Universiteit Leiden
The Netherlands

Bachelor Data Science & Artificial Intelligence

Embodied LLMs

Spatial Perception, Understanding, and Prediction

Daniel San Juan Palacios

First supervisor: Dr. Ir. Joost Broekens
Second supervisor: Dr. Peter van der Putten

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

10/02/2025

Abstract

In this thesis, we analyse the ability of LLMs to predict future spatial states and actions given a sequence of previous states and actions. It addresses these challenges by fine-tuning LLMs using data from physical robots, enabling them to predict future states, plan actions, and simulate emotions. This was carried out by collecting a dataset from robots using different policies and fine-tuning LLaMA 3.1 8B under various configurations. This resulted in the LLMs being able to predict low-level future states; however, they struggled with more complex ones. This thesis concludes that while large language models like LLaMA 3.1 8B can learn basic spatial dynamics from unstructured robot data, suggesting their potential as a tool for studying embodied learning, it also reveals areas where further development is needed.

Contents

1 Introduction

Large language models (LLMs) (Section 3.1) have shown remarkable capabilities in text-based tasks such as text generation, summarisation, and translation. However, as we are observing in our everyday lives, their potential extends beyond purely language-based tasks, due to their ability to process abstract tokens. (Section 3.1) In other words, any type of piece of information that can be encoded, such as text, images, and even sound. At the same time, interest and market in embodied and situated intelligence, the integration of AI into physical agents, robots (Section 3.1) is rapidly growing. Intelligent robots typically integrate two systems: an LLM and a distinct movement system. This movement system is usually a high-level symbolic sequence system that runs independently. [KKC+24]

The integration of large language models with symbolic movement systems in embodied AI presents two main challenges. First, the bidirectional information exchange between both systems is constrained by their differing computational paradigms. In this context, LLMs produce high-level, context-dependent (Section 3.1) commands using probabilistic reasoning, whereas symbolic systems require low-level, highly specified commands for motion planning. This hinders the system's ability to join abstract goals with concrete motor operations and to convert sensory feedback to meaningful language. [XWL+24] Second, symbolic systems underperform in real-world environments. Their reliance on fixed rules and discrete state transitions struggles to handle the uncertainty, variability, and partial observability inherent in dynamic settings. [LAK+]

For this reason, and its simplicity, research into unifying these systems into LLMs is quickly developing. As will be explained in Section 2, current approaches rely on the limited inherent spatial knowledge of LLMs and therefore conduct experiments in zero-shot and few-shot settings. [H+24] Due to the lack of training LLMs have in spatial perception, the results are either not favourable or they require large amounts of computing power, as they combine sequentially several LLMs [GSS+24] in an actor-critic manner.

However, while these multi-LLM architectures show promise in controlled simulations, their real-world deployment remains largely impractical. Such architectures are not only power-intensive but also unsuitable for moving in a dynamic environment as well as for real-time interaction. This further motivates the need to explore more efficient alternatives, such as specifically training LLMs for this purpose, so that they can internalise spatial reasoning and planning capabilities without relying on external symbolic modules or ensemble models.

This paper aims to research the application of LLMs to embodied AI. It will focus on fine-tuned LLMs, whose ability to learn and subsequently understand spatial perception remains underexplored. Fine-tuning offers a promising alternative to multi-component architectures, multi-LLM setups, and zero-shot and few-shot approaches by allowing the model to directly incorporate spatial and motor priors from embodied experience. In order to train an LLM in this field, data sets will be needed. For this, data from robots in different environments will be collected. By training these models to predict future states, plan actions, and recognise patterns, this paper aims to bridge the gap between abstract reasoning and real-world interaction.

2 Related Work

Recently, there has been significant attention on merging (LLMs) with embodied AI. However, a major challenge remains in making LLMs effective in real-world environment. Models such as GPT, LLaMA, and DeepSeek are outstanding at language-related tasks, but find it difficult to understand and engage with physical spaces, rendering them unsable for robotic tasks. [H+24] This challenge is the driving force behind this research; while LLMs are skilled in abstract thought, they need to improve in areas like spatial perception and interacting with the real world.

Recent efforts have begun incorporating LLMs into embodied AI as a single system. An example is Huang et al. [H+24], who explored how LLMs can enhance decision-making in robots through zero-shot learning. This paper introduces an LLM as a multi-agent framework that solves robotics tasks without prior training by processing images and task descriptions to generate action sequences. As it relies on multiple agents for planning, action generation, self-correction, and validation, with multiple loops between the generation and self-correction agents, it has a significant latency. This is due to the iterative processing leading to high computational demands. This makes the system impractical for real-time interactive robotics applications.

Similarly, Guruprasad et al. [GSS+24] proposed integrating pre-trained LLMs with visual processing to enhance robotic perception and planning capabilities. Their approach combines the general reasoning skills of LLMs to interpret visual scenes and inform action decisions. However, while their method shows promise in structured environments, it struggles with spatial reasoning in unfamiliar or complex scenes not represented in the training data. The system exhibits limited generalisation when faced with unseen spatial configurations, leading to poor performance in real-world tasks requiring precise spatial awareness and physical interaction. This limitation highlights the current gap between abstract language reasoning and grounded spatial understanding in robotic systems.

LLMs have also been studied in the context of robotic task planning and goal-directed behaviour in environments where spatial understanding is fundamental. Driess et al. [DXS+23] introduced PaLM-E, a large embodied multimodal language model designed to generate action plans from multimodal inputs, including language, vision, and proprioception in simulated environments. While their system demonstrates generalisation and reasoning across experiments, it is tested in controlled virtual settings and relies heavily on pretraining data. Thus, limiting its robustness in unpredictable, partially observable, real-world contexts.

Related, Lynch et al. [LS+23] focused on language-conditioned learning, enabling robots to execute manipulation tasks based on natural language commands. Although this method achieves decent performance within the scope of seen tasks, it remains bound by the limitations of its training data as it cannot handle new instructions or dynamic environments.

In summary, while recent work has shown the potential of LLMs in embodied AI, there are still significant challenges in spatial perception and real-world interaction. This thesis aims to address these limitations by fine-tuning LLMs using data from physical robots, enabling them to predict future states, plan actions, and simulate emotions within a unified architecture, eliminating the need for separate reasoning and control systems or multiple interacting LLMs.

3 Background

This section provides the theoretical foundations necessary to understand large language models, the architecture and design of the chosen model and spatial perception. It begins with an overview of the transformer architecture, which is the foundation of LLMs, introduced in "Attention Is All You Need" (Vaswani et al., 2017) [VSP+17] and its successors, until reaching LLMs. It will cover the different parts of LLMs and highlight the essential theory for fine-tuning LLMs. The second part of the section presents the specific model used in this research, Llama 3.1 8B [GDJt24], including its architecture, training data, and relevant modifications for spatial tasks.

3.1 Transformers & LLMs

The introduction of the transformer architecture by Vaswani et al. [VSP+17] opened a new chapter in artificial intelligence. Unlike earlier models such as RNNs and LSTMs [Elm90, HS97], transformers were able to capture long-range dependencies and parallelise computation effectively. This breakthrough enabled the training of much deeper models on longer sequences, laying the groundwork for today's LLMs. At the core of the transformer lies the self-attention mechanism, which allows the model to assign different importance weights to other tokens in the sequence regardless of their position.

LLMs rely on tokenisation to break input text into smaller discrete units, typically using subword methods such as Byte Pair Encoding or SentencePiece [SHB16, KR18]. These tokens are then embedded into high-dimensional vectors and processed by stacks of transformer blocks. Each block contains multi-head self-attention and a feedforward network, with residual connections and layer normalisation to stabilise training. The use of multi-head attention enables the model to capture different kinds of relationships between tokens in parallel.

Most modern LLMs, including those used in this project, adopt a decoder-only architecture with causal self-attention [RNSS18, RWC+19]. This structure ensures that predictions are based only on past tokens, making it suitable for autoregressive tasks such as next-token prediction. Once trained, these models generate output step by step, using sampling strategies like top-k, top-p, and temperature scaling to guide generation and control diversity [HBD+20, FLD18]. A full explanation of these architectural components and training techniques is provided in Appendix A.

3.2 Llama 3 & 3.1

The **LLaMA (Large Language Model Meta AI)** is Meta AI's series of open-access large language models. It was designed as a high-performance alternative to closed-source models such as OpenAI's GPT or Anthropic's Claude. LLaMA 1 [TLI+23] demonstrated that relatively smaller models could outperform larger closed-source models when trained on high-quality data. LLaMA 2 introduced instruction tuning and safety improvements that enabled widespread adoption. The most recent release, LLaMA 3 (2024), includes further refinements in model architecture and training scale. It brings larger context lengths, more efficient attention mechanisms, and enhanced tokeniser coverage across multiple languages. LLaMA 3.1 builds on LlaMa 3, improving training stability and inference performance, including optimised rotary embeddings, updated normalisation schemes, and further tuning of positional extrapolation behaviour [AI24].

LLaMA 3.1 8B is a decoder-only transformer model, based on the architecture introduced in the original transformer paper [VSP+17] and developed further in earlier LLaMA versions [TLI+23]. It consists of 32 transformer layers, with a hidden size of 4096 and 32 attention heads. Like its predecessors, LLaMA 3.1 discards the encoder-decoder structure of the original transformer with a unidirectional stack of self-attention blocks, suitable for autoregressive language modelling. [GDJt24]

These decoders in LLaMA 3.1 make use of **Group Query Attention (GQA)**, [GDJt24], a combination of the standard multi-head attention and multi-query attention. In standard multi-head attention, each attention head has its own set of key, query, and value projections, which can be computationally intensive at inference time due to the number of key-value pairs that must be computed and stored. Multi-query attention simplifies this by using a single set of key and value projections shared across all heads, which greatly reduces memory usage and increases decoding speed, but at the potential cost of representational capacity. GQA extends this idea by grouping multiple query heads to share the same key and value projections, thereby striking a balance: it retains much of the speed and memory efficiency of multi-query attention while preserving some of the expressivity of multi-head attention [Sha19]. This makes GQA particularly effective for deployment in resource-constrained environments.

To encode token positions, LLaMA 3.1 does not use absolute or learned positional embeddings, as the aforementioned transformer models do. Instead, it employs **Rotary Positional Embeddings (RoPE)** [SLP+21, GDJt24], a method that encodes relative positional information directly within the attention mechanism by rotating the query and key vectors in multi-head attention. As explained in Section 3.1, attention is calculated by the dot product of the query and attention. The dot product is sensitive to angular relationships. Thus, by rotating in a deterministic manner based on position, every query and key embedding vectors, RoPE effectively injects positional information into the attention scores. Unlike fixed absolute encodings, RoPE enables the model to better capture relative distances between tokens and generalise to sequences longer than those seen during training. This is known as positional extrapolation. In LLaMA 3.1, RoPE is further enhanced by scaling strategies that adjust the rotation frequency to maintain attention quality even beyond the training context length.

Another important architectural component is the use of **SwiGLU** activation functions [Sha20, GDJt24] in the feed-forward layers. SwiGLU is a gated variant of the ReLU family, meaning it modulates one input by another through element-wise multiplication, similar to how LSTMs work. Specifically, it combines

two linear projections of the input: one passed through a non-linear activation function (such as `SiLU`), and the other passed through a sigmoid function that acts as a gate, selectively allowing or suppressing information.

$$\text{SwiGLU}(x) = \text{SiLU}(xW_1 + b_1) \odot (xW_2 + b_2)$$

This gating mechanism enables the model to learn more complex and dynamic interactions between features than standard activations like ReLU or GELU.

Normalisation in LLaMA 3.1 is performed using **Root Mean Square Layer Normalisation (RMSNorm)** [ZT19, GDJt24], a simplified variant of standard LayerNorm. Unlike LayerNorm, which centres and scales each input vector by subtracting the mean and dividing by the standard deviation, RMSNorm removes the centring step and only normalises the input by its root mean square (RMS) value. This reduces computational overhead and improves numerical stability, especially in large-scale models with many layers.

These architectural choices allow LLaMA 3.1 8B to strike a balance between computational efficiency and modelling power [GDJt24], making it suitable for research applications like this thesis, which require both moderate hardware resources and high language modelling accuracy.

The model was pretrained using a causal language modelling objective on a dataset reported to contain over 15 trillion tokens, significantly larger than that used in LLaMA 2. The vocabulary consists of 128,000 tokens produced using a Byte-Pair Encoding (BPE) tokeniser, compatible with the tiktoken format. The model supports a context window of 8,192 tokens, enabling it to process longer and more complex sequences than previous generations. [AI24] The training corpus was curated from a mixture of publicly available and licensed sources. These include filtered web crawls, code repositories, technical documentation, scientific papers, and multilingual content. Although the dataset is not publicly released, it is described as "high-quality and diverse" [AI24]. This involves deduplication, filtering by perplexity, document length, and toxicity or safety classifiers. [GDJt24]

The tokeniser incorporates special tokens that serve as control mechanisms to distinguish between different types of content, such as conversation turns, system instructions, and generation boundaries, enabling structured interaction with the model [TLI+23]. These special tokens function as learned behavioural triggers that guide the model's response patterns and output formatting, allowing for more precise control over generation behaviour than natural language instructions alone [OWJ+22].

The whole architecture that was just explained can be summarised through Figure 1.

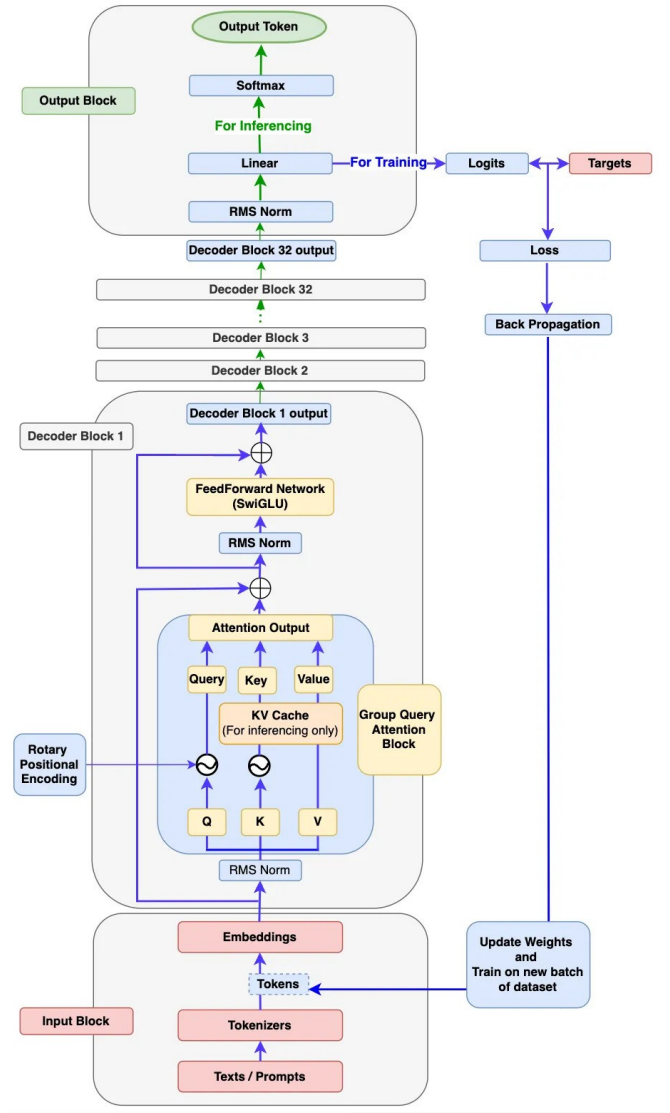


Figure 1: [Tam24] LLaMA 3 Architecture Diagram

4 Research Question

The central research question of this thesis is: Can sensor and actuator data collected from physical robots be effectively used to fine-tune LLMs for spatial perception and spatial understanding and thus, future state prediction?

This research explores whether spatial-temporal data, recorded at high frequency from physical robots navigating structured arenas, can serve as a viable training source for LLMs. The collected datasets encode sensor readings such as distance measurements, gyroscopic rotation, and wheel speeds and differences. By fine-tuning LLaMA 3.1 on this sequential data, we investigate the model's capacity to infer spatial structure and predict likely future states. While LLMs have shown strong performance in text-based reasoning tasks, their application to embodied, sensor-driven domains like robotics remains largely unexplored. This thesis aims to bridge that gap, evaluating whether transformer-based models can extract implicit spatial rules from real-world robot scenarios.

4.1 Hypotheses

H1. Sensor and actuator data can be collected from simple mobile robots in a way that is consistent, varied, and structured, preserving meaningful temporal and spatial features.

H2. LLMs are able to process spatial data as input and learn from it during training.

H3. Given a sequence of spatial data points, the LLM is able to predict future plausible states.

H4. Given a sequence of spatial data points, the LLM is able to predict future plausible actions.

5 Methodology

In this section, the methodology employed to conduct our experiments is described. It begins by covering the material used for the data collection and is later used for data processing, LLM fine-tuning, and LLM deployment. It then describes the setup of the data collection process and setup and configurations for the large language model fine-tuning. And we conclude this section by stating and justifying the chosen measurements for each of the hypotheses.

5.1 Materials

In this section, we describe the materials used in our experiments, including both hardware and software components. We also detail the experimental setup that enabled reliable data collection from the robotic platforms.

5.1.1 Data Collection

The first experiment that was carried out was the data collection. For this experiment, we used mBots.[Mak] The mBot is a robot Do-It-Yourself kit that allows for high customisation of the structure, allowing to merge of several kits together into a more complex structure. The core of the mBot robot is an Arduino board based on Arduino Uno, and includes a Bluetooth low consumption 2.4 GHz chip. The board comes with 4 RJ25 ports to connect the peripherals. Controlled by the board, we have 2 geared motors, with a rotational speed of 200 RPM, with an error of $\pm 10\%$. The whole system is powered by 4 AA batteries. [Mak24] From the set, we also used the UltraSonic sensors, which work by emitting high-frequency sound waves and measuring the time it takes for the echo to return after bouncing off a nearby object, allowing them to calculate distance based on the speed of sound. [MT21] The basic mBot architecture after being built is shown in Figure 2.

On top of the basic design, a 3-Axis accelerometer and gyroscope were implemented on the back part of the robot. The chosen chip was the MakeBlock adapted version of the MPU6050 accelerometer and gyroscope due to its compatibility and easy connectivity with the mCore board through the I^2C interface. The accelerometer component measures linear acceleration along the x, y, and z axes, allowing the system to detect directional movement and inclination. The x and y axes correspond to horizontal motion on the plane of the robot (forward–backwards and left–right), while the z-axis detects acceleration in the vertical direction (e.g., bumps). The gyroscope, in turn, measures the angular velocity around each of these axes, and in this project, the z-axis gyroscope value (rotation around the vertical axis) was used to compute the change in heading or orientation. [Inv13] We also implemented 2 more ultrasonic sensors, on top of the default front one, obtained from other spare kits. These were respectively positioned on the left and right of the robot. The final design of the robot can be seen in Figure 4.
The arenas for the robots were made with assemblable cardboard panels for the exterior walls and the proper boxes of the mBots for the inside obstacles.

Five policies were developed for the robots' data collection to ensure generalisation and variety of actions. The first policy is **Random Navigation with Wall Avoidance**. This policy drives the robot forward until it detects a wall within a predefined threshold. Upon detection, it selects a new direction based on a probabilistic strategy. If both sides are clear, the robot turns randomly left or right,

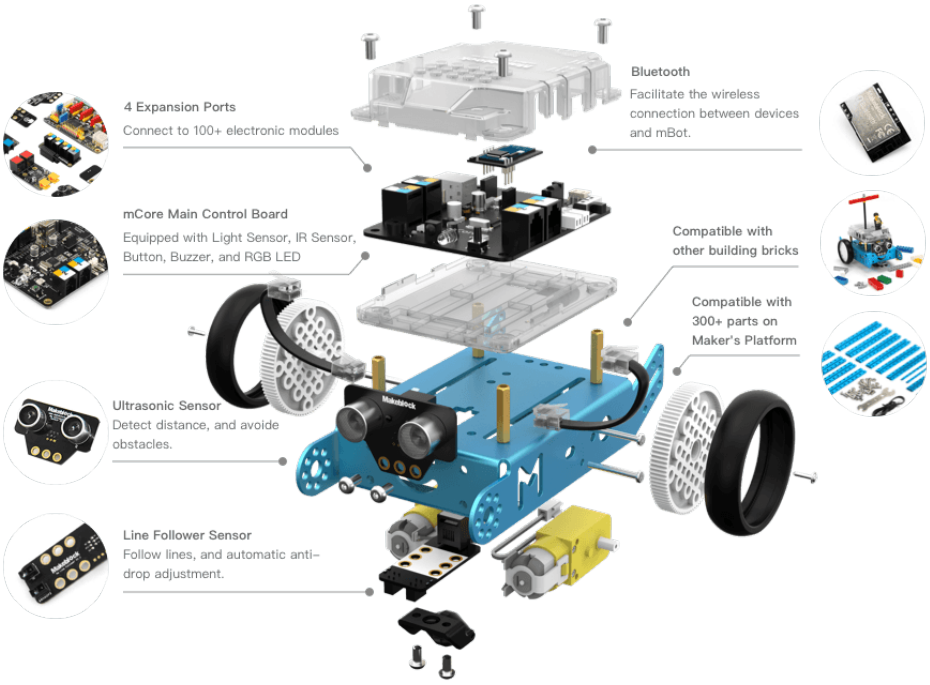


Figure 2: [Mak] Architecture of the basic mBot.

with a small chance of favouring one direction over the other depending on proximity to surrounding walls. If one side is blocked, it always chooses the safer direction. The rotation angle is randomised, introducing variability in turning behaviour. The pseudo code for each policy can be found in Appendix B.

The second policy, **Right-Wall Following**, attempts to follow the right-hand wall using a continuous correction system inspired by force dynamics. The robot maintains a desired distance from the wall by adjusting motor speeds based on deviations. If it drifts too far from the wall, the left motor slows down; if it gets too close, the right motor slows down instead. When a frontal obstacle is detected, the robot turns away from it using a smooth turn function, proportional to how close the obstacle is.

The third policy is a **Greedy policy**. It focuses on movement toward more open space. When no obstacles are in front, the robot adjusts wheel speeds based on the difference between the left and right distance readings: the wheel closer to an obstacle slows down, pushing the robot away from it. If an obstacle is detected ahead, the robot turns toward the side with more space, defaulting to the right in ambiguous cases. This policy aims to explore spacious areas and avoid confinement.

The fourth policy is just a mirrored version of the second policy, **Left-Wall Following**. It causes the robot to follow the left-hand wall using the force distance compensation system.

The fifth policy, **Random Navigation with Wall Avoidance and Randomness**, is a variant of Policy 1 but introduces a probability of executing random turns, even when such a manoeuvre is not strictly necessary. The robot continues forward when unobstructed, but unlike Policy 1, where random turns are disabled, Policy 5 allows occasional spontaneous changes.

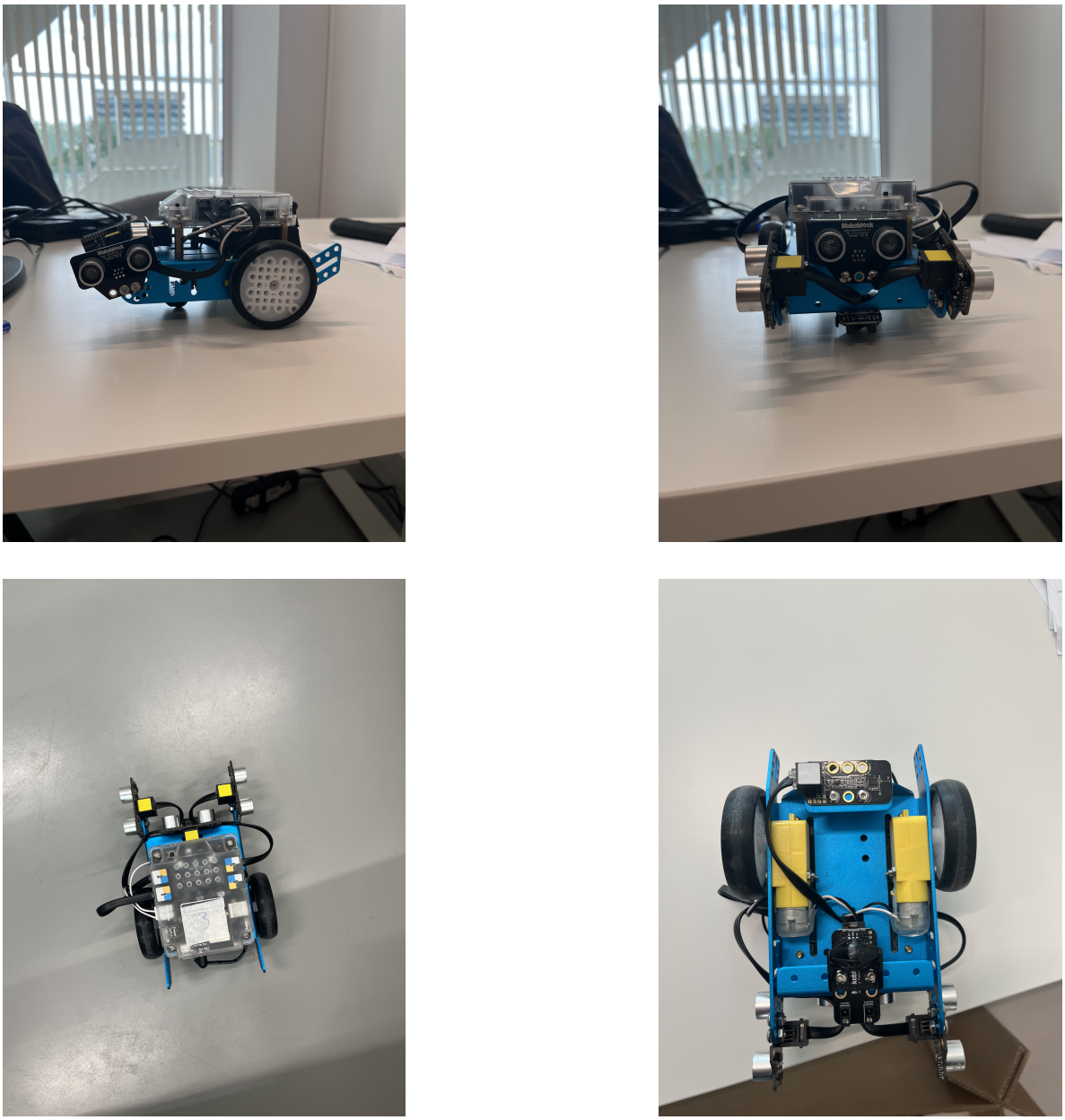


Figure 3: Final physical mBot architecture.

While the policy is being executed, the robot will take ”screenshots” X number of times every second. This screenshot is sent through Bluetooth Low Energy (BLE) protocol to a 2020 MacBook Pro. This was done by pairing each robot with its BLE address and subscribing to its notifications channel. Each sequence sent had to be of the specified length and start and end with { and end with } to avoid broken sequences. Lost messages occur with BLE, particularly as it breaks down every message to a maximum of 20 bytes. These screenshots contained the front, left and right distances captured by the ultrasonic sensor, the gyroscopic angular difference with regards to the previous screenshot, and the wheels' speed. In the end each datapoint looks as such: {”left”: X, ”front”: X, ”right”: X, ”gyro”: X, ”lwheel”: X, ”rwheel”: X, ”extra”: X}. The system was developed to store each robot in a separate file marked with the map name and the policy currently being used.

To collect as much information as possible, an attempt was also made to incorporate accelerometer data. This proved particularly challenging, as the mBot Arduino library does not expose raw accelerometer values. To overcome this, a custom demo library, *Verdejo*, was developed to access the sensor data, perform numerical integration of acceleration to estimate velocity, and subsequently integrate velocity to approximate displacement. However, this method yielded highly inaccurate results and was ultimately discarded to avoid introducing noise into the dataset. When researching the issue, it seems to be a common problem found in the robotics community when it comes to lower-end accelerometers.

For data processing, a preprocessing library was developed to transform the raw robot logs into usable input sequences for training. The library extracted temporal sequences of sensory and motor data, applied optional data augmentation to increase variability, and merged individual robot logs into a single unified dataset. The goal of preprocessing was to ensure that the resulting dataset contained diverse lengths, had enough data points, avoided excessive overlap, and was suitable for training sequence models. Each input sequence consists of sensor and motor values with normally distributed lengths, and follows the format: `Left: X Front: X Right: X Gyro: X LWheel: X RWheel: X`.

5.1.2 LLM Fine-Tuning & Inference

The model selected for fine-tuning in this thesis is LLaMA 3.1 8B by Meta, as introduced in Section 3.2. This choice is motivated by several factors. First, LLaMA 3.1 is an open-source model for academic researchers. Second, it is highly optimised for performance and memory efficiency, making it feasible to fine-tune using relatively modest computational resources. Third, the model is well-documented and supported by a broad ecosystem of tools and libraries, particularly through HuggingFace. These resources significantly streamline the development process: once the base architecture is implemented and the hyperparameters configured, datasets in various formats can be easily loaded and applied within a consistent training framework. Lastly, LLaMA 3.1 8B is particularly suitable for this research due to its low computational requirements and fast inference speed, enabling efficient near-real-time deployment on modest hardware.

For this thesis, fine-tuning was conducted using a Hugging Face's high-level Trainer API. The dataset was first loaded and validated using the *datasets* library, followed by *tokenisation* with *AutoTokenizer*, ensuring consistent handling of input length and padding. The model (architecture and weights) was instantiated via *AutoModelForCausalLM*, and its token embeddings were resized to accommodate the tokeniser changes. A loop was implemented to freeze lower transformer layers to reduce memory usage

and accelerate training. The *DataCollatorForLanguageModeling* was used for dynamic batch construction and efficient padding, while *EarlyStoppingCallback* monitored validation loss to terminate training once convergence stalled.

For the model's hyperparameter configuration, the tokenisation of the data to figure max length and padding, a custom-built computer with an Nvidia RTX 2060 and AMD Ryzen 5 5600X was used. For the model's fine-tuning, a server using an RTX 6000 Ada with 48 GB of VRAM was initially used. However, due to the process being computationally intensive, the fine-tuning was subsequently moved to a server with an H100 PCIe with 80 GB of VRAM. The storage memory also started at 80 GB; however, due to the model size and storing the latest two copies of the model after each epoch, the resources were increased to a total of 180 GB. Finally, for the model inference, depending on availability, the RTX 6000 Ada or A40 with 48 GB of VRAM were used.

5.2 Experimental Setup

In this section, we will cover the experiment setups for both the data collection and LLM training and inference. We will begin by covering the physical layout of the robot arenas and the configuration of the experiment. Then, we will cover the data configurations, training configurations and inference configurations in each of the trained models.

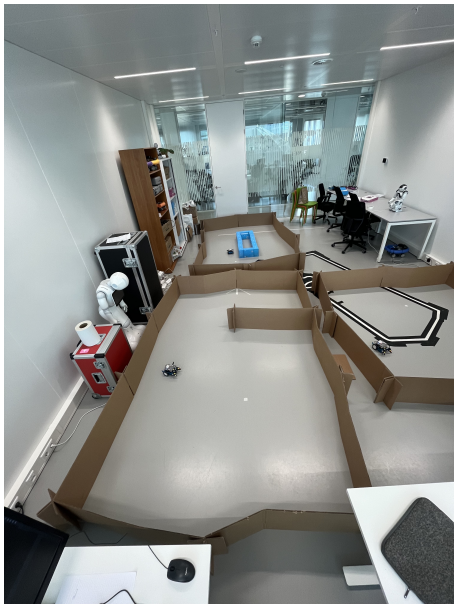
5.2.1 Data Collection

The five robot policies in Section 5.1.1 were placed in three different arenas or maps. Each of the arenas had different dimensions and obstacles. This is to ensure the data is diverse enough, the sensors read varying distances and the turns also have to adapt to the different conditions. The map designs are shown in Figure 4.

(a) Map A, 3x2 with an obstacle in the middle

(b) Map B, 2x2 and no obstacles

(c) Map C, 3x2 with a small explorable compartment in the top right corner



(d) Map Set-Up Overview

Figure 4: Maps for data collection with the mBot inside of them.

To ensure the safety of the robots and the map and to maintain realistic data without the robot approaching too close to the walls in normal scenarios, policies were configured to become alert when a wall is nearer than 40cm in the front and 15cm on the sides. By that distance, the robot would slowly start changing route in all policies. The closer the robot gets to the wall, the stronger the force to move away would be.

The robots were also configured to send sensor readings at 5 Hz (five times per second). This frequency

was selected as it provides a good balance between capturing fine-grained actions, such as turns, and ensuring sufficient variation between individual data points. It also allowed for the collection of a large and diverse dataset within a short period of time.

Each policy was executed across all three maps, resulting in five distinct policy–map combinations. Every policy was run for approximately 45 minutes per map, producing around $\approx 13{,}500$ data points per session. In total, this yielded $206{,}056$ data points. The 45-minute duration was chosen based on preliminary testing, which revealed a noticeable decline in robot speed due to battery depletion occurring between 55 minutes and 1 hour 10 minutes of continuous operation. Thus, to keep the variables stable, 5 readings per second for 45 minutes produced enough results. Thus, to keep the experimental conditions stable, and given that collecting five readings per second over a 45-minute interval already yielded a sufficient volume of data, each policy was limited to this duration.

With these datasets, a total of 5 unified datasets were generated with the developed data-processing library. The approach for the generation and training was a top-down approach, starting from the most processed to the simpler and more raw. These datasets are described in Table 1.

Dataset	**Description**
Dataset 1	Discretised data: distances and angles binned into ranges, wheel speed differences mapped to actions (`left`, `slight left`, etc.). Includes special tokens (<STATE>, <ACTION>, <CONTEXT>, <PREDICTION>, <ACTIONS>). Context length: 5–15 states; prediction length: 1–5 states.
Dataset 2	Raw continuous data (not discretised), without special tokens. Sequence lengths range from 5 to 70. No data augmentation applied, resulting in fewer training examples.
Dataset 3	Same as Dataset 2, but with data augmentation as described in Section 5.1.1. This increases data diversity and example count.
Dataset 4	Similar to Dataset 3, but with reduced sequence length: 1 to 25 states. Logs from policy 1 were removed due to a lack of variety in wheel speed and due to its similarity to policy 5. Helps the model focus on short-term dependencies and generalisation.
Dataset 5	Identical to Dataset 4, but adds newline characters between entries to improve readability and parsing for certain models.

Table 1: Overview of the five generated datasets.

5.2.2 LLM Fine-Tuning & Inference

The fine-tuning process was configured with attention to performance, to ensure it will not run out of VRAM. The batch size was set to 14 per device, with gradients accumulated over 8 steps. This results in an effective batch size of 112, allowing the model to benefit from larger batch dynamics while fitting

within GPU memory constraints. The learning rate was set to 2×10^{-5}, with 100 warm-up steps to allow the optimiser to stabilise before full training starts.

A weight decay of 0.01 was applied for regularisation, helping prevent overfitting during training. Mixed-precision training was enabled using bfloat16, which balances numerical stability with improved memory efficiency supported by the mentioned GPUs in Section 5.1.2. Gradient checkpointing was also activated to reduce memory usage further by recomputing intermediate activations during backpropagation.

To ensure robust evaluation, training was conducted over 5 epochs, with validation and model checkpointing triggered at the end of each epoch. The eval_loss metric was used to monitor model performance, and the best-performing model was retained via load_best_model_at_end=True. Additionally, to avoid overfitting and unnecessary computation, an early stopping criterion was set with a patience of 2 evaluation steps.

Logging was configured to occur every 50 steps, with all logs directed to TensorBoard. Only the two most recent checkpoints were retained during training to manage disk usage. Padding during batch construction was aligned to multiples of 8 tokens to ensure optimal performance on GPU hardware. Four worker threads were used to parallelise data loading.

The LLaMA 3.1 8B was then fine-tuned based on this configuration for each of the five datasets and stored as different instances. In addition, in the configuration of the first model, the special tokens were also added. In the training of the fifth model, the learning rate is also halved after every epoch. For the last 2 models, the datasets based of the first policy were removed due to their lack of movement variation in an attempt to prevent imbalance.

For inference, the fine-tuned model and tokenizer were reloaded using AutoTokenizer and AutoModelForCausalLM from the HuggingFace Transformers library. The model was used in evaluation mode to prevent any updates to its parameters. Text prompts were encoded using the tokeniser and passed to the model via the generate method, which handles autoregressive generation of new tokens. During generation, do_sample=True was enabled to allow sampling-based decoding rather than greedy selection. This enabled the use of decoding parameters such as temperature, top_k, and top_p to control randomness and diversity in the model's responses.

Throughout experimentation, multiple inference configurations were tested to balance accuracy and creativity. Due to resource constraints, hyperparameter tuning was only conducted during the initial two model runs. The most stable configuration from those preliminary experiments was then used for all subsequent models without further tuning. The most stable configuration had a temperature value of 0.7, more deterministic. Likewise, top_k was tried for values between 10 and 30 to limit sampling to the top-k most probable next tokens, while top_p (nucleus sampling) values between 0.85 and 0.95 were explored to dynamically sample from the smallest set of tokens whose cumulative probability exceeded p. In the end, top-k with a k of 15 was settled.

5.3 Measures & Evaluation

This section outlines the evaluation methods used to assess the three hypotheses stated in Section 4.1. Given the complex nature of each hypothesis, a combination of quantitative and qualitative measures has been employed. These measures are tailored to evaluate not only the statistical performance of the model but also the interpretability and plausibility of its predictions in a spatial reasoning context.

For the first hypothesis, "Data can be collected from mBots in a way that is consistent, varied and structured, preserving meaningful temporal and spatial features." the repetition rate by Bertoldi et al. [BCF13] is calculated to ensure the data is varied:

$$RR = \left(\prod_{n=1}^{4} \frac{\sum_S (V(n) - V(n,1))}{\sum_S V(n)} \right)^{\frac{1}{4}}$$

Where: n is the n-gram length, ranging from 1 to 4. S is a sliding window over the text (e.g., 1000-word segments). $V(n)$ is the number of unique n-gram types within window S. $V(n,1)$ is the number of n-gram types that occur exactly once (singletons) in S. The numerator, $V(n) - V(n,1)$, gives the number of repeated n-grams in S. The denominator, $V(n)$, normalises by the total number of n-grams. Here we calculate the repetition rate by segments.

To ensure temporal and sensor coherence, a few trajectories are randomly chosen, plotted and analysed to ensure the robot moves coherently. In other words, there are no sudden leaps and wheel differences, and gyro matches with the sensor changes. These trajectories will also be compared between maps and policies to ensure movement and sensor variety.

For the second hypothesis, "LLMs are able to process spatial data as input and learn from it during training." the loss will be calculated with the cross-entropy formula, the default calculation when training the LLaMA in causal language modelling with HuggingFace. [Fac] Given a token sequence of length T, the model predicts a probability distribution $\hat{p}_t$ over the vocabulary at each time step t, aiming to maximize the likelihood of the correct next token y_t. The loss function is defined as:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T_i} \log \hat{p}_t^{(i)}(y_t^{(i)})$$

where N is the batch size, T_i is the sequence length for sample i, and $\hat{p}_t^{(i)}(y_t^{(i)})$ is the probability assigned to the correct token y_t at time t. Padding positions and special tokens are ignored via a loss mask.

For the third hypothesis, "Given a sequence of spatial data-points, the LLM is able to predict future plausible states." To evaluate whether the fine-tuned LLM is capable of predicting future plausible spatial states from a sequence of spatial data points, we employ both quantitative and qualitative measures. The main quantitative evaluation consists of measuring the absolute prediction error for each distance sensor and the gyroscope, averaged over a held-out test set of N examples.

Let $\hat{d}_i^{(s)}$ be the predicted distance at time step i from sensor $s \in \{\text{front}, \text{left}, \text{right}\}$, $d_i^{(s)}$ be the corresponding ground-truth value, $\hat{\theta}_i$ be the predicted angular change from the gyroscope, θ_i be the ground-truth

gyroscopic change. Then, we compute the Mean Absolute Error (MAE) for each sensor and the gyroscope as:

$$\text{MAE}_s = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{d}_i^{(s)} - d_i^{(s)}\right|, \qquad \text{MAE}_\theta = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{\theta}_i - \theta_i\right|$$

To complement the quantitative analysis, we perform a manual inspection of prediction sequences that exhibit significant error. Specifically, we filter cases where any of the distance sensor errors exceeds ±10 cm:

$$\exists s \in \{\text{front}, \text{left}, \text{right}\} \text{ such that } |\hat{d}_i^{(s)} - d_i^{(s)}| > 10$$

These cases are then visually inspected to assess whether the predicted future state sequence remains physically and spatially plausible. We analyse trends such as approach to walls, cornering behaviour, or continuous straight movement, checking whether the changes in sensor values align with expected robot behaviour.

To further validate spatial coherence, we measure whether the wheel speed difference at time step t correctly predicts the direction of rotation measured by the gyroscope at time step $t+1$. Let: $w_d^{(t)} = w_{\text{left}}^{(t)} - w_{\text{right}}^{(t)}$ be the wheel speed difference at time t, $\theta^{(t+1)}$ be the ground-truth angular change at the next time step, $\text{sign}(x)$ be the sign function:

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

We calculate the gyro sign accuracy as:

$$\text{Accuracy}_{\text{gyro}} = \frac{1}{N}\sum_{t=1}^{N} \mathbb{1}_{\left[\text{sign}(w_d^{(t)}) = \text{sign}(\theta^{(t+1)})\right]}$$

This accuracy captures whether the predicted wheel speed (expressed by motor asymmetry) results in a change in the gyro expected direction, ensuring that action predictions align with expected rotational behaviour.

For the fourth hypothesis, "Given a sequence of spatial data-points, the LLM is able to predict future plausible actions." the future prediction actions (wheel speed difference) are analysed both quantitatively, by computing the MAE, and qualitatively, to assess their plausibility. The analysis focuses on common navigational scenarios where an agent is expected to react in a consistent manner, such as approaching corners, encountering frontal walls, or lateral walls. For each case, we assess whether the predicted action aligns with the typical or expected response that the real robots would exhibit in that situation.

6 Results

In this section, we present the experimental findings organised by the four research hypotheses (Section 4.1). For each hypothesis, the relevant quantitative and qualitative results are reported, stating whether the findings support the hypothesis.

6.1 Hypothesis 1

For H1, as described in Section 5.3, the Repetition Rate will be plotted, represented in bar graphs and a qualitative analysis of the temporal and sensor coherence.

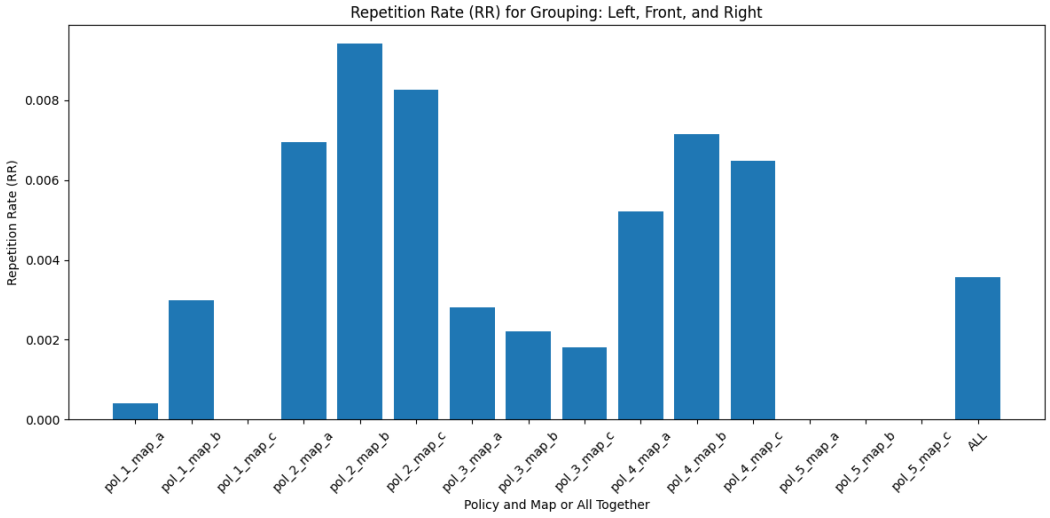


Figure 5: Repetition Rate where each 1-gram is represented only by the left, front and right ultrasonic sensor readings.

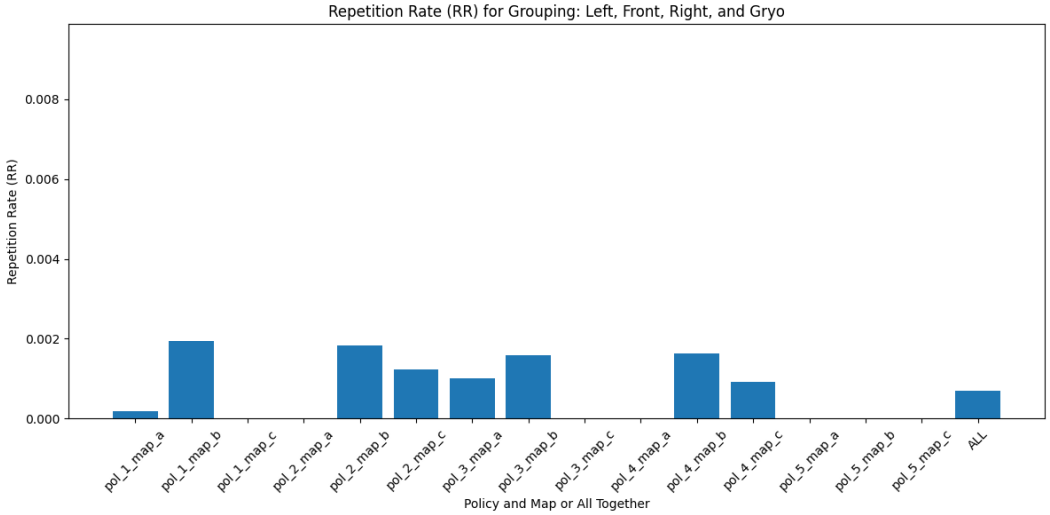


Figure 6: Repetition Rate where each 1-gram is represented only by the left, front, and right ultrasonic sensor and gyro readings.

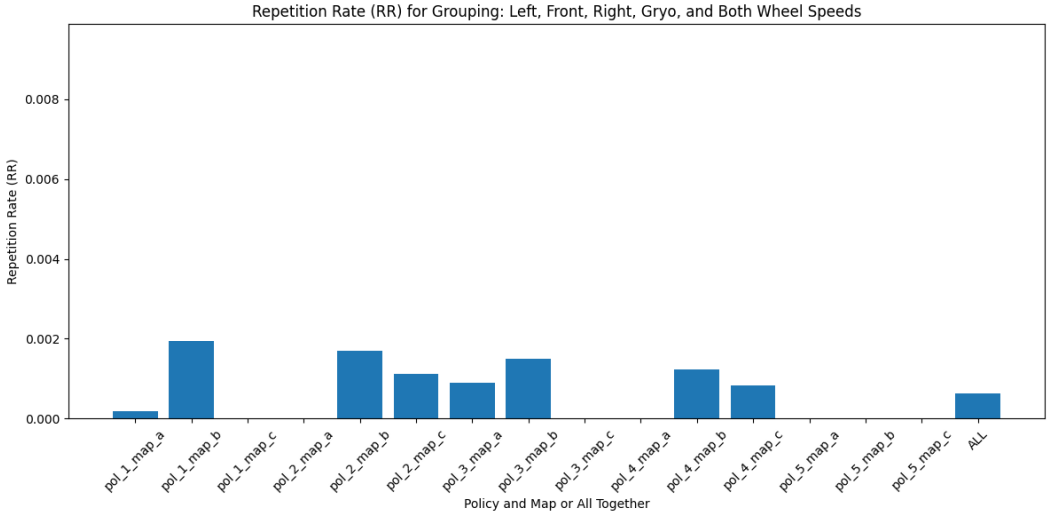


Figure 7: Repetition Rate where each 1-gram is represented by the left, front, and right ultrasonic sensor, gyro readings and wheel speeds.

Across all three graphs, the Repetition Rate remains consistently low. In the first graph, all values are below 0.01, while in the second and third graphs, where additional input is included, the Repetition Rate drops further, remaining below 0.002.

To qualitatively assess the consistency and interpretability of the collected data, multiple samples were manually reviewed multiple samples from each policy. For each policy, we selected a few samples where the robot begins by going straight and ends up turning to avoid obstacles or walls. These behaviours are illustrated in Figures 8 9 10 11 12.

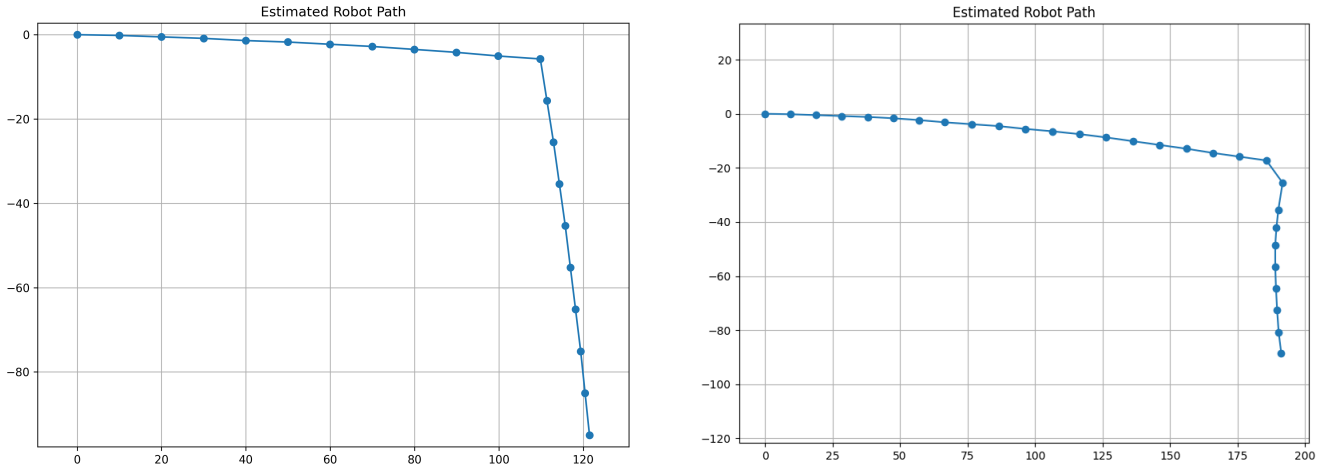


Figure 8: Sampled robot trajectories under Policy 1, illustrating straight movement and turning behaviour. The graph on the left represents a trajectory in map A, and the figure on the right in map C.

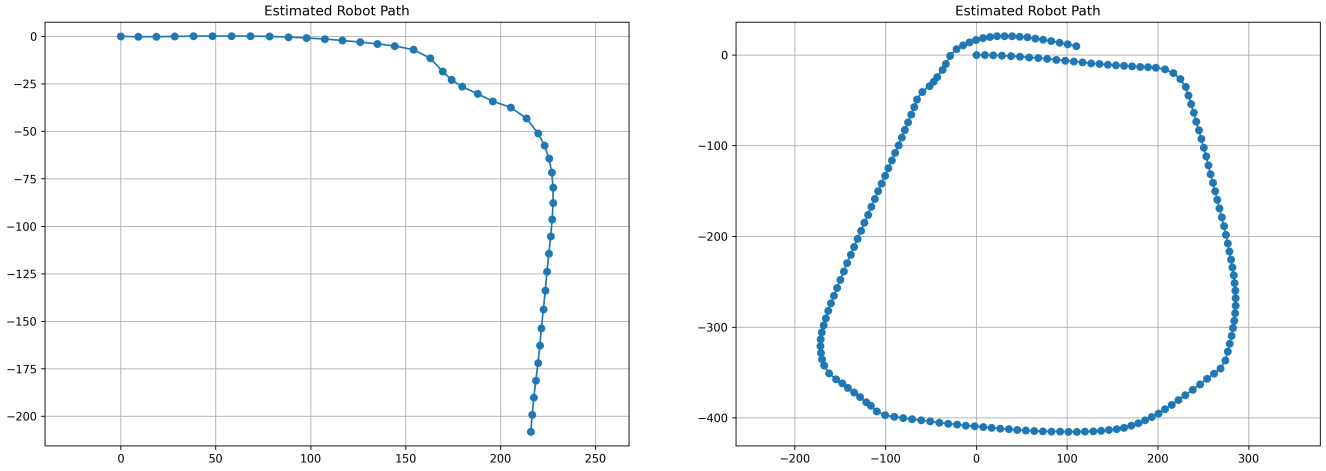


Figure 9: Sampled robot trajectories under Policy 2, the left illustrating straight movement and turning behaviour, while the right illustrates a leap around the whole map. The graph in the left represents a trajectory in map A, and the figure on the right in map B.

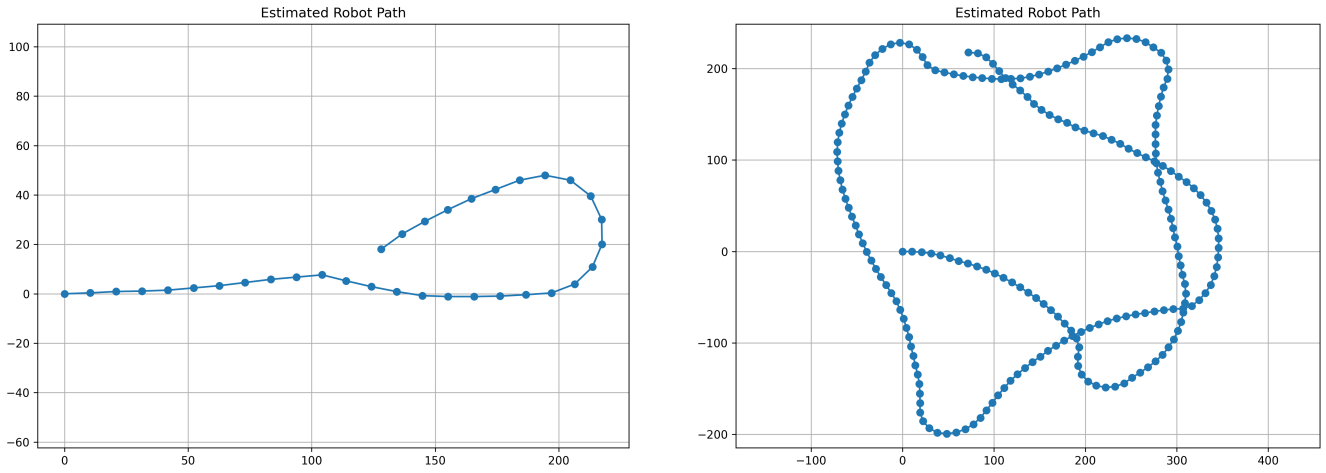


Figure 10: Sampled robot trajectories under Policy 3, the left illustrating a full turn-around while the right illustrates a trajectory where on both sides there were tight walls, and turns once the wall on the left is left behind. The graph on the left represents a trajectory in map B, and the figure on the right in map C.

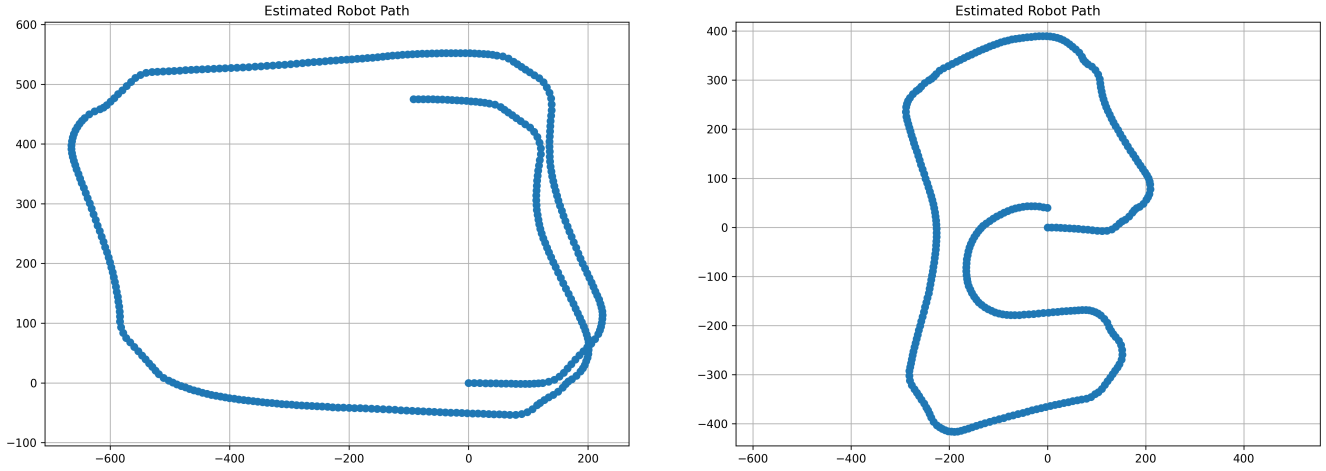


Figure 11: Sampled robot trajectories under Policy 4, the left illustrating a full leap where the object was in the middle, the right illustrating a leap where there is a partial wall towards the down-left corner, which the robot moves around. The graph on the left represents a trajectory in map B, and the figure on the right in map C.

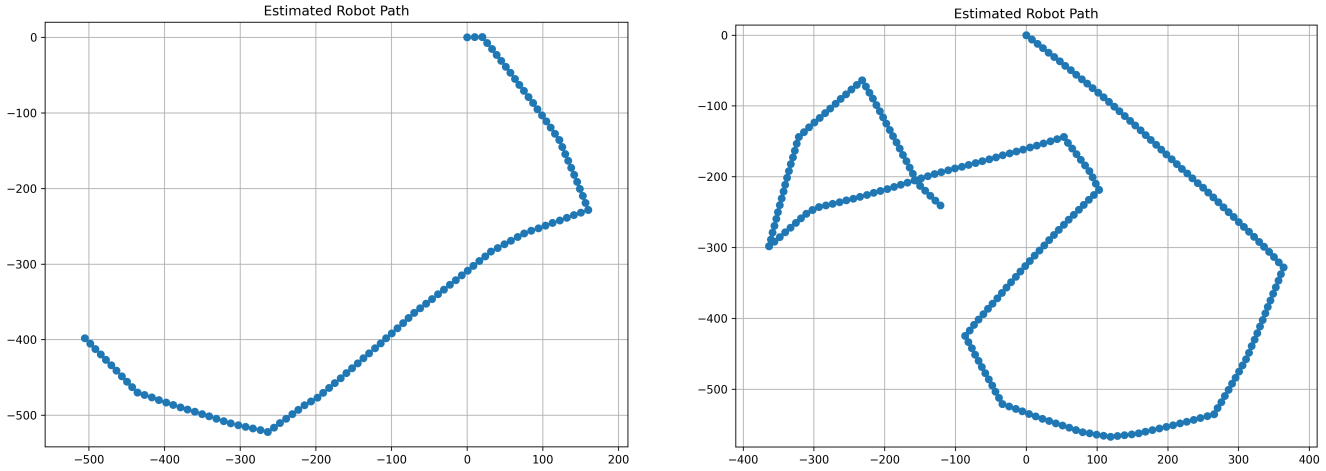


Figure 12: Sampled robot trajectories under Policy 4, the left illustrating a full leap where the object was in the middle, the right illustrating a leap where there is a partial wall towards the down-left corner, which the robot moves around. The graph on the left represents a trajectory in map A, and the figure on the right in map C.

The trajectories shown in the figures exhibit plausible movement patterns consistent with the expected behaviour of the robot in each environment. Straight-line motion and turning behaviours are clearly distinguishable and correspond with the layout of the respective maps. The Repetition Rate calculated across all files remained consistently low (all values under 0.009), indicating a high degree of variation in the recorded sequences. These results indicate that the collected data maintains spatial and temporal structure and variability, demonstrating Hypothesis 1.

6.2 Hypothesis 2

For H2, as described in Section 5.3, we assess whether the model was able to learn from the spatial data, we tracked its training and validation loss over time. it is logged every 50 steps for the training loss and every epoch for the validation loss. The resulting learning curve are shown in Figures 13 and 17

Figure 13: Training loss of the LLM over 5 epochs logged every 50 steps.

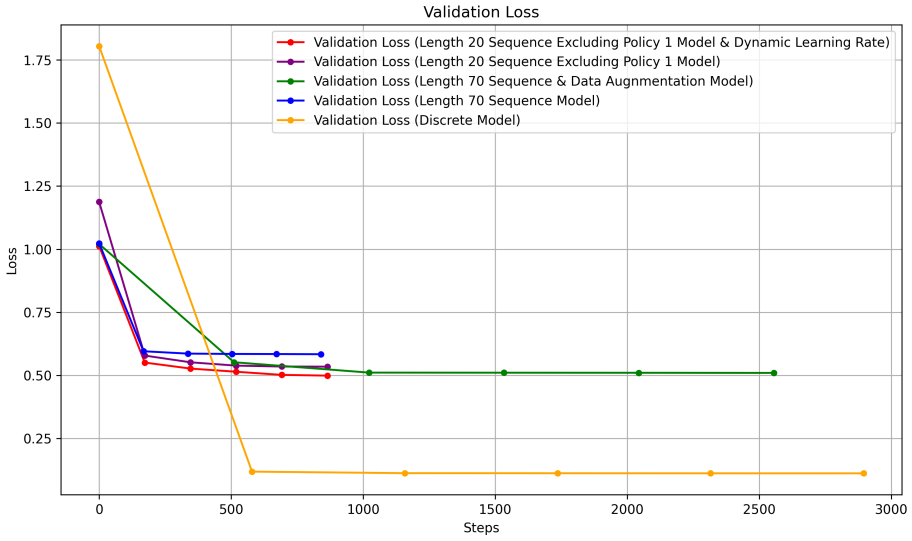


Figure 14: Validation loss of the LLM over 5 epochs logged every epoch.

The training loss decreased steadily over the course of the first training epoch, and kept slowly decreasing after as it converged, as shown in Figure 13 and Figure 17. These curves demonstrate that the model was able to learn from the spatial data and reach a stable training state over time, thus supporting H2.

6.3 Hypothesis 3

For H3, as described in Section 5.3, we assess whether the model is able to predict plausible spatial data. We calculated the MAE of each sensor and the gyro, and we also analysed the sign accuracy of the gyro, since it depends on the previous wheel speed. Lastly, we qualitatively analysed the predictions with high error to identify potential patterns.

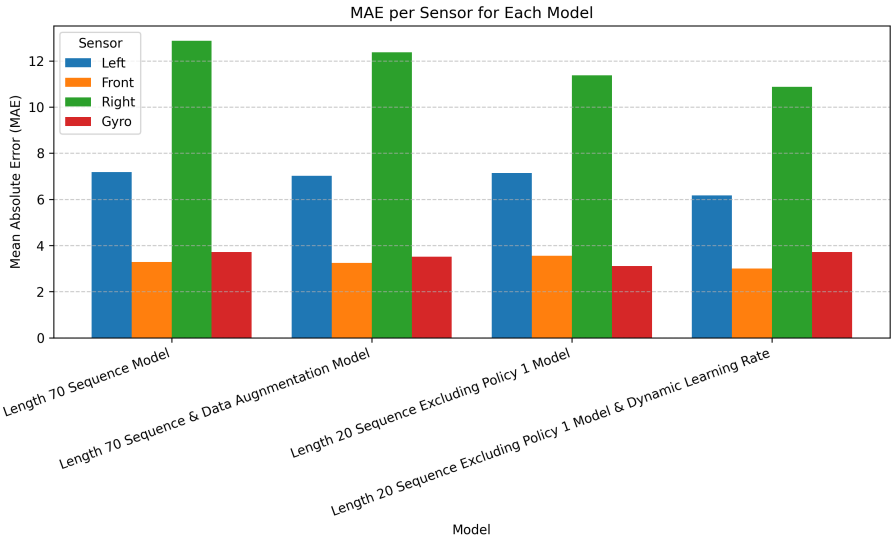


Figure 15: Mean Absolute Error per Sensor from the First State Predicted by the Fine-tuned LLM. The Context Length Ranged from 1 to 24.

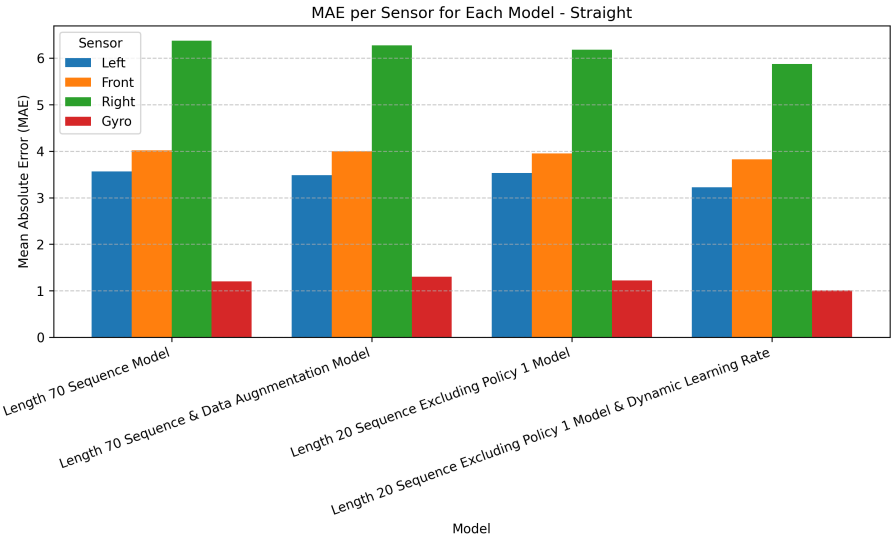


Figure 16: Mean Absolute Error per Sensor from the First State Predicted by the Fine-tuned LLM. The Context Length Ranged from 1 to 24. Only data points with ground truths with a wheel speed difference smaller than 15 are considered.

	Dynamic Learning Rate	Length 20	Length 70 + Augmentation	Length 70
Gyro Sign Accuracy	0.702	0.638	0.638	0.638

Table 2: Gyro sign accuracy per model.

When analysing the predictions with the highest errors (> 10cm as mentioned in Section 5.3), we observed that the model incorrectly repeats the last state even when the robot is too close (less than 10 cm) and never reaches under 6 cm from the wall. However, when the robot is indeed expected to go straight, the prediction error for the front, left, and right sensors is typically under 6 cm. All models struggled with turns and failed to produce any correct predictions in turning scenarios.

Hence, while the models are able to predict plausible states during forward motion, they are unable to generate correct states following turns.

6.4 Hypothesis 4

For H4, as described in Section 5.3, we assess whether the model is able to predict plausible future movements. We calculated the MAE of each wheel speed and qualitatively analysed the behaviour of the model in different scenarios.

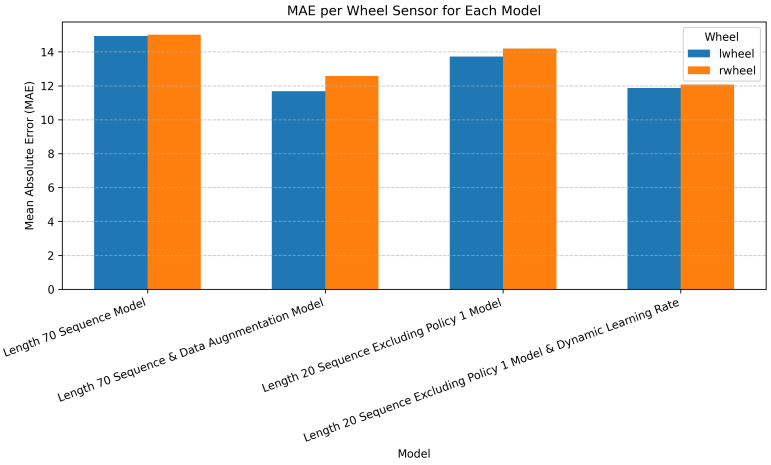


Figure 17: Mean Absolute Error per wheel speed from the First State Predicted by the Fine-tuned LLM.

When analysing how the model behaves in different scenarios, all models succeeded when the action of going forward was possible. However, when the models were put in more complex scenarios, such as an object in front, all models failed and continued the wheel speed as if going forward. The only exception was *Length 20 Sequence Excluding Policy 1 Model & Dynamic Learning Rate*, where it once attempted to avoid the wall, predicting the trajectory shown in Figure 18.

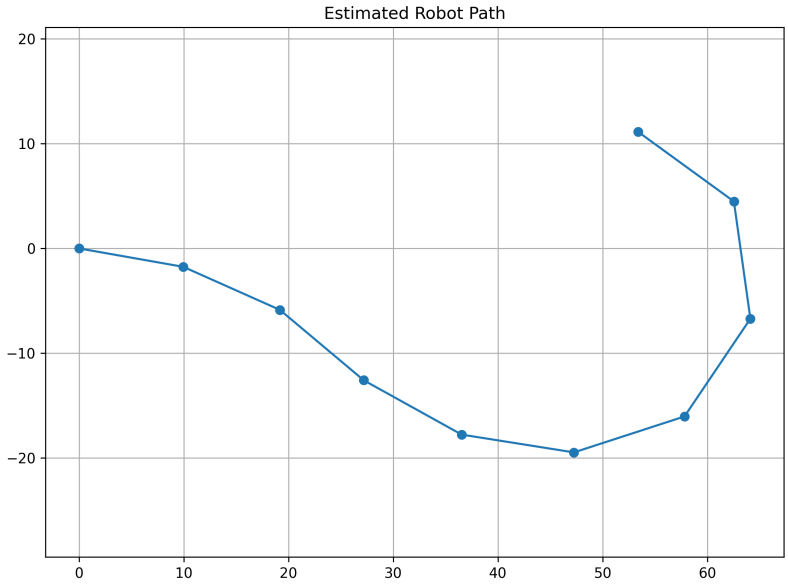


Figure 18: Predicted path when the model was approaching a front wall and also having a reduced amount of space on the left.

When tested beyond the test prompts, this model occasionally produced plausible future paths, demonstrating some generalisation ability. Nevertheless, similar to the others, it often failed to escape close proximity to walls.

7 Discussion

The following section reflects on the strengths and limitations observed during evaluation, examining how well the datasets and models met the original hypotheses. We discuss the implications of the results, particularly in relation to spatial prediction accuracy and action understanding.

7.1 Reflection on Results

The results presented in the previous sections demonstrate that with relatively simple robots and policies, varied and meaningful datasets can be constructed, and that fine-tuning a large language model (LLM) on spatial perception data enables it to produce partially plausible predictions for both future states and actions. However, its performance varies significantly across types of predictions and scenarios.

The analysis of sensor prediction errors (Hypothesis 3) showed that the model could predict forward movement with reasonable accuracy, with the mean absolute error for the front, left, and right distance sensors typically below 10 cm. Notably, the right sensor exhibited the highest error across all models, due to higher changes in that sensor during the turns in the turns that it cannot predict. The gyro predictions, though relatively accurate in magnitude, presumably due to their small values (except during turns) showed inconsistencies in sign, indicating difficulty in understanding rotational direction that is dependent on the previous speed wheel difference. The sign accuracy analysis confirmed this, with only one model barely exceeding 70% sign consistency. None of the models were able to predict the spatial changes during turns and would just repeat the last state, indicating their ability to predict it is not possible to go any further, but not being able to predict how.

In the case of Hypothesis 4, the LLM showed limited ability to predict realistic wheel speeds (interpreted as actions). While it could reproduce forward motion consistently when the scene permitted it, it often failed in more complex navigation tasks, particularly when encountering frontal obstacles. All models defaulted to forward movement regardless of the spatial constraints. The only exception was the *Length 20 Sequence Excluding Policy 1 Model & Dynamic Learning Rate*, which in some cases attempted a trajectory adjustment when approaching a wall. However, this was only based on the gyro, and the wheel speed difference would be minimal, not realistic for the degrees it predicted to turn. This further shows that it did not establish a connection between the previous state's wheel speeds and the next state's gyro difference.

At hypotheses 3 and 4, the differences between models highlight the influence of training decisions. Models trained with shorter input sequences or data augmentation showed marginally better generalisation, with the best model being composed of short sequences and a dynamic learning rate.

Overall, while the LLM developed a basic notion of spatial continuity and inertia (e.g., moving forward when unobstructed), it struggled with discontinuities and critical decision points, such as when being too close to a wall. This suggests that while transformer-based LLMs are capable of capturing low-level patterns in sequential spatial data.

7.2 Future Work

Future work could explore more temperature, top-k and top-p configurations as well as different variations to structure the data. In particular, restructuring the data to better capture the relationship between the previous state's wheel speeds and the resulting gyro value may help the model learn directional implications more effectively. A more balanced dataset containing more of the special cases would also be an option, to analyse whether the imbalance in the number of datapoints going straight and turning has an impact on the probability, making going straight more likely. Additionally, different architectures could be tested.

8 Conclusion

This thesis explored the ability of a fine-tuned large language model (LLM) to perform spatial state prediction and generate plausible future actions based on context sequences of spatial data. Through four hypotheses, we evaluated the capacity to generate good datasets from basic robots, we evaluated the model's capacity to predict the next state, understand directional movement, and propose actions in a robot navigation context.

The results show that the LLM is able to capture basic spatial continuity, producing low-error predictions when forward movement is appropriate. Sensor predictions, particularly for the front, right and left readings, were accurate to within a few centimetres in most basic cases. However, the model struggled with more complex spatial reasoning tasks. While realising that going forward, when a wall was in front was not possible, it failed to adjust its behaviour when facing obstacles and often defaulted to repeating the last state over and over. In terms of action prediction, the model rarely generated turns or stopping behaviours, with only the last configuration occasionally producing such responses.

Despite these limitations, the model demonstrated partial spatial understanding and some generalisation capability, particularly in smooth, forward-motion scenarios. These results indicate that LLMs can learn basic spatial dynamics from unstructured data, but lack the reasoning mechanisms needed for more robust autonomous behaviour.

References

[AHS85] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for Boltzmann Machines. *Cognitive Science*, 1985.

[AI24] Meta AI. Meta LLaMA 3 system card, 2024. https://ai.meta.com/blog/meta-llama-3/.

[BCF13] Nicola Bertoldi, Mauro Cettolo, and Marcello Federico. Repetition rate: A new measure for MT Evaluation. In *Proceedings of the 14th Machine Translation Summit*. Asia-Pacific Association for Machine Translation, 2013.

[DXS+23] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. PaLM-E: An Embodied Multimodal Language Model. *arXiv preprint*, 2023.

[Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 1990.

[Fac] Hugging Face. modeling_llama.py – Hugging Face Transformers github repository. https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modelingllama.py.

[FLD18] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018.

[GDJt24] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, and *et al.* The LLaMA 3 herd of models. *arXiv preprint3*, 2024.

[GSS+24] Pranav Guruprasad, Harshvardhan Sikka, Jaewoo Song, Yangyue Wang, and Paul Pu Liang. Benchmarking Vision, Language, & Action Models on Robotic Learning Tasks. *arXiv preprint*, 2024.

[H+24] Wenlong Huang et al. Solving robotics problems in zero-shot with Vision-Language Models. *arXiv preprint*, 2024.

[HBD+20] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of Neural Text Degeneration. In *International Conference on Learning Representations (ICLR)*, 2020.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[Inv13] InvenSense (TDK). Mpu-6000/mpu-6050 product specification. Technical datasheet, 2013. Combined 3-axis accelerometer and gyroscope with onboard DMP.

[KKC+24] Youngwoo Kim, Donghyeon Kim, Jaehyeon Choi, et al. A survey on integration of Large Language Models with Intelligent Robots. *Intelligent Service Robotics*, 2024.

[KR18] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018.

[LAK+] Zhen Lu, Imran Afridi, Hong Jin Kang, Ivan Ruchkin, and Xi Zheng. Surveying Neuro-Symbolic Approaches for reliable Artificial Intelligence of Things. *Journal of Reliable Intelligent Environments*.

[LS+23] Corey Lynch, Pierre Sermanet, et al. Language-conditioned imitation learning for Robot Manipulation Tasks. *arXiv preprint*, 2023.

[Mak] Makeblock. mbot STEM Kit. Makeblock product page. Specifications and features for the mBot robot.

[Mak24] Makeblock. mbot: First Robot Kit for coding and STEM Learning. Makeblock product page, 2024.

[MT21] Akeem Whitehead Mubina Toa. Ultrasonic sensing: A practical guide to Measurement and Applications. Texas Instruments, 2021.

[OWJ+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, and Alex et al. Ray. Training language models to follow instructions with Human Feedback. *Advances in Neural Information Processing Systems*, 2022.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by Generative Pre-Training. 2018. OpenAI Blog.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

[Sha19] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint*, 2019.

[Sha20] Noam Shazeer. GLU variants improve Transformer. *arXiv preprint*, 2020.

[SHB16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with Subword Units. 2016.

[SLP+21] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv preprint*, 2021.

[Tam24] Milan Tamang. Build your own LLaMA 3 architecture from scratch using PyTorch. Towards AI Pub, 2024.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint*, 2023.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[XWL+24] Haoyi Xiong, Zhiyuan Wang, Xuhong Li, Jiang Bian, Zeke Xie, Shahid Mumtaz, Anwer Al-Dulaimi, and Laura E. Barnes. Converging paradigms: The synergy of Symbolic and Connectionist AI in LLM-Empowered Autonomous Agents. *arXiv preprint*, 2024.

[ZT19] Biao Zhang and Ivan Titov. Root mean square layer normalization. *arXiv preprint*, 2019.

A Indepth LLM Background

The introduction of the transformer architecture by Vaswani et al [VSP+17] opened a new chapter in artificial intelligence. Up to that point, sequence processing and prediction where done with Recurrent Neural Networks (RNNs) [Elm90] as well as Long-Short Term Memory networks (LSTMs) [HS97]. However, these architectures struggled when it came to processing sequences that were not relatively short. Vanishing gradients limited the length of these sequences, as text became diluted and non-immediate dependencies would not be captured. Moreover, the sequential structure of these models prevented computational parallelisation, as each hidden state depends on the previous one. This created a bottleneck problem and required high computational power. This also limited the length of sequences as well as the size of datasets. Transformers were a solution to all these problems, [VSP+17] as they could capture long-range dependencies and allowed parallelisation. As the solution to these limitations, they became the foundation for all current models.

Before examining the architecture of transformers, it is fundamental to understand how text sequences are converted into numerical representations that can be fed to these models. This process is called **tokenisation**, in which the text is broken into discrete units called **tokens**, which are the atomic units that will be processed. Tokens can represent various units, from individual characters to parts of words to words to phrases. Most modern LLMs apply subword tokenisation methods such as Byte Pair Encoding [SHB16] or SentencePiece [KR18]. These divide the word into several subparts, allowing compositional learning (understanding words it has never seen before) as well as being robust against typographical errors. Each token is then mapped to a unique integer identifier and embedded into a high-dimensional vector. These vectors will then be used as the input for the transformers and capture semantic relations which is essential for the attention mechanisms.

The most important novelty in the transformer models is the **self-attention mechanism**. [VSP+17] This mechanism enables all tokens to dynamically attend to all other tokens regardless of the distance from one another. It is done in a single sequence, where each position attends to all positions in the same sequence (including itself). This solves the issue with vanishing gradients in long sequences, as earlier tokens will still attend to later ones. The self-attention mechanism will convert each token embedding into three representations:

- **Query (Q)**: It encodes what kind of information this embedding requires from other tokens in the sequence.
- **Key (K)**: It encodes the type of information this token offers to others.
- **Value (V)**: It encodes the actual information the token offers.

Then, an attention score (often represented as *(e)* is calculated.This attention score describes how compatible or relevant two tokens are to one another. It is computed as the compatibility of one's query with another key. Thus, for each query vector *(q)*, we calculate the compatibility of its query with all keys *(k)* from all other tokens using the dot product.

$$e_{ij} = q_i \cdot k_j$$

This score is later converted into a probability through the softmax function. This solves the parallelisation problem, as any attention scores can be computed simultaneously with others. Along with this, the

simplicity of the calculations and their scalable complexity $\Theta(n^2)$ allows for relatively low computational requirements.

This parallelisation also allows for **multi-head attention** [VSP+17], where we can learn different matrices for each head to compute different attention scores, so that each focuses on different relations amongst the tokens at the same time. For example, one head can focus on syntactic dependencies while another captures semantic similarities. This is done by learning different projection matrices *(W)* for each of the three representations.

$$W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{(\text{embedding dimension} \times \text{key dimension})} \quad \text{for } i = 1, \ldots, h$$

Then we calculate the attention scores for each head independently, and lastly we concatenate all the scores and project them back to the original model dimensions.

This parallelisation also prevents the model from understanding or considering the order of the tokens. To deal with this, **positional encodings** [VSP+17] are added to the input embeddings, indicating the position of the token with respect to the other tokens in the sequence.

The basic building block of all LLMs and their predecessors is **transformer blocks**. Each transformer block is composed of two elements. First, a multi-head self-attention layer, followed by a **Position-wise Feed-Forward Network (FNN)** [VSP+17]. This FNN is a simple two-layer non-linear Neural Network (NN) which helps the model understand representations beyond what self-attention captures. After each element, the input is added to the just calculated output, preventing vanishing, and is subsequently normalised to stabilise training. Transformer models usually stack several transformer blocks on top of one another.

Training these stacked transformer architectures requires optimisation of key hyperparameters that control the learning process. The **learning rate** controls the step size during gradient descent optimisation, in other words, it determines how much the model parameters are updated at each training iteration, with too high values causing training instability and too low values leading to slow convergence. Additionally, **weight decay** acts as a regularisation technique that penalises large parameter values by adding a penalty term to the loss function, helping prevent overfitting and encouraging the model to learn more generalizable representations.

Modern-day LLMs work with **decoder-only architectures** [RNSS18, RWC+19]. These models consist of a stack of transformer blocks that use causal **self-attention**, also known as causal masking [VSP+17]. Causal masking ensures that each token in a sequence can only ”attend to” (i.e., use information from) previous tokens, not future ones. This is essential because LLMs are trained in an autoregressive manner, they learn to predict the next token in a sequence based only on the tokens that come before it. By preventing the model from seeing future tokens during training, causal masking ensures that the learning process mirrors the real-world use case of generating text one token at a time, avoiding any information leakage that would make the task artificially easier.

Once trained, these models generate text by iteratively predicting the next token, feeding that prediction back in as input for the next step. Each prediction step begins with the model computing a probability distribution over the entire vocabulary using the output of the final transformer layer. However, directly

sampling from this raw distribution can lead to incoherent or uninformative outputs, especially since it might include many low-probability words. [HBD+20] To guide generation, sampling strategies are applied to hopefully make the output more meaningful.

One common method is **top-k** sampling [FLD18], which restricts the sampling pool to the k most probable next tokens, eliminating the low-probability option trail. Another is **top-p** sampling [HBD+20], which chooses the smallest set of tokens whose cumulative probability exceeds a threshold p. Additionally, **temperature scaling** adjusts the sharpness of the probability distribution before sampling: a lower temperature (<1.0) sharpens the distribution, favoring more likely tokens, while a higher temperature (>1.0) flattens it, promoting more diverse and unpredictable outputs [HBD+20, AHS85].

B Robot Policies Pseudocode

Algorithm 1: Probabilistic Exploration with Reactive Obstacle Avoidance

```
main loop
Read leftDistance, frontDistance, rightDistance from ultrasonic sensors
canLeft ← leftDistance ≥ sideThreshold
canFront ← frontDistance ≥ frontThreshold
canRight ← rightDistance ≥ sideThreshold
if not canFront then
    goLorR(canLeft, canRight, leftDistance, rightDistance, frontDanger = true)
else
    if (leftDistance ≤ sideDanger or rightDistance ≤ sideDanger) and (canLeft or canRight) then
        goLorR(canLeft, canRight, leftDistance, rightDistance, frontDanger = false)
    else
        if (canLeft or canRight) and epsilonBoolean() then
            goLorR(canLeft, canRight, leftDistance, rightDistance, frontDanger = false)
        else
            Move forward at fixed speed
        end
    end
end

goLorR(canLeft, canRight, leftDistance, rightDistance, frontDanger) diff ← —leftDistance −
rightDistance—
if canLeft and canRight then
    if epsilonBoolean() and not frontDanger then
        Turn left randomly
    else
        if diff ≥ 30 then
            if leftDistance ¿ rightDistance then
                Turn left randomly
            else
                Turn right randomly
            end
        else
            Turn right randomly
        end
    end
else
    if canLeft then
        Turn left randomly
    else
        if canRight then
            Turn right randomly
        end
    end
end

epsilonBoolean() return random real in [0,1] ≤ epsilon
```

Algorithm 2: Right-Wall Following Policy Pseudocode

```
main loop frontDistance ← read front ultrasonic sensor
rightDistance ← read right ultrasonic sensor
speed ← calculateSpeeds(100, rightDistance, frontDistance)
set left motor to −speed.left
set right motor to speed.right
wait for (10 + speed.extra) milliseconds

calculateSpeeds(baseSpeed, rightDistance, frontDistance) deviation ← rightDistance −
  sideThreshold
if frontDistance ≤ frontThreshold then
    extra ← 400
    turnFactor ← constrain((3·frontThreshold−frontDistance)/frontThreshold, 0, 0.5)
    leftSpeed ← baseSpeed ·(1.0− turnFactor)
    rightSpeed ← baseSpeed ·(1.0+ turnFactor)
else
    extra ← 0
    if |deviation| ≤ 2 then
        leftSpeed ← baseSpeed
        rightSpeed ← baseSpeed
    else
        if deviation > 0 then
            rightSpeed ← max(30, baseSpeed − (2 · deviation)^1.2)
            leftSpeed ← baseSpeed
        else
            leftSpeed ← max(30, baseSpeed − (2 · |deviation|)^1.2)
            rightSpeed ← baseSpeed
        end
    end
end
return (leftSpeed, rightSpeed, extra)
```

Algorithm 3: Greedy Spatial Balancing and Obstacle Avoidance Policy

```
main loop leftDistance ← read left ultrasonic sensor
frontDistance ← read front ultrasonic sensor
rightDistance ← read right ultrasonic sensor
speed ← calculateSpeeds(100, leftDistance, frontDistance, rightDistance)
set left motor to −speed.left
set right motor to speed.right
wait for (3 + speed.extra) milliseconds

calculateSpeeds(baseSpeed, leftDistance, frontDistance, rightDistance) if frontDistance <
  FRONT_THRESHOLD then
    if rightDistance > leftDistance + 25 then
        leftSpeed ← baseSpeed + 70,   rightSpeed ← MIN_SPEED
                                                          // Turn right
    else
        if leftDistance > rightDistance + 25 then
            leftSpeed ← MIN_SPEED,   rightSpeed ← baseSpeed + 70
                                                           // Turn left
        else
            leftSpeed ← baseSpeed + 70,   rightSpeed ← MIN_SPEED
                                                       // Prefer right
        end
    end
    extra ← 250
else
    totalDistance ← leftDistance + rightDistance
    if totalDistance = 0 then
        totalDistance ← 1
    end
    leftForce ← TOTAL_FORCE · (rightDistance / totalDistance)
    rightForce ← TOTAL_FORCE − leftForce
    leftSpeed ← map(leftForce, 0 → TOTAL_FORCE, MIN_SPEED → MAX_SPEED)
    rightSpeed ← map(rightForce, 0 → TOTAL_FORCE, MIN_SPEED → MAX_SPEED)
    extra ← 0
end
return (leftSpeed, rightSpeed, extra)
```

Algorithm 4: Left-Wall Following Policy Pseudocode

```
main loop frontDistance ← read front ultrasonic sensor
leftDistance ← read left ultrasonic sensor
speed ← calculateSpeeds(100, leftDistance, frontDistance)
set left motor to −speed.left
set right motor to speed.right
wait for (10 + speed.extra) milliseconds

calculateSpeeds(baseSpeed, leftDistance, frontDistance) deviation ← leftDistance −
  sideThreshold
if frontDistance ≤ frontThreshold then
    extra ← 500
    turnFactor ← constrain((3·frontThreshold−frontDistance)/frontThreshold, 0, 0.5)
    leftSpeed ← baseSpeed ·(1.0+ turnFactor)
    rightSpeed ← baseSpeed ·(1.0− turnFactor)
else
    extra ← 0
    if |deviation| ≤ 2 then
        leftSpeed, rightSpeed ← baseSpeed
    else
        if deviation > 0 then
            leftSpeed ← max(30, baseSpeed − (2·deviation)^1.15)
            rightSpeed ← baseSpeed
        else
            rightSpeed ← max(30, baseSpeed − (2·|deviation|)^1.2)
            leftSpeed ← baseSpeed
        end
    end
end
return (leftSpeed, rightSpeed, extra)
```