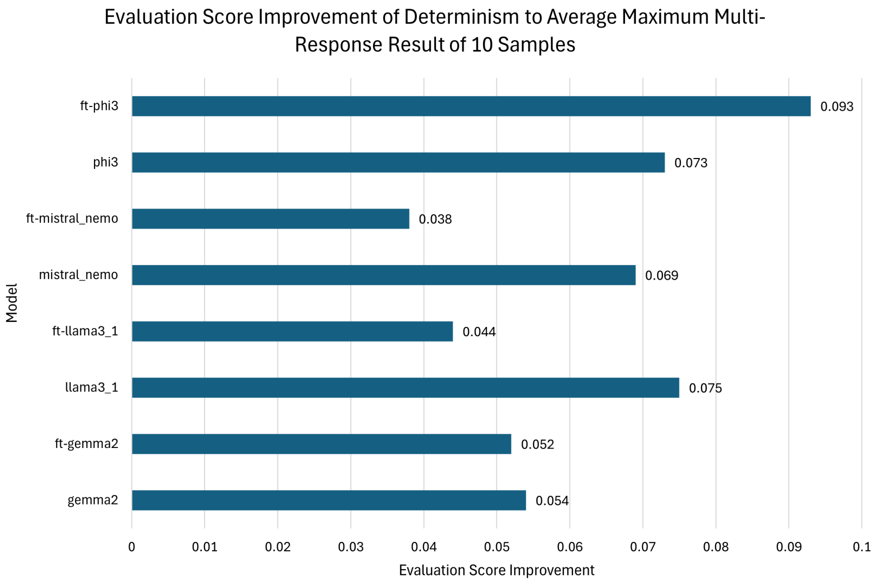
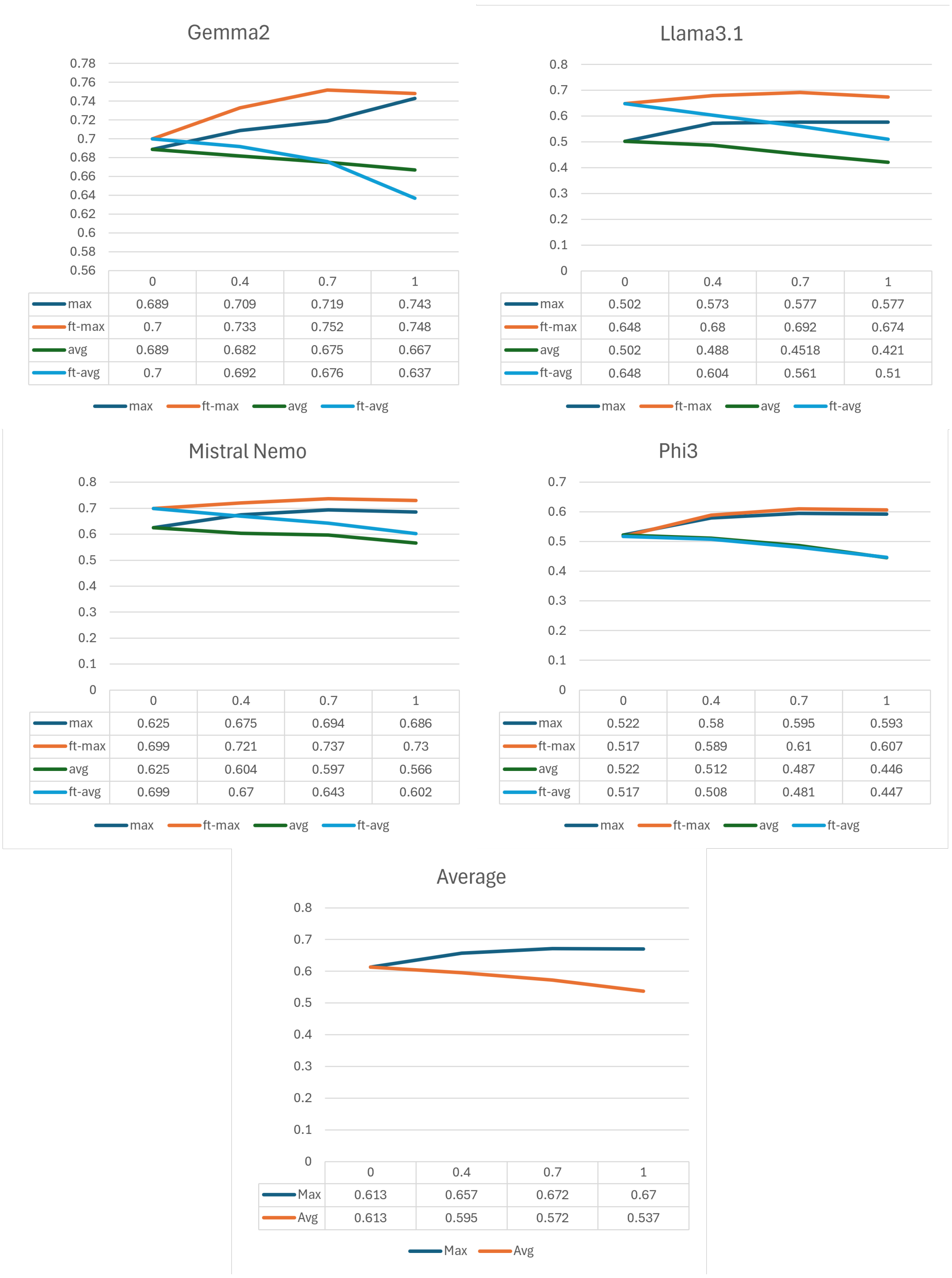


Figure 5.28: Improvement in the evaluation score for each LLM with their best performing temperature setting averaged over 10 samples, looking at the maximum score of 3 responses.

Looking at the average scores for the top three responses from all ten samples in 5.28, it is clear that this method improved all models, with scores ranging from 0.038 to 0.093 and an average improvement of 0.06225.



Gemma2

	0	0.4	0.7	1
max	0.689	0.709	0.719	0.743
ft-max	0.7	0.733	0.752	0.748
avg	0.689	0.682	0.675	0.667
ft-avg	0.7	0.692	0.676	0.637

Llama3.1

	0	0.4	0.7	1
max	0.502	0.573	0.577	0.577
ft-max	0.648	0.68	0.692	0.674
avg	0.502	0.488	0.4518	0.421
ft-avg	0.648	0.604	0.561	0.51

Mistral Nemo

	0	0.4	0.7	1
max	0.625	0.675	0.694	0.686
ft-max	0.699	0.721	0.737	0.73
avg	0.625	0.604	0.597	0.566
ft-avg	0.699	0.67	0.643	0.602

Phi3

	0	0.4	0.7	1
max	0.522	0.58	0.595	0.593
ft-max	0.517	0.589	0.61	0.607
avg	0.522	0.512	0.487	0.446
ft-avg	0.517	0.508	0.481	0.447

Average

	0	0.4	0.7	1
Max	0.613	0.657	0.672	0.67
Avg	0.613	0.595	0.572	0.537

Figure 5.29: Temperature Analysis Results for each LLM. The Y-axis shows the evaluation score of an average of all 10 samples, and the X-axis the temperature. The evaluation contains the results for the average of 3 responses and the maximum value of 3 responses.

Conclusion

In this experiment, the impact of the temperature parameter on the performance of the integrated LLMs was analyzed, with the goal of determining how well the LLMs scale with multiple responses. The objective was to assess how suitable this approach is for controlling robotics systems via a chatbot interface.

Meeting the expectations, it was found that a higher temperature leads to a worse average response, while the maximum response also improves. However, this improvement stopped for almost all models after the temperature setting of 0.7, with only Gemma2 performing better with 1.0.

Concluding the multi-response analysis, which answers the second research question, it is proposed that the temperature setting of 0.7 be used, with the first generated response produced with the deterministic setting of 0.0, ensuring that the user always receives a good first response, while the following responses, though on average worse, are likely to contain a better one.

Looking at the full integration, the fine-tuned models performed better than the base models after the full integration process, which further shows that the combination of all steps is beneficial for the performance of the LLMs.

6 Conclusion

In this work, the task of translating user intent into robotic goals was tackled by using large language models (LLMs). Problems were found with current approaches in the usage of LLMs for translation in the domain of robotics control systems, as well as issues with the robotics control systems themselves. The proposed robotics control system is based on a transaction system with a digital twin for validating and simulating the execution of instructions, allowing for concurrent user interaction without interference while providing an important safety layer. Current approaches directly execute the LLM output, which is found to be problematic due to the hallucinations of LLMs. A chat interface was presented for the user to interact with the LLM to control the robotics control system, where the user can choose between multiple LLM responses to their request instead of executing the LLM output directly. To validate this approach, the main contribution is a novel generalized process for integrating LLMs into robotics control systems, as seen in fig. 6.1.

This process is evaluated on multiple LLMs from different vendors, including Microsoft's Phi3, Meta's LLaMA 3.1, Google's Gemma2, and Mistral's Nemo. Following the integration process steps, it was found that, to make LLMs work in general, a guided generation approach is needed to always obtain valid output from the LLMs for the robotics control system. With that setup, along with the creation of an evaluation dataset and metric, the LLMs were evaluated after each integration step to determine how they improved over this baseline.

In fig. 6.3, the LLM evaluation score over the integration process steps is shown. It can be seen that the LLMs improved significantly over the integration process steps, with an average improvement of 95.2%. Gemma2 improved from 0.51 to 0.752, LLaMA 3.1 from 0.23 to 0.692, Mistral Nemo from 0.46 to 0.737, and Phi3 from 0.23 to 0.61. It is concluded that this work's novel integration process is suitable for integrating LLMs into a robotics control system, with very high scores for the LLMs. Especially when looking at the base usage category in fig. 6.2, where a user would simply type simple sentences,

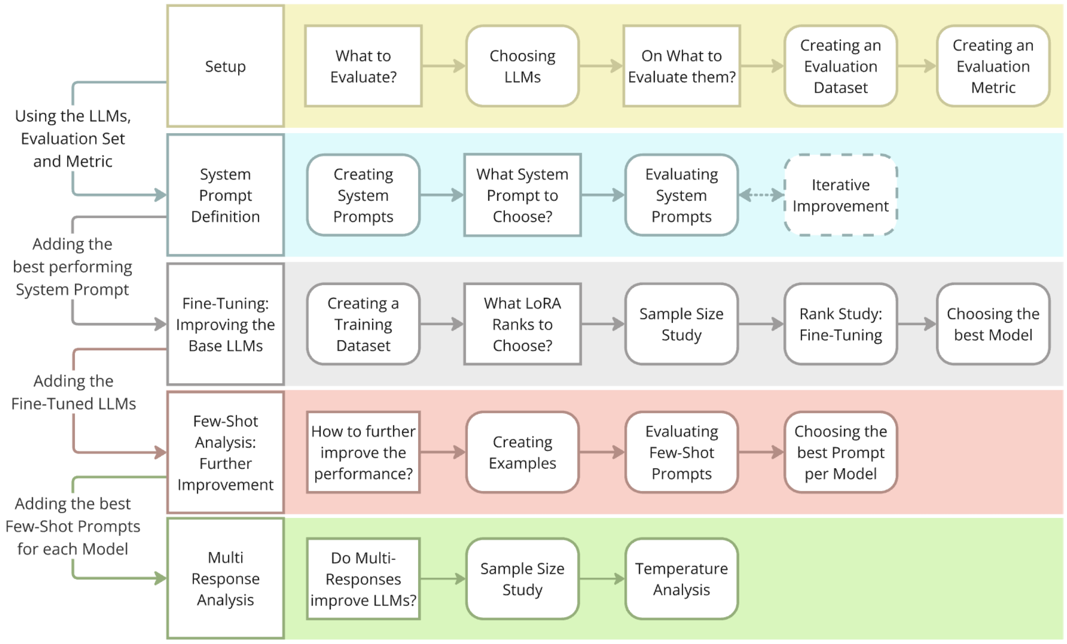
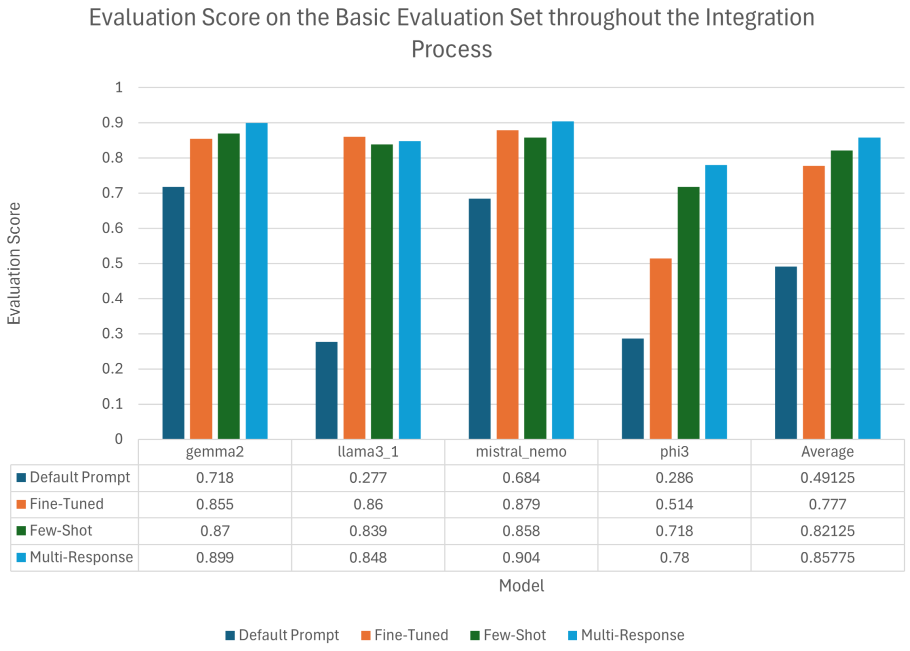
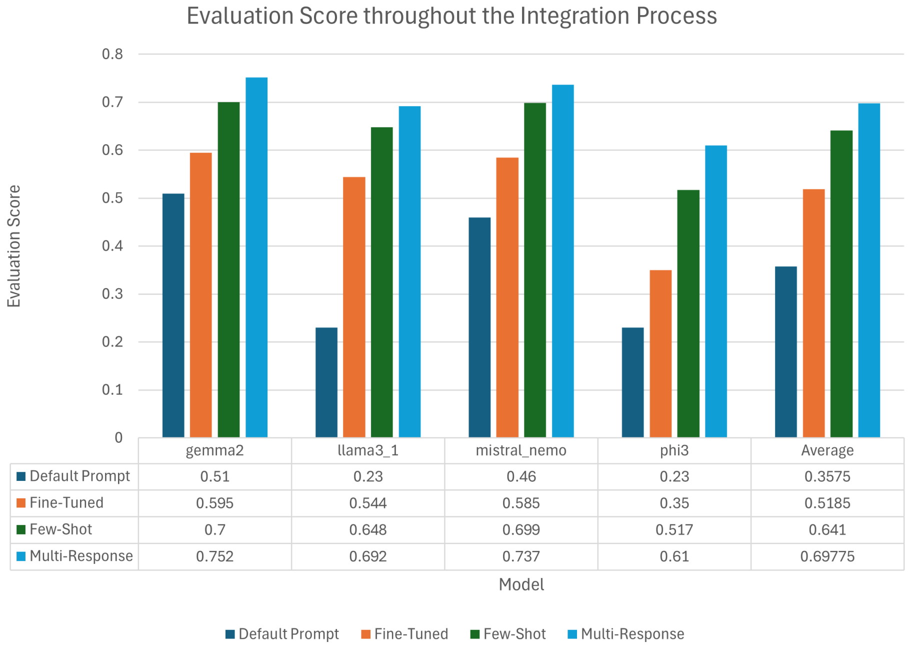

Setup
What to Evaluate?
Choosing LLMs
On What to Evaluate them?
Creating an Evaluation Dataset
Creating an Evaluation Metric
Using the LLMs, Evaluation Set and Metric
System Prompt Definition
Creating System Prompts
What System Prompt to Choose?
Evaluating System Prompts
Iterative Improvement
Adding the best performing System Prompt
Fine-Tuning: Improving the Base LLMs
Creating a Training Dataset
What LoRA Ranks to Choose?
Sample Size Study
Rank Study: Fine-Tuning
Choosing the best Model
Adding the Fine-Tuned LLMs
Few-Shot Analysis: Further Improvement
How to further improve the performance?
Creating Examples
Evaluating Few-Shot Prompts
Choosing the best Prompt per Model
Adding the best Few-Shot Prompts for each Model
Multi Response Analysis
Do Multi-Responses improve LLMs?
Sample Size Study
Temperature Analysis

Figure 6.1: Approach Overview.



	gemma2	llama3_1	mistral_nemo	phi3	Average
Default Prompt	0.718	0.277	0.684	0.286	0.49125
Fine-Tuned	0.855	0.86	0.879	0.514	0.777
Few-Shot	0.87	0.839	0.858	0.718	0.82125
Multi-Response	0.899	0.848	0.904	0.78	0.85775

Figure 6.2: LLM Evaluation Score of the Basic Evaluation Set over Integration Process Steps.



	gemma2	llama3_1	mistral_nemo	phi3	Average
■ Default Prompt	0.51	0.23	0.46	0.23	0.3575
■ Fine-Tuned	0.595	0.544	0.585	0.35	0.5185
■ Few-Shot	0.7	0.648	0.699	0.517	0.641
■ Multi-Response	0.752	0.692	0.737	0.61	0.69775

Figure 6.3: LLM Evaluation Score over Integration Process Steps.

it is observed that the LLMs all reached very high scores, with an average of around 0.86.

These results show that for basic usage, around 86% of the LLMs average output is valid, with Mistral Nemo reaching 90.4%, while for more complex usage, on average, the LLMs still reached around 70%, with Gemma2 reaching around 75% of answer correctness.

Based on these great results, it is concluded that the presented approach is suitable for integrating LLMs into a robotics control system, demonstrating a generalized process for multiple LLMs, all achieving high performance in translating user intent into robotic goals.

Bibliography

[1] *GitHub - serde-rs/json: Strongly typed JSON library for Rust — github.com.* https://github.com/serde-rs/json. – [Accessed 15-02-2025]

[2] Agarwal, Aditya ; Slee, Mark ; Kwiatkowski, Marc: Thrift: Scalable Cross-Language Services Implementation / Facebook. URL http://thrift.apache.org/static/files/thrift-20070401.pdf, 4 2007. – Forschungsbericht

[3] AI, Mistral: *Mistral NeMo — mistral.ai.* https://mistral.ai/news/mistral-nemo/. – [Accessed 11-01-2025]

[4] Clemente, Fabiana ; Ribeiro, Gonçalo M. ; Quemy, Alexandre u. a.: ydata-profiling: Accelerating data-centric AI with high-quality data. In: *Neurocomputing* 554 (2023), S. 126585. – URL https://www.sciencedirect.com/science/article/pii/S0925231223007087. – ISSN 0925-2312

[5] Colledanchise, Michele u. a.: *GitHub - BehaviorTree/BehaviorTree.CPP: Behavior Trees Library in C++. Batteries included. — github.com.* https://github.com/BehaviorTree/BehaviorTree.CPP. – [Accessed 25-01-2025]

[6] Colledanchise, Michele ; Natale, Lorenzo: Handling Concurrency in Behavior Trees. In: *IEEE Transactions on Robotics* 38 (2022), August, Nr. 4, S. 2557–2576. – URL http://dx.doi.org/10.1109/TRO.2021.3125863. – ISSN 1941-0468

[7] Contributors, Chainlit: *Chainlit: Build Conversational AI in minutes.* 2024. – URL https://github.com/Chainlit/chainlit. – Accessed: 2024-11-09

[8] Coumans, Erwin ; Bai, Yunfei: *PyBullet, a Python module for physics simulation for games, robotics and machine learning.* http://pybullet.org. 2016–2021

[9] Diener, Marc: *Cohen's d*, 01 2010. – ISBN 9780470479216

[10] Du, Weihua ; Yang, Yiming ; Welleck, Sean: *Optimizing Temperature for Language Models with Multi-Sample Inference*. 2025. – URL https://arxiv.org/abs/2502.05234

[11] Dubey, Abhimanyu ; Jauhri, Abhinav ; Pandey, Abhinav u. a.: *The Llama 3 Herd of Models*. 2024. – URL https://arxiv.org/abs/2407.21783

[12] Fourrier, Clémentine ; Habib, Nathan ; Lozovskaya, Alina ; Szafer, Konrad ; Wolf, Thomas: *Open LLM Leaderboard v2*. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard. 2024

[13] Fox, M. ; Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. In: *Journal of Artificial Intelligence Research* 20 (2003), Dezember, S. 61–124. – URL http://dx.doi.org/10.1613/jair.1129. – ISSN 1076-9757

[14] Garcia, Xavier ; Bansal, Yamini ; Cherry, Colin ; Foster, George ; Krikun, Maxim ; Feng, Fangxiaoyu ; Johnson, Melvin ; Firat, Orhan: *The unreasonable effectiveness of few-shot learning for machine translation*. 2023. – URL https://arxiv.org/abs/2302.01398

[15] Giessler, Walter: *SIMATIC S7*. 4. VDE Verlag, 2009. – URL http://www.content-select.com/index.php?id=bib_view&amp;ean=9783800738748. – ISBN 9783800738748

[16] Helmert, M.: The Fast Downward Planning System. In: *Journal of Artificial Intelligence Research* 26 (2006), Juli, S. 191–246. – URL http://dx.doi.org/10.1613/jair.1705. – ISSN 1076-9757

[17] Hendrycks, Dan ; Burns, Collin ; Kadavath, Saurav ; Arora, Akul ; Basart, Steven ; Tang, Eric ; Song, Dawn ; Steinhardt, Jacob: *Measuring Mathematical Problem Solving With the MATH Dataset*. 2021. – URL https://arxiv.org/abs/2103.03874

[18] Hu, Edward J. ; Shen, Yelong ; Wallis, Phillip ; Allen-Zhu, Zeyuan ; Li, Yuanzhi ; Wang, Shean ; Wang, Lu ; Chen, Weizhu: *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. – URL https://arxiv.org/abs/2106.09685

[19] Huggingface: *TRL - Transformer Reinforcement Learning — huggingface.co*. https://huggingface.co/docs/trl/index. – [Accessed 17-01-2025]

[20] Izzo, Riccardo A. ; Bardaro, Gianluca ; Matteucci, Matteo: BTGenBot: Behavior Tree Generation for Robotic Tasks with Lightweight LLMs. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oktober 2024, S. 9684–9690. – URL http://dx.doi.org/10.1109/IROS58592.2024.10802304

[21] Kannan, Shyam S. ; Venkatesh, Vishnunandan L. N. ; Min, Byung-Cheol: *SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models*. 2024. – URL https://arxiv.org/abs/2309.10062

[22] Kim, Yeseung ; Kim, Dohyun ; Choi, Jieun ; Park, Jisang ; Oh, Nayoung ; Park, Daehyung: A survey on integration of large language models with intelligent robots. In: *Intelligent Service Robotics* 17 (2024), Sep, Nr. 5, S. 1091–1107. – URL https://doi.org/10.1007/s11370-024-00550-5. – ISSN 1861-2784

[23] Kolve, Eric ; Mottaghi, Roozbeh ; Han, Winson ; VanderBilt, Eli ; Weihs, Luca ; Herrasti, Alvaro ; Gordon, Daniel ; Zhu, Yuke ; Gupta, Abhinav ; Farhadi, Ali: AI2-THOR: An Interactive 3D Environment for Visual AI. In: *arXiv* (2017)

[24] Kong, Xiangrui ; Zhang, Wenxiao ; Hong, Jin ; Braunl, Thomas: *Embodied AI in Mobile Robots: Coverage Path Planning with Large Language Models*. 2024. – URL https://arxiv.org/abs/2407.02220

[25] Latif, Ehsan: *3P-LLM: Probabilistic Path Planning using Large Language Model for Autonomous Robot Navigation*. 2024. – URL https://arxiv.org/abs/2403.18778

[26] Levene, Howard: Robust Tests for Equality of Variances. In: Olkin, Ingram (Hrsg.): *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*. Palo Alto : Stanford University Press, 1960, S. 278–292

[27] Lin, Kevin ; Agia, Christopher ; Migimatsu, Toki ; Pavone, Marco ; Bohg, Jeannette: Text2Motion: from natural language instructions to feasible plans. In: *Autonomous Robots* 47 (2023), November, Nr. 8, S. 1345–1365. – URL http://dx.doi.org/10.1007/s10514-023-10131-7. – ISSN 1573-7527

[28] Liu, Fang ; Liu, Yang ; Shi, Lin ; Huang, Houkun ; Wang, Ruifeng ; Yang, Zhen ; Zhang, Li ; Li, Zhongqi ; Ma, Yuchi: *Exploring and Evaluating Hallucinations in*

LLM-Powered Code Generation. 2024. – URL https://arxiv.org/abs/2404.00971

[29] Macenski, Steven ; Foote, Tully ; Gerkey, Brian ; Lalancette, Chris ; Woodall, William: Robot Operating System 2: Design, architecture, and uses in the wild. In: *Science Robotics* 7 (2022), Mai, Nr. 66. – URL http://dx.doi.org/10.1126/scirobotics.abm6074. – ISSN 2470-9476

[30] Marah Abdin, Sam Ade J. ; Awan, Ammar A. ; Aneja, Jyoti ; Awadallah, Ahmed u. a.: *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. 2024. – URL https://arxiv.org/abs/2404.14219

[31] Minitab, LLC: *Minitab*. – URL https://www.minitab.com

[32] OpenAI: *Prompt engineering*. https://platform.openai.com/docs/guides/prompt-engineering. – [Accessed 12-01-2025]

[33] Ouyang, Long ; Wu, Jeff ; Jiang, Xu ; Almeida, Diogo ; Wainwright, Carroll L. u. a.: *Training language models to follow instructions with human feedback*. 2022. – URL https://arxiv.org/abs/2203.02155

[34] Quigley, Morgan ; Conley, Ken ; Gerkey, Brian P. ; Faust, Josh ; Foote, Tully ; Leibs, Jeremy ; Wheeler, Rob ; Ng, Andrew Y.: ROS: An open-source robot operating system. In: *Workshops at the IEEE International Conference on Robotics and Automation*, 2009

[35] Rein, David ; Hou, Betty L. ; Stickland, Asa C. ; Petty, Jackson ; Pang, Richard Y. ; Dirani, Julien ; Michael, Julian ; Bowman, Samuel R.: *GPQA: A Graduate-Level Google-Proof Q&A Benchmark*. 2023. – URL https://arxiv.org/abs/2311.12022

[36] Schulhoff, Sander ; Ilie, Michael ; Balepur, Nishant ; Kahadze, Konstantine u. a.: *The Prompt Report: A Systematic Survey of Prompting Techniques*. 2024. – URL https://arxiv.org/abs/2406.06608

[37] Segway: *Loomo's packaging contents & hardware specifications — loomo.com*. https://loomo.com/360003429831-Loomo-s-packaging-contents-hardware-specifications.htm. – [Accessed 15-02-2025]

[38] Sprague, Zayne ; Ye, Xi ; Bostrom, Kaj ; Chaudhuri, Swarat ; Durrett, Greg: *MuSR: Testing the Limits of Chain-of-thought with Multistep Soft Reasoning*. 2024. – URL https://arxiv.org/abs/2310.16049

[39] Srivastava, Ankit: *Sense-Plan-Act in Robotic Applications*. 02 2019

[40] Student: The probable error of a mean. In: *Biometrika* 6 (1908), Nr. 1, S. 1–25

[41] Suzgun, Mirac ; Scales, Nathan ; Schärli, Nathanael ; Gehrmann, Sebastian u. a.: *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. 2022. – URL https://arxiv.org/abs/2210.09261

[42] Team, Gemma ; Riviere, Morgane ; Pathak, Shreya ; Sessa, Pier G. u. a.: *Gemma 2: Improving Open Language Models at a Practical Size*. 2024. – URL https://arxiv.org/abs/2408.00118

[43] Victoria, University of: *LibGuides: Prompt Design For Beginners: Set The Temperature — libguides.uvic.ca*. https://libguides.uvic.ca/promptdesign/temp?utm_source=chatgpt.com. – [Accessed 03-03-2025]

[44] Vieira, Inacio ; Allred, Will ; Lankford, Séamus ; Castilho, Sheila ; Way, Andy: *How Much Data is Enough Data? Fine-Tuning Large Language Models for In-House Translation: Performance Evaluation Across Multiple Dataset Sizes*. 2024. – URL https://arxiv.org/abs/2409.03454

[45] Wang, Jiayin ; Ma, Weizhi ; Sun, Peijie ; Zhang, Min ; Nie, Jian-Yun: *Understanding User Experience in Large Language Model Interactions*. 2024. – URL https://arxiv.org/abs/2401.08329

[46] Wang, Ruoyu ; Yang, Zhipeng ; Zhao, Zinan ; Tong, Xinyan ; Hong, Zhi ; Qian, Kun: *LLM-based Robot Task Planning with Exceptional Handling for General Purpose Service Robots*. 2024. – URL https://arxiv.org/abs/2405.15646

[47] Wang, Yubo ; Ma, Xueguang ; Zhang, Ge ; Ni, Yuansheng ; Chandra, Abhranil u. a.: *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. 2024. – URL https://arxiv.org/abs/2406.01574

[48] Willard, Brandon T. ; Louf, Rémi: *Efficient Guided Generation for Large Language Models*. 2023. – URL https://arxiv.org/abs/2307.09702

[49] Wolf, Thomas ; Debut, Lysandre ; Sanh, Victor ; Chaumond, Julien ; Delangue, Clement ; Moi, Anthony u. a.: *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. – URL https://arxiv.org/abs/1910.03771

[50] Wu, Yi ; Xiong, Zikang ; Hu, Yiran ; Iyengar, Shreyash S. ; Jiang, Nan ; Bera, Aniket ; Tan, Lin ; Jagannathan, Suresh: *SELP: Generating Safe and Efficient Task Plans for Robot Agents with Large Language Models*. 2024. – URL https://arxiv.org/abs/2409.19471

[51] Xia, Yinghui ; Chen, Yuyan ; Shi, Tianyu ; Wang, Jun ; Yang, Jinsong: *AICoderEval: Improving AI Domain Code Generation of Large Language Models*. 2024. – URL https://arxiv.org/abs/2406.04712

[52] Xie, Yaqi ; Yu, Chen ; Zhu, Tongyao ; Bai, Jinbin ; Gong, Ze ; Soh, Harold: *Translating Natural Language to Planning Goals with Large-Language Models*. 2023. – URL https://arxiv.org/abs/2302.05128

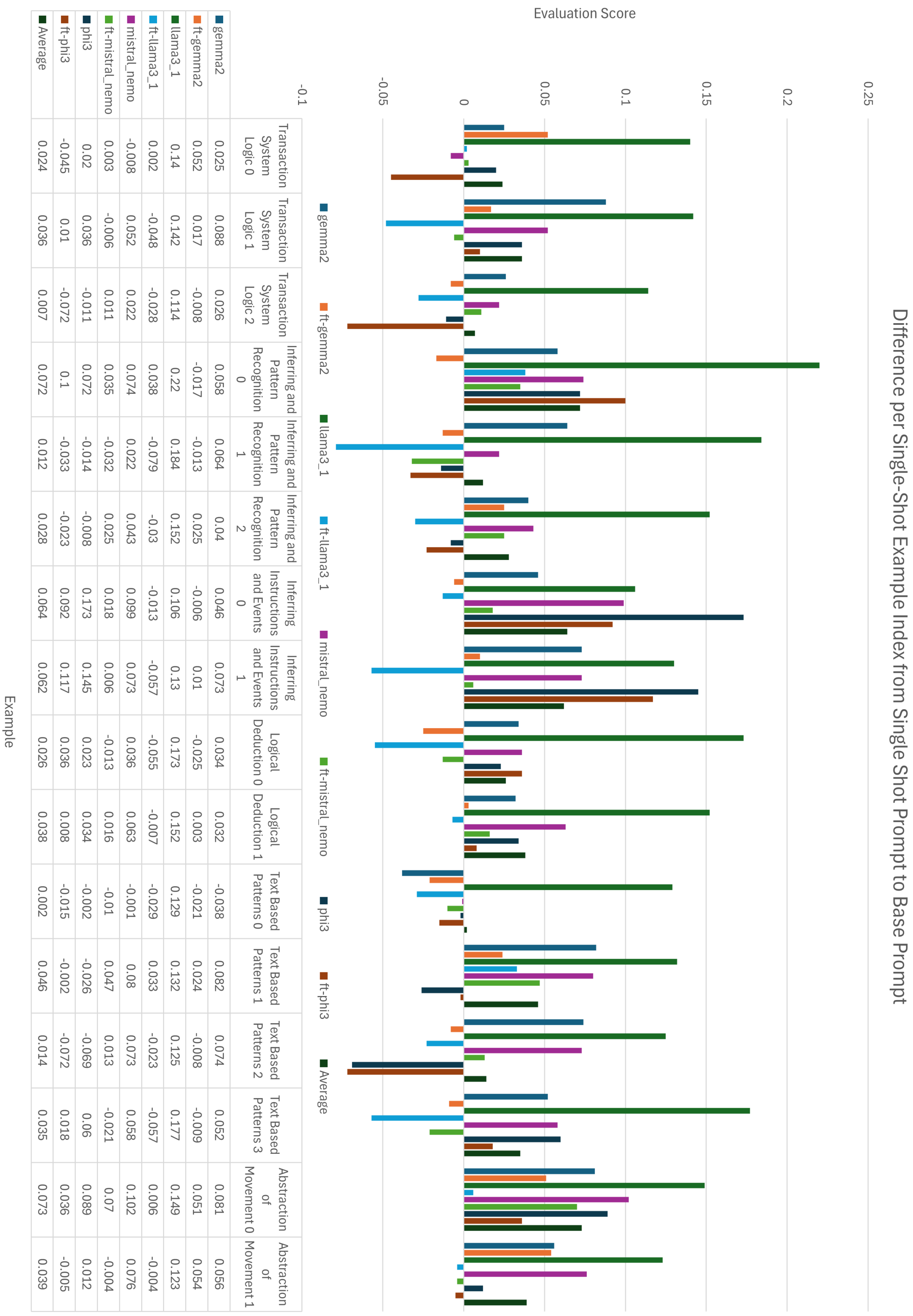
[53] Yoshida, Lui: *The Impact of Example Selection in Few-Shot Prompting on Automated Essay Scoring Using GPT Models*. S. 61–73. In: *Artificial Intelligence in Education. Posters and Late Breaking Results, Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky*, Springer Nature Switzerland, 2024. – URL http://dx.doi.org/10.1007/978-3-031-64315-6_5. – ISBN 9783031643156

[54] Zhou, Jeffrey ; Lu, Tianjian ; Mishra, Swaroop ; Brahma, Siddhartha ; Basu, Sujoy ; Luan, Yi ; Zhou, Denny ; Hou, Le: *Instruction-Following Evaluation for Large Language Models*. 2023. – URL https://arxiv.org/abs/2311.07911

A Appendix

A.1 Single Shot Results

A.2 Temperature Analysis Box Plots



	Transaction System Logic 0	Transaction System Logic 1	Transaction System Logic 2	Inferring and Pattern Recognition 0	Inferring and Pattern Recognition 1	Inferring and Pattern Recognition 2	Inferring Instructions and Events 0	Inferring Instructions and Events 1	Logical Deduction 0	Logical Deduction 1	Text Based Patterns 0	Text Based Patterns 1	Text Based Patterns 2	Text Based Patterns 3	Abstraction of Movement 0	Abstraction of Movement 1
gemma2	0.025	0.088	0.026	0.058	0.064	0.04	0.046	0.073	0.034	0.032	-0.038	0.082	0.074	0.052	0.081	0.056
ft-gemma2	0.052	0.017	-0.008	-0.017	-0.013	0.025	-0.006	0.01	-0.025	0.003	-0.021	0.024	-0.008	-0.009	0.051	0.054
llama3_1	0.14	0.142	0.114	0.22	0.184	0.152	0.106	0.13	0.173	0.152	0.129	0.132	0.125	0.177	0.149	0.123
ft-llama3_1	0.002	-0.048	-0.028	0.038	-0.079	-0.03	-0.013	-0.057	-0.055	-0.007	-0.029	0.033	-0.023	-0.057	0.006	-0.004
mistral_nemo	-0.008	0.052	0.022	0.074	0.022	0.043	0.099	0.073	0.036	0.063	-0.001	0.08	0.073	0.058	0.102	0.076
ft-mistral_nemo	0.003	-0.006	0.011	0.035	-0.032	0.025	0.018	0.006	-0.013	0.016	-0.01	0.047	0.013	-0.021	0.07	-0.004
phi3	0.02	0.036	-0.011	0.072	-0.014	-0.008	0.173	0.145	0.023	0.034	-0.002	-0.026	-0.069	0.06	0.089	0.012
ft-phi3	-0.045	0.01	-0.072	0.1	-0.033	-0.023	0.092	0.117	0.036	0.008	-0.015	-0.002	-0.072	0.018	0.036	-0.005
Average	0.024	0.036	0.007	0.072	0.012	0.028	0.064	0.062	0.026	0.038	0.002	0.046	0.014	0.035	0.073	0.039

Figure A.1: Single Shot Result Difference between Single Shot Prompt and Base Prompt per Model.

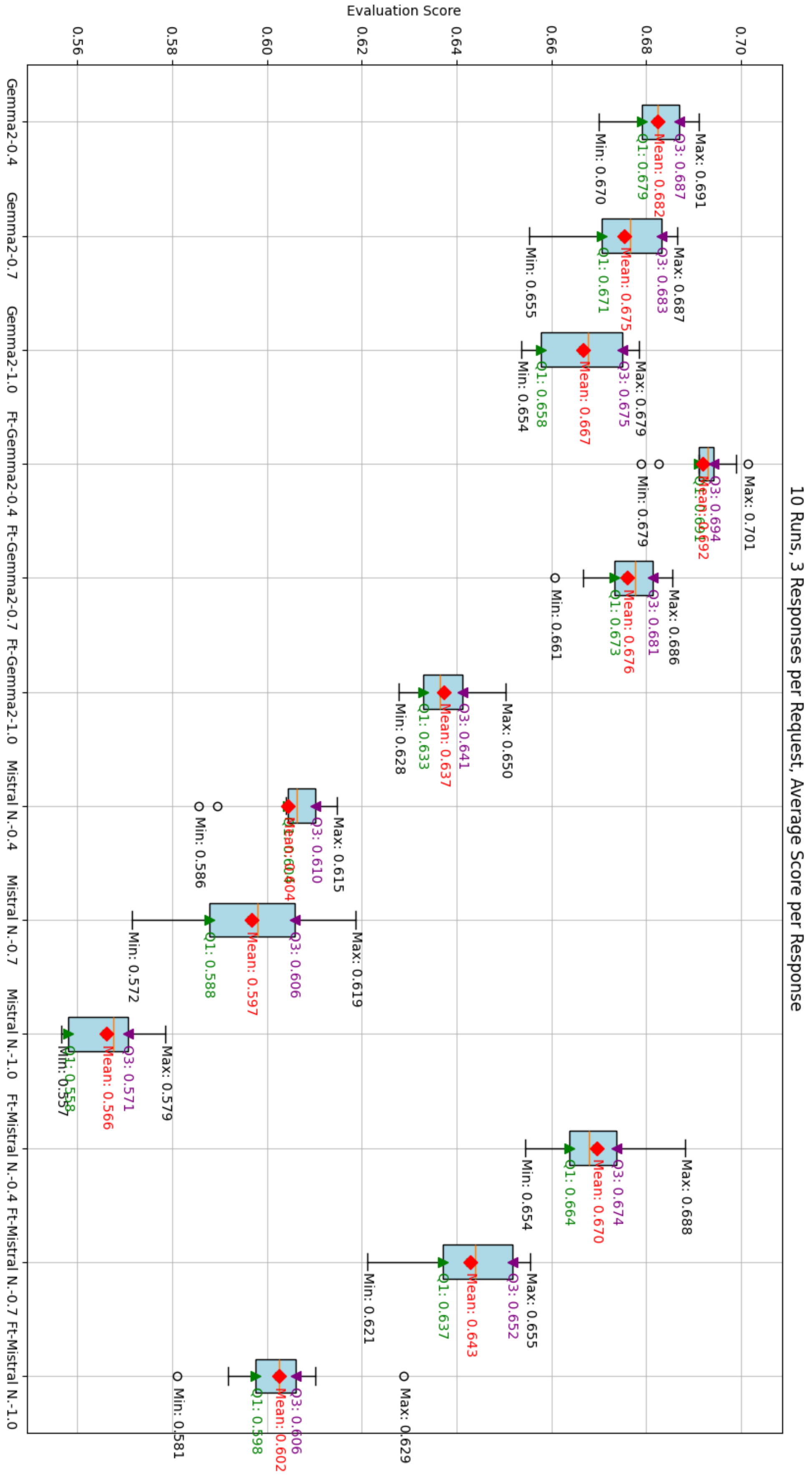


Figure A.2: Box Plot of multiple LLMs with 10 runs each generating 3 responses on the temperatures 0.4, 0.7 and 1.0, calculating the average of the responses.

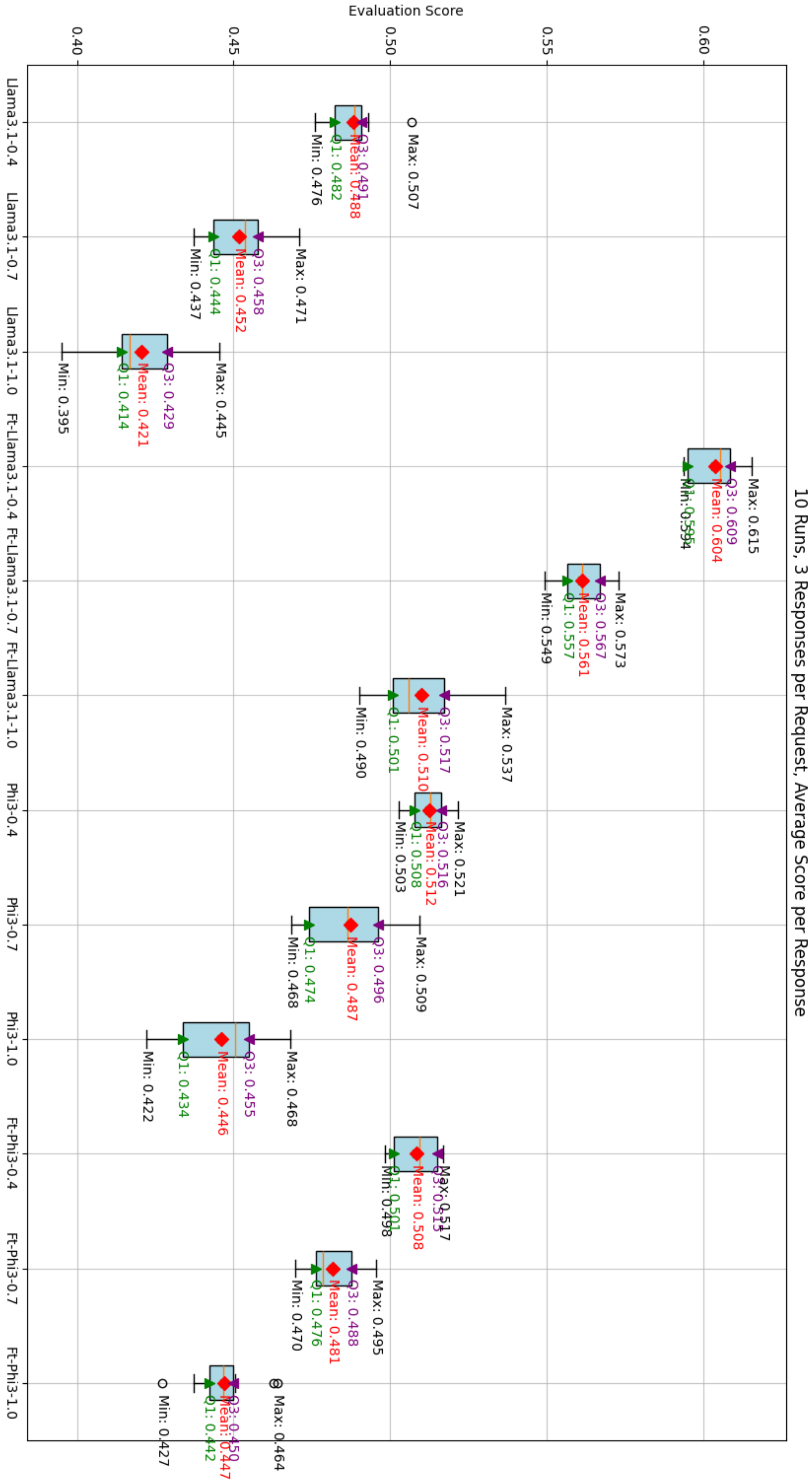


Figure A.3: Box Plot of multiple LLMs with 10 runs each generating 3 responses on the temperatures 0.4, 0.7 and 1.0, calculating the average of the responses.

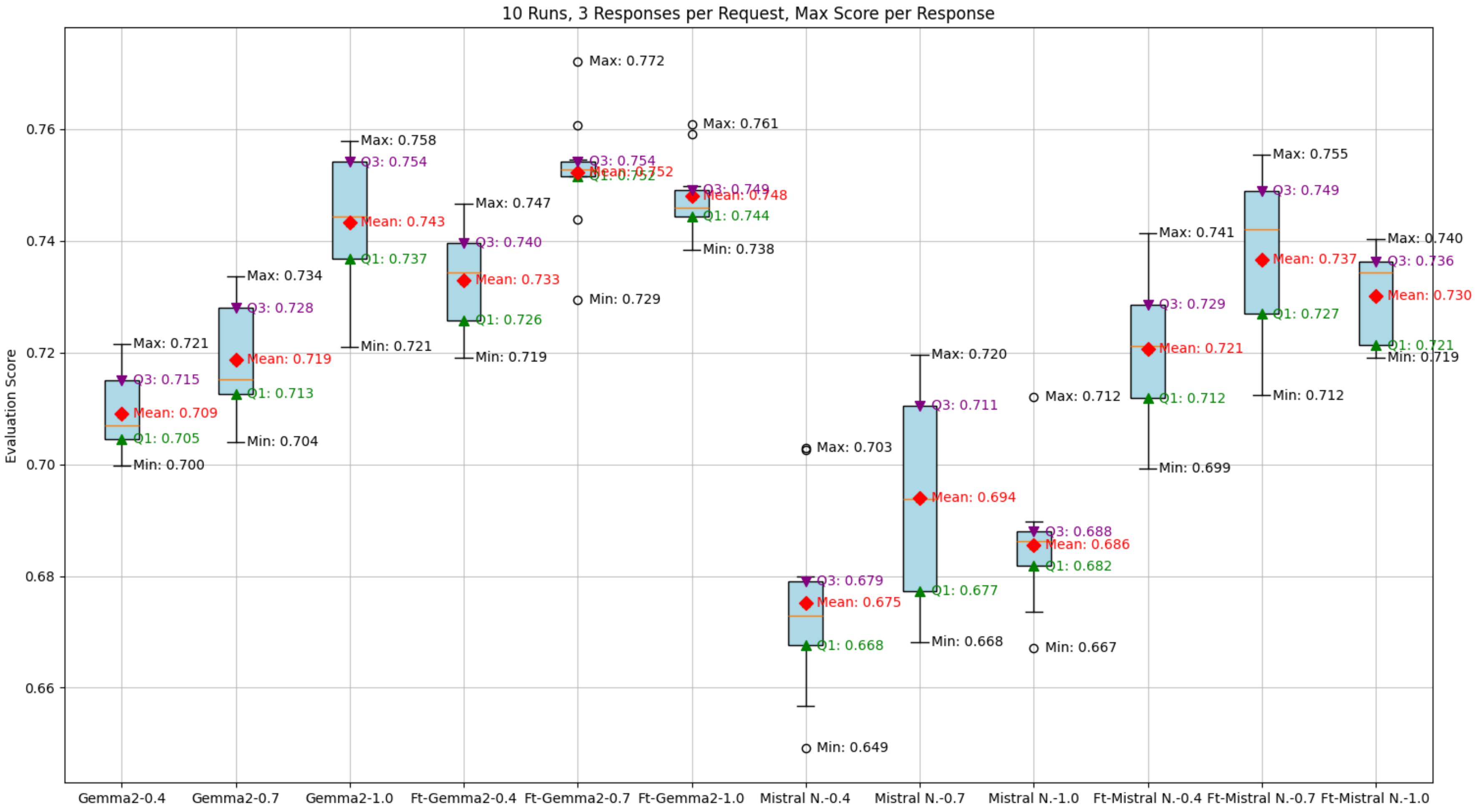


Figure A.4: Box Plot of multiple LLMs with 10 runs each generating 3 responses on the temperatures 0.4, 0.7 and 1.0, calculating the max value of the responses.

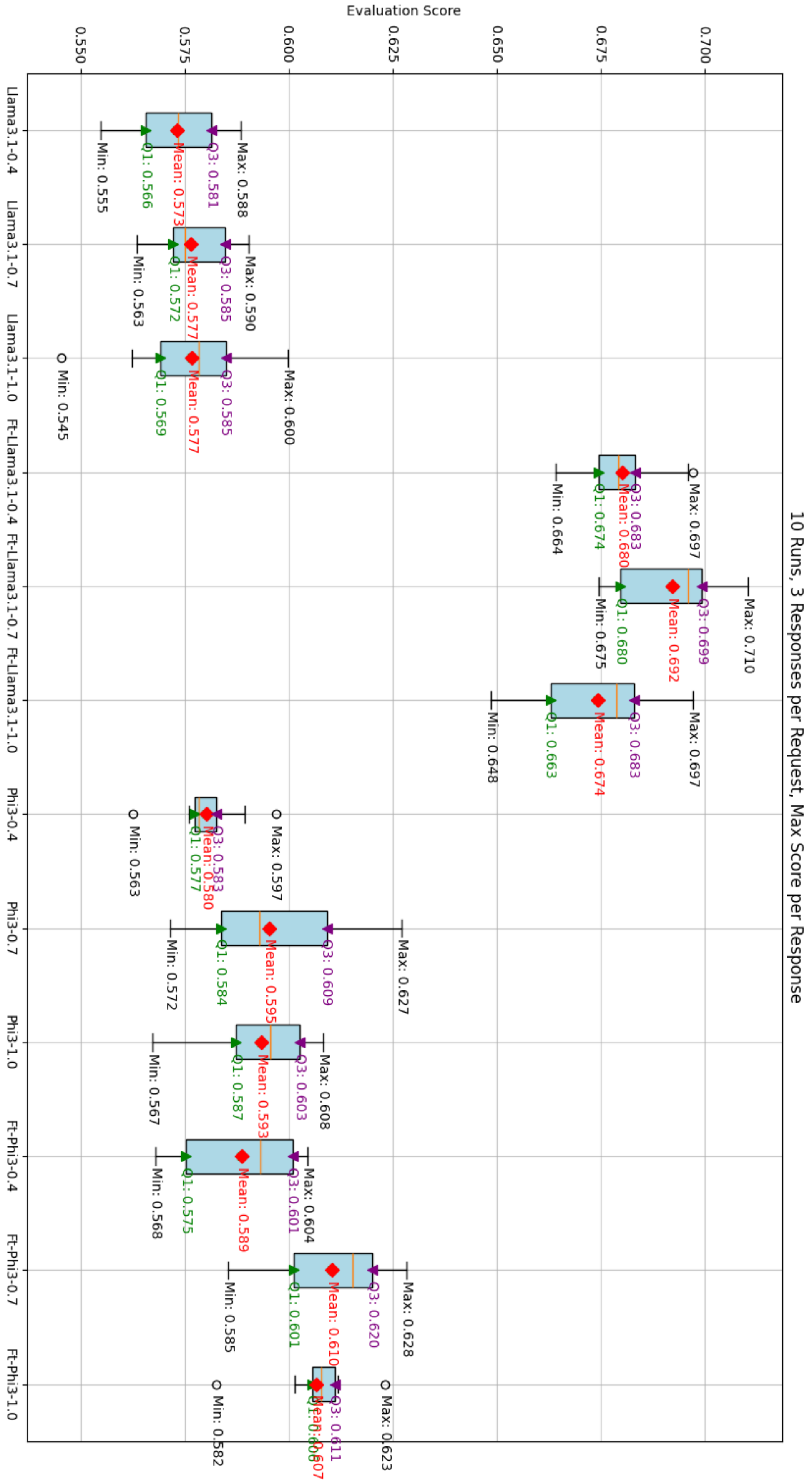


Figure A.5: Box Plot of multiple LLMs with 10 runs each generating 3 responses on the temperatures 0.4, 0.7 and 1.0, calculating the max value of the responses.

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort

Datum

Unterschrift im Original