

# Training LLMs to translate Mission Descriptions in Natural Language into Controller for Robot Swarms

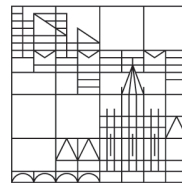
Master Thesis

submitted by

Julian Jandeleit (01/836946)

at the

Universität  
Konstanz



Cyber-Physical Systems

Department of Computer and Information Science

1. Reviewer: Prof. Dr. Heiko Hamann
2. Reviewer: Jun.-Prof. Dr. Andreas Spitz

Konstanz, 26th February 2025

## Abstract

Collective robotics is a challenging field, particularly when designing controllers for swarm robots, where complex inter-robot interactions lead to unexpected collective behavior on the global level. Traditional methods based on evolutionary algorithms can solve individual tasks but require extensive computation, and human expertise for modeling and defining fitness functions, and typically yield controllers that do not generalize across different types of missions. For example, a controller for aggregation cannot be used on a foraging mission without modification.

This master thesis explores the potential of Large Language Models (LLMs) to serve as a method for general swarm behavior generation. LLMs have already demonstrated the ability to solve a wide range of tasks in robotics already. This work explores whether LLMs can bridge the gap between individual robot actions and overall swarm behavior on the global level by generating robot controllers for swarm missions. This could have the potential to further enable natural language interaction with swarms as well as one-shot generation of valid controllers without the need for dedicated optimization.

In the scope of this thesis, a dataset of natural language descriptions and robot controllers is created. Using AutoMoDe-Maple, controllers for the mission types Aggregation, Foraging, Distribution, and Connection are obtained. In several experiments, a 7B LLM is fine-tuned and the LLM-generated controllers are compared against the AutoMoDe baseline. Factors such as dataset size, the impact of natural versus formalized language prompts, reinforcement learning, and transferability to unseen scenarios are investigated.

The results indicate that, while the fine-tuned LLM can replicate state-of-the-art controllers in individual cases and show promising generalization capabilities by generating viable controllers for mission types not encountered during training, their average performance does not reach that of AutoMoDe. It shows large variability regarding quality. These findings suggest, that LLMs hold potential as a general model for swarm behavior and are viable for controller generation from natural language, but further research with larger models and more diverse training datasets is necessary for more decisive and optimal results.

# Contents

Abstract . . . . .	i
1 Introduction . . . . .	1
2 Background . . . . .	3
2.1 Robot Swarms . . . . .	3
2.2 AutoMoDe-Maple . . . . .	8
2.3 LLMs and Robotics . . . . .	11
3 Experiments Pipeline . . . . .	17
3.1 Dataset Collection . . . . .	17
3.2 Reference Controller Design . . . . .	22
3.3 LLM Training . . . . .	24
4 Experiments and Results . . . . .	26
4.1 AutoMoDe Baseline . . . . .	26
4.2 LLM Experiments . . . . .	28
5 Discussion . . . . .	42
6 Conclusion . . . . .	44
Appendix . . . . .	45
References . . . . .	50

# 1 Introduction

Robotics is a rapidly evolving interdisciplinary field, where systems are developed that perform programmable actions in the real world (Ben-Ari and Mondada, 2017). Robots take different forms, from fixed arms to airborne robots and walking robots, whose shape mirrors that of humans (Ben-Ari and Mondada, 2017). Robotics has found wide application in many areas such as industrial manufacturing, medicine, environmental monitoring, and home assistance (Siciliano and Khatib, 2016).

Many tasks for single-robot systems can be replaced by a number of simpler and cheaper robots when collaborating as a swarm. Then, collective robotics has the opportunity to build more robust and adaptive systems that can scale comparatively easily with the number of robots and the size of the task (Hamann, 2018).

Robot swarms have a diverse set of applications, as summarized by Dias et al. (2021). Applications include logistics and industry, for example in smart warehouses. Monitoring can be performed in different environments like under water or from the air. This has advantages for farming, where additional laborious tasks can be conducted by robot swarms as well. Their robustness makes them a suitable option for extreme environments such as space for exploration and in-orbit assembly.

Usually, a swarm is considered to consist of identical robots that share both hardware and software, without relying on a global coordinator. Thus, swarm behavior needs to emerge from local interactions alone. This can lead to a self-organization effect, in which simple local interactions can lead to strong complex patterns at the global level (Camazine et al., 2001). Predicting how these global patterns emerge from the behaviors of individual agents is a challenging approach, especially when such a system is to be designed (Hamann and Wörn, 2007). This bridge between the microscopic and macroscopic world can be called the *micro-macro link* (Hamann and Wörn, 2007).

The micro-macro link in swarm robotics has previously been approached in different ways, from manual modeling with Brownian agents (Hamann and Wörn, 2008), swarm languages like Proto (Beal and Bachrach, 2006) that use an amorphous medium (Abelson et al., 2000) as a swarm abstraction (Pinciroli and Beltrame, 2016), to formal design methods rooted in evolutionary computing like AutoMoDe (Francesca et al., 2014).

Each of these methods looks at different parts of the micro-macro problem. They have in common that challenges arise with qualitative changes in the scenario. Either they are only intended for a single class of applications, like Brownian agents for spatial movement, or a considerable amount of work and resources need to be spent to accommodate for new constraints. For example, Proto needs to adjust its library of elementary behaviors if these constraints change. In addition, a new implementation needs to be found for the new desired behavior by the programmer. Similarly, for AutoMoDe, a new fitness function needs to be designed, implemented, and optimized for each mission. This in itself is not always straightforward and resource-intensive. The respective robot behavior would need to be optimized again.

Thus, an overall model that understands the general relationships in the micro-macro dynamic, would be desirable. Such a model could adjust robot controllers for different scenarios according to specifications in a zero-shot manner and accurately estimate the effect of how microscopic interactions change macroscopic outcomes.

Recently, machine learning with the transformer architecture by Vaswani et al. (2017) has led to many advances in the field of natural language processing. Although originally designed for language translation, it was quickly adapted to be used for other tasks such as text classification and question answering (Devlin et al., 2019). In tasks like question answering, they are reported by Taloni et al. (2023) to achieve better scores than humans in some test variants. Training transformers on a corpus of text results in them building contextualized word representations (Peters et al., 2018), a high-dimensional vector that gets assigned to every word. This way, a pre-trained transformer can internally represent sequences of words as a sequence of features describing their semantic meaning. By training transformer models with enough

parameters and texts, the internal features can be leveraged to generate text output, solving even more complex tasks (Zhao et al., 2023). Those kinds of models are then called Large Language Models (LLMs). They can be adapted to specific tasks by fine-tuning on task-specific data.

Large Language Models have already been used to solve tasks in robotics. Cao and Lee (2023) use LLMs to convert behavior trees across domains like car manufacturing and PC assembly. Bucker et al. (2023) demonstrate the use of transformers to modify the existing trajectories of a robotic arm in the real world by encoding both language, image, and geometry information in the prompt. Brohan et al. (2023) build a multi-modal transformer model that directly outputs robot actions for natural language instructions as part of the text token encoding space. Lykov and Tsetserukou (2024) fine-tune an LLM to generate the behavior tree for an individual robot from a textual description and predefined nodes. Yu et al. (2023) use LLMs to generate a reward function from a task description, which can then be used to optimize the controller of a robot.

This body of research shows that LLMs can be successfully applied to robot tasks. Their adaptability to new fields and success in those, especially robotics, opens the question of whether Large Language Models can also be applied to bridge the micro-macro gap. Little work has been done for multi-robot setups, and especially LLMs for swarm robotics. Existing research focuses mostly on multi-robot interaction rather than larger distributed robot swarms and looks at making use of the models' pre-trained capabilities using prompt engineering, rather than trying to improve those with fine-tuning.

Using evolutionary methods such as AutoMoDe-Maple (Kuckling et al., 2018), concise textual representations of behavior trees as robot controllers for several classes of swarm missions can be generated and optimized individually. If done in a structural way, by varying parameters of these missions, a dataset of natural language descriptions of different scenarios and working robot controllers can be obtained.

Can such a dataset be used to fine-tune a pre-trained LLM to generate working robot controllers for diverse swarm missions? To what extent does the trained model learn a general representation of collective behavior that it can use for unseen tasks? According to the literature reviewed for this thesis, no such model has been trained and evaluated before. To approach these questions, this thesis evaluates experimentally if LLMs are even suitable for creating controllers for swarm robotic missions using the aforementioned approach.

Specifically, the following is done: First, a dataset of instances for 4 different kinds of swarm missions is created. This dataset contains a description of a specific scenario in natural language, as well as configuration for AutoMoDe-Maple. In the second step, the missions are implemented for AutoMoDe-Maple and a behavior tree for each scenario is optimized. Finally, a pre-trained LLM is fine-tuned on the mission descriptions and behavior trees in order to generate behavior trees as robot controllers on its own. Fine-tuned LLMs using this approach are evaluated in different kinds of experiments.

Natural language like informal prompts have the advantage of allowing more intuitive interaction with robot swarms, potentially even by non-experts. LLMs can give zero-shot answers to unseen scenarios so that they have the potential to give results without dedicated optimization at inference time. As motivated in this section, the biggest opportunity for LLMs in swarm robotics could be that they potentially become a model for general translation between the microscopic and macroscopic world and generalize, so that they can generate swarm controllers for previously unseen tasks. If successful, this work could pave the way for more large-scale experiments, aimed at a more thorough understanding of swarm systems and targeted on more diverse kinds of dynamics in collective systems.

## 2 Background

This section gives an overview of relevant background and similar approaches to those used in this thesis. First, robot swarms are introduced in general, and then works that describe how to automatically obtain controllers for robot swarms are shown. Finally, related work about how LLMs have been employed to control robots is discussed.

### 2.1 Robot Swarms

[Dorigo and Şahin \(2004\)](#) define *swarm robotics* as "the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment". From this definition, several key aspects of robot swarms can be extracted. A swarm consists of agents that are in large number, physically embodied and simple. In this thesis, agents are considered to be individual robots, either physically or in simulation. For context, the more general field of multi-robot systems usually assumed to not exceed a handful of robots, employing more complex interaction capabilities and central control.

The second part of the definition is a so-called *collective behavior*. Collective behavior of a robot swarm can be understood as the "resulting overall behavior of all swarm members" according to [Hamann \(2018\)](#). In this sense, *collective robotics* can be seen as a synonym for *swarm robotics*. Remembering the aforementioned definition of swarm robotics, collective behavior has several properties:

- a desired objective,
- emerges from local interactions,
- emerges from interactions with the environment.

The key of this definition is the desired behavior of the overall swarm, which originates from local interactions with the swarm members. Usually, these local interactions are interpreted to be controlled in a decentral way, without a central coordinator. This decentralization is highlighted by [Beni and Wang \(1993\)](#) as one of the key attributes that show the similarity of robot swarms with insect swarms found in nature. A second attribute they highlight is that the large number of robots are usually considered quasi-identical. This includes the physical structure of the robots, as well as their driving logic. The attributes of robot swarms discussed in this section are also present in the experimental work of this thesis.

#### — 2.1.1 Advantages of Robot Swarms

The definition from Section 2.1 leads to several desirable properties of robot swarms, as described in [Şahin \(2005\)](#):

Robustness is a direct result of the large number of quasi-identical robots. This redundancy overcomes the problem of a single point of failure so that a certain number of robots can fail without significantly altering the swarm's ability to achieve its objective. Similarly, robot swarms show a large range of flexibility because limitations of individual robots can often be overcome by cooperation. Furthermore, by only interacting locally, the swarm can easily scale in the number of swarm members and the physical area covered without modification. Moreover, scaling the swarm size can even lead to super-linear performance in certain cases, as well as emergent behavior due to the internal complexity and a high number of local interactions ([Darley, 1994](#)).

### — 2.1.2 Missions and Scenarios

This thesis defines a robot *mission* as the general kind of environment, robots, and objective constraints of a task involving robots. In contrast, the term *scenario* describes a specific instance of the above constraints. For example, a mission could be an aggregation mission, where  $n$  robots are distributed uniformly with the objective of aggregating at a ground are placed in the environment. A corresponding scenario would be the actual size of the arena, the number of robots, and the placement of the black circle and light. A visualization of such a scenario is shown in Table 1 on Page 21.

A formal definition as used in the creation of the training dataset is given in Section 3.1.1

### — 2.1.3 Swarm Behaviors

There are several kinds of behaviors that are typically found in collective robotics, often inspired by biological observation (Brambilla et al., 2013, Schranz et al., 2020). Overviews over common swarm behaviors are for example given in Dias et al. (2021) or Hamann (2018). This thesis focuses on variants of four frequently found behaviors. For most swarm behaviors, several possible variants exist, so the following paragraph defines the understanding of each behavior as used in the thesis's experiments.

**Aggregation** is a behavior, where all robots have to aggregate together at a specific spot. The spot is indicated as a specific color on the ground, making it detectable by the robots.

**Distribution** is a behavior, where the robots have to spread out uniformly to cover an area of a given size. Here, the actual spot or area is not defined by the environment and is to be dynamically decided by the swarm.

**Connection** is a behavior, where the robots physically connect two places by forming a line between them. How the connection is made is not defined. For example, the swarm can decide to stand still after having formed a line or have its robots move continually between both places. The places are defined by a distinct ground color.

**Foraging** is a behavior, where the robots have to carry objects from a source region to a target region. The focus is not on how robots pick up and release objects but on how the swarm can effectively transport as many objects from source to target as possible. The source and target regions are indicated each by a distinct ground color.

An example for each behavior can be seen in Table 2 in Section 4.1 when the actual experiments are discussed.

### — 2.1.4 Micro-Macro Gap

A robot swarm can be looked at from two points of view: a global perspective, looking at the swarm as a whole, and a local perspective looking at the individual robots. Hamann (2018) defines these as the microscopic and macroscopic perspectives, respectively, and argues that, while the macroscopic perspective may not necessarily be physically existent, both perspectives can be considered distinct. He argues that, for example, in systems with collective perception or collective memory, there exists a global

level of understanding that is distributed over the swarm members and not present on the individual level of each robot, which only has the limited understanding of its direct neighborhood.

Finding a connection between both layers could be considered a “micro-macro link” (Hamann and Wörn, 2007) and, in its absence, the micro-macro gap emerges. As the objective behavior of a robot swarm is defined on the macroscopic level, finding a micro-macro link is essential for effectively designing swarm robot systems. Finding a link between both levels is challenging because in complex robot interactions on the micro level, emergent behaviors on the global level can arise that are difficult to predict (Darley, 1994).

While there exist several kinds of methods, as described in Section 2.1.6, the distinction between the microscopic and macroscopic levels still has implications for the design of swarm robot systems and poses a “grand challenge” (Hamann, 2018).

A specific method for obtaining microscopic robot controllers for each robot is discussed in Section 2.2. This method is used for creating controllers for the fine-tuning dataset that is used to train the LLM in this thesis.

#### — 2.1.5 Agent-based Swarm Modeling

Generally, modeling robot swarms can be done in two ways. On one hand, one can start from the global swarm-level specification. In this top-down approach, a global specification gets translated into requirements for every agent. In the bottom-up approach, the capabilities of each agent are aggregated toward possible outcomes on a global level. Both engineering approaches can produce similar results; however, the top-down approach, which is the focus of this thesis, can become nonviable with swarms having limited communication and information propagation (Crespi et al., 2008).

For example, **Rate-Equations** are inspired by chemistry and were introduced by Martinoli et al. (1999). Here, an individual robot is assumed to have several states. The robot swarm on the global level is described as the fraction of the swarm that is in each state (Hamann, 2018). The dynamics of the system is described by state transitions whose likelihoods are formalized as the rate at which robots in one state transition to another specific state.

A more spatial approach is taken by Hamann and Wörn (2008), who apply **Brownian Agents** (Schweitzer, 2007) to swarm robotics. Inspired by statistical physics, the trajectory of a robot is described by a stochastic differential equation consisting of both a deterministic and a non-deterministic part. From this equation, the trajectory of an individual robot is described by the Langevin equation with the global swarm behavior being described by a respective Fokker-Planck equation. These result in microscopic trajectories of individual robots and macroscopic swarm densities, respectively.

These kinds of models are useful for analysis and successfully predict swarm-level properties such as the speed of collective motion (Hamann and Wörn, 2008). However, they do not provide a formal recipe, for creating arbitrary complex robot behavior, such as foraging, from the global description. For that, iterative modeling and testing is required. Modeling the inter-robot communication especially is a very difficult task (Brambilla et al., 2013).

#### — 2.1.6 Formal Design of Swarm Controllers

A straightforward approach to designing robot controllers for swarms is educated trial and error. Starting from a high level of abstraction, assumptions are simulated and the abstraction level is iteratively reduced while modifying the controller until a final controller is evaluated on a real robot. This approach can also



be used to develop swarm controllers

(Hamann, 2018). While this approach can lead to a desired swarm behavior, extensive experience and work need to be put into each step of abstraction reduction. The approach is limited in that small incremental steps are not always feasible to produce the desired result. Then, larger, more fundamental changes need to be applied (Hamann, 2018), making it more difficult to achieve the desired outcome. Thus, this technique is not feasible to obtain arbitrary swarm behaviors at larger scales.

More automatic techniques involve training a controller model using machine learning (Hamann, 2018). Training a controller using backpropagation (Rumelhart et al., 1986) requires a model described in a differentiable form and labeled training data consisting of features and labels. Despite recent successes in other fields, deep learning techniques cannot be applied to swarm robotics in a straightforward way (Hamann, 2018). On the other hand, reinforcement learning, with one of the earliest works being Watkins (1989), and evolutionary algorithms as discussed in Eiben and Smith (2015) have different requirements. In these methods, a score called reward or fitness, respectively, is optimized directly without the need for a differentiable model or a labeled training set. However, an execution environment must be present as well as a function that evaluates the performance of a given controller. These techniques have been successfully applied to swarm robotics (Hamann, 2018).

However, designing a proper fitness function is a complex process in itself, especially when the complexity of the behavior increases, resulting in an even more intricate function Mataric and Cliff (1996). These systems usually target a specific scenario and do not necessarily generalize to other behaviors. Instead, they require individual training on every specific scenario. Further, evolutionary robotics suffers from the reality gap, which describes that behaviors optimized by evolutionary algorithms in simulation do not always transfer to the real world (Floreano and Mattiussi, 2008).

There exist methods that are shown to generalize across related environments for the same kind of target behavior. Ericksen et al. (2017) use neuroevolution of augmented topologies (NEAT) Stanley and Miikkulainen (2002) to obtain a multilayered perceptron. NEAT is a kind of evolutionary algorithm—more specifically, a genetic algorithm, that simultaneously evolves both the structure and the weights of a neural network. The authors apply NEAT to evolve a swarm controller for foraging behavior. They show that their method, **NeatFA**, can “discover behaviors that generalize across environments and scale with swarm size.”

Similarly, Ferrante et al. (2013) use an evolutionary approach for their **GESwarm** method. However, in this case the controller representation is not a neural network, but a set of rules evolved by a specially developed grammar. They report that the resulting controllers “significantly outperformed a hand-coded foraging collective behavior.” Both NeatFA and GESwarm use a version of ARGoS Pinciroli and Beltrame (2016) as the simulator, the robot swarm and respective controller instances get evaluated in.

A different class of methods does not train a controller from scratch but views controller generation as a composition problem. There, elementary behaviors are combined in such a way as to maximize controller performance. Nagavalli et al. (2017) propose an algorithm for **optimal sequencing** of basic behaviors for a navigation task. They report that their algorithm finds the optimal sequence for a task but does not meet real-time requirements. Also, their base behaviors are already swarm behaviors on a global level, such as “Flock North.” This means their algorithm does not need to cross the micro-macro gap.

The task of controller generation can also be viewed as a composition problem from the perspective of an individual robot. **AutoMoDe** by Francesca et al. (2014) defines elementary robot behaviors, such as phototaxis or random walk, which are then optimized to perform a swarm behavior on the global level. The authors also use an evolutionary algorithm for optimization. Thus, fitness functions need to be formulated and then evaluated in ARGoS (Pinciroli et al., 2012), as done by others mentioned above. AutoMoDe directly optimizes a controller for a specific behavior. By doing so, the micro-macro gap is crossed for this specific scenario. While the results are reported to successfully cross the reality gap as well, they do not generalize to different kinds of behaviors at the same time as these need to be optimized

individually.

In contrast to these methods, this thesis evaluates LLMs as a general method to potentially work for different mission types simultaneously and with natural language prompts without the need for individual optimization of each scenario.

### — 2.1.7 Swarm Languages

In the previous methods, the intent of the desired behavior needs to be communicated to the optimization algorithm. In most cases this is done using a fitness function that specifies how well a controller fits the objective. Other constraints, such as the environment, are usually implicitly included by shaping the simulation accordingly. Expressing all constraints in a more formal way allows for finer control of the parameters. Ideally, this can be exploited by optimization algorithms and eventually lead to a form of declarative programming for robot swarms where the user inputs a global description of the conditions and desired outcomes, which is then translated into instructions for the individual robots (Hamann, 2018).

Khurshid (2011) defines a categorical markup language (**CML**), inspired by category theory, to specify formal descriptions of different swarm scenarios and behaviors for modeling and verification purposes.

Pinciroli and Beltrame (2016) define **Buzz**, an extensible programming language for multi-robot systems to express swarm algorithms. In Buzz, the robots communicate via virtual stigmergy running on their custom virtual machine on each robot. While the programmer can use Buzz to express a swarm algorithm, the design of the controller to achieve the desired behavior is not addressed.

A different approach is taken by Bachrach et al. (2010), who use the **Proto** language (Beal and Bachrach, 2006) to program collective behavior. They assume a robot swarm to be similar to an *amorphous medium* (Abelson et al., 2000), which is continuous in nature. Swarm programs are described using “functional operations on fields of values” and compiled to a local perspective for each point in the medium, and then discretely approximated. Compilation is done using a predefined library of abstract elementary behaviors, such as gradients through the medium and their required local rules. However, the challenge with this approach is to “translate these abstract descriptions into executable code” (Hamann, 2018) for individual points.

Similarly, Aguzzi et al. (2023) use the *aggregate computing* approach for programming robot swarms in their **MacroSwarm** system. In contrast to the amorphous medium, aggregate computing is more discrete in nature as its system model consists of a set of computing nodes equipped with sensors and actuators that are connected to a network via neighborhood relationships. Thus, behaviors can be expressed, for example, using the *ScaFi* aggregate programming toolkit in Scala. A library of elementary behaviors is implemented by the authors using this framework. They state that their approach increases the practicality and compositionality in swarm programming while maintaining a scalable formal approach. However, the implementations of base behaviors that are already at a global level need to be identified first.

Another approach is taken by Bozhinoski and Birattari (2022) in their Swarm Mission Language (**SML**). The authors acknowledge that the behavior often depends not only on the swarm itself but also on the environment in which the mission takes place. Their language is separated into several components describing different aspects of the mission. The language is formal, yet can be expressed in sentences similar to natural language. The desired behavior can be specified by describing how the fitness function is computed according to a pre-implemented template. The natural language-like descriptions correspond to elements in their simulation and optimization framework, AutoMoDe (Francesca et al., 2014), which

computes the robot controller from individual behaviors to maximize fitness.

In general, these swarm languages focus on the specification part of swarm behavior design, while simplifying certain aspects through composition. The implementation of the elementary swarm behaviors is still assumed to have been modeled already. These and the final expressions need to be implemented by experts. While these works primarily focus on formal specification aspects, the formal design aspect itself is often only indirectly addressed. How the desired behavior is obtained from the composition is not always obvious and is only addressed directly by [Bozhinoski and Birattari \(2022\)](#) with subsequent optimization of fitness.

## 2.2 AutoMoDe-Maple

The robot controllers of AutoMoDe-Maple ([Kuckling et al., 2018](#)) are used for training of the LLM in the experiments and later as reference for its performance. As it is a major building block for the experiments in this thesis, it is discussed in this section in more detail.

The basis of the thesis is the automatic design of robot controllers, specifically controllers for individual robots in decentralized homogeneous robot swarms. A robot controller is the processing part of the robot that transforms sensory inputs into actuator outputs that steer the robot. In collective robotics it is typically assumed that every robot has an individual copy of the same controller that every swarm member uses. In this sense, the same controller is shared across the swarm and will also be called the *swarm controller* from now on. This is illustrated in the following figure.

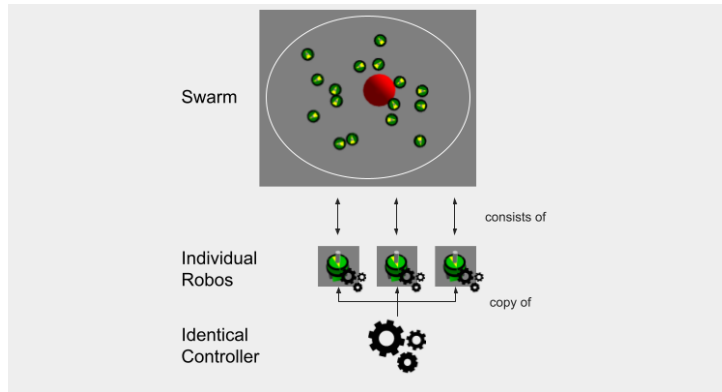


Figure 1: This figure illustrates what is considered to be a swarm controller. The swarm of robots in green (around a red light) consists of several identical individuals. Each individual is controlled independently by its copy of the same controller. In AutoMoDe-Maple, this controller is a behavior tree. The robots depicted are visualizations from ARGoS ([Pinciroli et al., 2012](#)).

One popular family of frameworks that can create working swarm controllers is AutoMoDe ([Francesca et al., 2014](#)). Inspired by evolutionary robotics, AutoMoDe selects and arranges preexisting modules into a complete robot controller and has been shown to successfully overcome the reality gap ([Francesca et al., 2014](#)). The reality gap is a common challenge in evolutionary robotics, as results obtained from simulation do not necessarily apply to the real world ([Floreano and Mattiussi, 2008](#)). The final behavior that AutoMoDe creates is determined by defining a fitness function appropriate to the desired objective. During optimization, which is described in Section 2.2.2, AutoMoDe finds those configurations that maximize fitness.

This thesis uses AutoMoDe-Maple which is a variant of the AutoMoDe family. It builds on top of AutoMoDe-Chocolate (Francesca et al., 2015) by adapting the control structure of finite state machines to use behavior trees as robot controllers instead. AutoMoDe-Maple defines several base behaviors and conditions for individual robots from which complex more complex individual behaviors are assembled. The behaviors are influenced by the capabilities of the e-puck robot (Mondada et al., 2009), which is the underlying robot platform used by AutoMoDe-Maple.

AutoMoDe-Maple defines the following low-level behaviors:

- *Exploration*: Defines a random walk consisting of going straight and turning for a random duration if an obstacle is observed.
- *Stop*: The robot stands still.
- *(Anti-) Phototaxis*: The robot moves toward the light source or away from it, respectively.
- *Attraction/Repulsion*: The robot moves toward or away from neighbors, respectively.

The last two behaviors default to a straight line with obstacle avoidance if a detection threshold is not met.

Additionally, the following conditions can be detected:

- *Black/Gray/White-Floor*: True if the ground has the respective color.
- *(Inverted-) Neighbor-count*: True if a specific neighbor count is reached.
- *Fixed-probability*: True with a certain probability.

All conditions are probabilistic and some conditions and behaviors can be controlled more finely by specifying parameters. These basic modules are assembled into a complex behavior by AutoMoDe using an overarching model. Traditionally, this model has been finite-state machines (FSMs). AutoMoDe-Maple uses behavior trees, which were shown to be at least as good as FSMs for AutoMoDe by the authors of Maple, and are chosen here. Behavior trees are advantageous as they generalize several typical robot architectures (Colledanchise and Ögren, 2016).

As a central building block of the LLM training pipeline of this thesis’ experiments and a reference for baseline performance, AutoMoDe-Maple will be mentioned repeatedly throughout this thesis. If the focus is not on the distinction of Maple to other variants of AutoMoDe, it will for brevity often be referred to as AutoMoDe only.

### — 2.2.1 Behavior Trees

Recently, behavior trees have become increasingly adopted for robotic control (Colledanchise and Ögren, 2016), with potential as a good intermediate representation to be automatically generated. Similar to decision trees, they are evaluated from the top until a leaf is reached. The leaves represent the actual actions that are executed by the robot, while the nodes encapsulate the control structure logic. Here, the definition by Marzinotto et al. (2014) is described.

There is the root node and two other types of nodes:

- **Control-flow nodes** with subtypes: Selector, Sequence, Parallel, and Decorator.
- **Execution nodes** with subtypes: Action and Condition.

Each node type has a respective algorithm that controls its behavior. At a fixed frequency, starting from the root, each node emits a tick to its child nodes, triggering their algorithms and deciding on its state, which can be either **Succeed**, **Running**, **Failed**, or **NotTicked**. These states are passed up the tree to the root node depending on the respective node algorithms:

$$\text{state} \leftarrow \text{Tick}(\text{child}(i))$$

- **Selector nodes** tick their children until the first returns a non-failure state.
- **Sequence nodes** tick their children until the first non-success state is reached.
- **Parallel nodes** tick every child and return a state depending on thresholds of their children's states.
- **Decorator nodes** have exactly one child. This child is ticked based on conditions evaluated on internal variables. The state is returned depending on applied internal functions.
- **Action nodes** evaluate internal state space and return that state. In the case of **Running**, a control step is executed.
- **Condition nodes** behave like action nodes but do not have a **Running** state.

Extensions allow behavior trees to enable memory by storing a pointer to the most recent running child and synchronization of sub-trees of multiple agents using message broadcasting.

The implementation of behavior trees that is used by [Kuckling et al. \(2018\)](#) has a text representation as command line arguments. This representation is used for training the LLM and to report examples in this thesis. It generally follows the structure of first stating the object and then then its properties using multiple following arguments. They can be visualized in an interactive tool by [Kuckling et al. \(2021\)](#). An example of such a visualization can be found in Figure 2. It visualizes the following behavior tree, created by AutoMoDe for an *Aggregation* mission (line breaks are for visualization only and do not belong to the tree):

```
--nroot 3 --nchildroot 4
  --n0 0 --nchild0 2
    --n00 6 --c00 5 --p00 0.6627
    --n01 5 --a01 3 --p01 0
  --n1 0 --nchild1 2
    --n10 6 --c10 0 --p10 0.9127
    --n11 5 --a11 4 --att11 4.3675 --p11 0
  --n2 0 --nchild2 2
    --n20 6 --c20 5 --p20 0.6156
    --n21 5 --a21 2 --p21 0
  --n3 0 --nchild3 2
    --n30 6 --c30 5 --p30 0.0786
    --n31 5 --a31 1 --p31 0
```

The first element is always the node identifier (like `--nroot` or `--n0`), followed by the type of node. Types 0 and 1 define selector nodes, 2 and 3 sequence node. The second id indicates a variant with memory that gets visualized with a `*` and prevent repeated execution of already executed children until all children finish ([Colledanchise and Ögren, 2018](#)). Type 4 is a decorator node, 5 an action and 6 a condition. Different versions of the same node type are specified by `--c[nodeID]` parameter for conditions and `--a[nodeID]` parameter for actions. Conditions range from 0 – 6 with the order black floor, gray floor, white floor, neighbors count, inverted neighbors count, fixed probability and light. Depending on

condition they have additional attributes like probability ( $p$ ) or neighbors and epsilon ( $p, w$ ). Actions range from 0 – 5 in the order exploration, stop, phototaxis, anti-phototaxis, attraction and repulsion. The all have a success probability parameter  $p$ . Additional parameters for some actions are steps range ( $rwm$ ), attraction ( $att$ ) and repulsion  $rep$ .

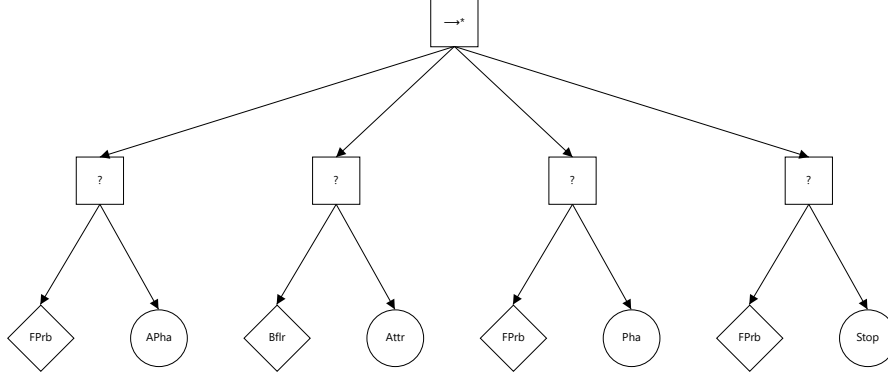


Figure 2: Visualization of a behavior tree for an *Aggregation* mission. It shows a sequence root node with memory and four selector children. The first child has a *fixed probability* condition depicted in a rhombus. If it returns a failure state, the selector node activates its second child, an *anti-phototaxis* action. Most nodes are successful only with certain probability, but node parameters like probabilities not shown. The second selector has a *black floor* condition that activates an *attraction* action if the condition is not met. The third selector node activates *phototaxis* and the last one a *stop* action, each with a given fixed probability. For reference, the same controller is also displayed in Tables 1 and 2

### — 2.2.2 Model Optimization

To optimize the behavior tree to produce maximum fitness for a given scenario, AutoMoDe-Maple uses iterated F-race (Birattari et al., 2010) for optimization (see also Kuckling et al. (2018)). Here, the scenario is fixed in every aspect except the realization of the uniform distribution of robots and, of course, the robot controller. Starting from a uniform sample of parameters, a population of solutions is formed. Each solution represents a specific robot controller, in the case of AutoMoDe-Maple, a behavior tree. In each so-called *race*, the population is evaluated on the given scenario. For robustness, each scenario is evaluated multiple times. Similar to evolutionary methods, the worst solutions according to the Friedman test (Friedman, 1937, Conover, 1999) are removed. To refill the population to match the initial size, new solutions are sampled from the survivors and repeated races are performed until one solution remains or a maximum number of races is met.

## 2.3 LLMs and Robotics

Another key part of the thesis is Large Language Models (LLMs). Recently, they have emerged as a powerful tool not only in natural language processing (NLP) but also in many adjacent fields (Zhao et al., 2023).

### — 2.3.1 Transformers and Next Token Prediction

An initial driver of the recent success of language models has been the transformer architecture (Vaswani et al., 2017). In contrast to LSTMs (Hochreiter and Schmidhuber, 1997), the transformer processes the entire input sentence as a whole, eliminating the forgetting problem (Wu et al., 2020). It consists of an encoder and a decoder. First, the text input sequence is split into parts called *tokens*. A word can consist of one or multiple tokens. Each token is assigned a vector, which is the actual data model used by the transformer. The encoder consists of a stack of attention modules that convert the input into an internal representation, or *embedding*. The decoder is another stack of attention modules that takes the embedding and the previously generated output tokens as input. Starting from an empty sequence, the decoder is applied iteratively, appending the next token until a stop token is produced.

Transformers show signs of capturing the semantic structure behind sentences (Vaswani et al., 2017). In natural language processing (NLP), semantic understanding can be conceived as suggested by Mikolov et al. (2013): the semantics of a word is understood by the words it is similar to. This concept extends to the vector space of encoded words: two words are similar when the difference between their vector representations is small according to cosine similarity (Mikolov et al., 2013). Such a vector space is recovered through training, allowing words to have different meanings depending on their surrounding context. This leads to contextualized word representations, which are also produced by transformers (Peters et al., 2018) and used for various downstream tasks such as text generation or text classification (Devlin et al., 2019).

### — 2.3.2 Large Scale Language Models

Enabled by their good parallelizability (Vaswani et al., 2017), transformers can be efficiently trained with a large number of parameters. Increasing model size strongly increases the performance of generative language models, to the point where the qualitative improvements obtained by merely increasing the number of parameters have been considered emergent (Wei et al., 2022a). The first such model was GPT-2, which generalized to a “surprising amount of tasks without the need for explicit supervision” (Radford et al., 2019). However, to achieve this generalization, the training dataset must be sufficiently diverse (Radford et al., 2019). These generative language models are referred to as *Large Language Models* (LLMs).

### — 2.3.3 Finetuning of Transformer-Based Language Models

Using a pre-trained language model and training it on additional data is called fine-tuning. Several methods have been developed for transformer-based LLMs. This section describes the two methods used in this thesis.

**Supervised Finetuning** The first strategy is supervised fine-tuning. It generally follows the training methodology described by Vaswani et al. (2017). A dataset of feature-label pairs is provided during training; the feature is the input text and the label is the output text generated by the transformer. The difference between the model output and the label is propagated back through the model to update the weights.

Typically, LLMs are pre-trained on dedicated training hardware that is not universally accessible. To feasibly finetune LLMs on workstations with VRAM on the order of 24 Gigabytes, special techniques



are required. For example, LLMs can be loaded with a lower memory footprint by compressing 16-bit weights to an average of 4.75 bits (Dettmers et al., 2023). Furthermore, only adapters with lower memory footprints can be trained and then applied to the model (Dettmers et al., 2024).

**Direct Preference Optimization** In contrast to supervised fine-tuning, which trains on a plain text basis, LLMs can also be trained on performance scores. Direct Preference Optimization (DPO) (Rafailov et al., 2024) is a variant of reinforcement learning for fine-tuned, pre-trained language models. Here, the loss is determined not directly by the text label but by the relative score it achieves. It is based on RLHF (reinforcement learning from human feedback) (Christiano et al., 2017), which was developed to communicate complex implicit reward functions to machine learning models. DPO requires two candidate solutions for training, along with their respective scores. The difference between the scores is then used to update the model’s weights, so that the model learns to quantify how good a response is.

The transformer architecture (Vaswani et al., 2017) enabled the recent success of LLMs. Soon, these LLMs were shown to be effective multitask learners (Radford et al., 2019). While originally developed for language translation, their understanding of human language and generalization capabilities have been applied to other fields, including robotics. This section presents their applications to the control of robots in general and collective robotics specifically.

#### — 2.3.4 Related Work

LLMs have been used as a control mechanism for robotic agents using both fine-tuning and prompt engineering which will be shown in this section. While fine-tuning changes the weights of a pre-trained LLM by supervised training on a specific dataset, prompt engineering leverages the model’s generalization and instruction-following capabilities by describing the task using natural language instructions, examples, and templating.

**Single Robot Systems** LLMs are used in settings where they directly output control actions for individual robots based on natural language input. Brohan et al. (2023) propose **RT-2**, the second iteration of a robotics transformer. They view robot actions as another language encoded into a shared token space. By decoding the output of RT-2, tokens representing actions are directly transformed into low-level robot actions (e.g., position and rotation changes).

RT-2 builds on the concept of an “Embodied Multimodal Language Model” (Driess et al., 2023). Driess et al. (2023) further propose **PaLM-E**, a model that combines natural language and images through neural scene representation to form “multimodal sentences.” This enables the model to dynamically encode and mix both modalities. The output can be fed into a control loop that executes the actions and updates the model with new environmental information, thus embodying the model in the world. PaLM-E and RT-2 both demonstrate the ability to generalize from the image domain to the real world (e.g., by performing the same tasks on new kinds of objects). However, in contrast to RT-2, PaLM-E only produces low-level language instructions (e.g., “open the top drawer”), whereas RT-2 returns actual robot actions. In this sense, PaLM-E functions more like a sub-task planner than a full controller. Nonetheless, primitive language instructions can be translated into real-time robot actions. In fact, Lynch et al. (2023) employ a transformer-based architecture for **visual-lingo-motor control** that converts text and image-based observation history into motor commands based on a large dataset of human demonstrations. Their model is capable of outputting new actions at 5 Hz.

There are variants of these applications. **LATTE** (Bucker et al., 2023) modifies the trajectory of a robotic arm based on natural language and image input, as well as the object poses and the original trajectory.



For example, their system can maintain a greater distance from certain scene objects upon instruction.

Due to the generalization abilities of LLMs, fine-tuning is not always necessary for robotic tasks. [Fan et al. \(2024\)](#) use prompt engineering techniques to generate high-level task plans and even produce G-code as machine instructions for industrial robots.

Many applications of LLMs in robotics involve planning smaller subtasks. In this way, the **Phase-Step prompt** method by [Cao and Lee \(2023\)](#) converts behavior trees from one domain to another by dividing tasks in the source behavior tree into several high-level “Phases.” Low-level actions such as “Align wheel” and “Insert screws” in the car maintenance domain are interpreted as “Connect RAM” and “Secure RAM” in the target domain of PC building.

Behavior trees are also employed by [Lykov and Tsetserukou \(2024\)](#) in their **LLM-BRAIn** model. They finetune a 7B-parameter model to generate XML behavior trees for a ROS2 interpreter. The input is a concise natural language instruction (e.g., “if object is visible, move towards it, take it and process it; else, scan the area”). The resulting behavior tree consists of elementary robot actions with descriptive IDs like “MoveToObject.” These trees were reported to be distinguishable from human-generated ones in 45.5% of cases.

Similarly, **BTGenBot** ([Izzo et al., 2024](#)) finetunes several LLMs (with at most 7 billion parameters) to generate behavior trees for a single mobile robot (TurtleBot3). Their models are fine-tuned on a custom dataset of 600 entries, obtained by providing `gpt-3.5-turbo` with an example behavior tree and asking for a description. Additionally, [Ao et al. \(2024\)](#) finetune three different LLMs—including the Mistral-7B model ([Jiang et al., 2023](#))—to generate behavior trees for robotic arms. They report “mostly perfect logical coherence,” although they note greater challenges in generating complete behavior trees compared to unit trees for a recursive approach.

**Multi-Robot Systems and Swarms** All previous methods focus on integrating LLMs into single robot systems and do not address multi-robot systems or swarm robotics and the micro-macro gap. However, there are a few initial works on LLMs in multi-robot systems.

**CLIPSwarm** ([Pueyo et al., 2024](#)) is a system for generating drone shows from a single input word. The authors select a fitting color by maximizing a text-to-image similarity measure derived from the CLIP vision-language model ([Radford et al., 2021](#)). The input word is expanded into a short expression (e.g., “A green Leaf shape”) to determine the optimal color. Then, a small population of random, elementary 2D robot formations is sampled and converted into concave contour shapes. The best shapes, as determined by the same similarity measure, are saved and varied to optimize the measure. The robots are controlled at a global level by the Optimal Reciprocal Collision Avoidance algorithm ([Snape and Manocha, 2010](#)). CLIPSwarm emphasizes creating shapes from text, as opposed to distributed navigation solved globally.

A study by [Mounsif et al. \(2023\)](#) investigates **flocking dynamics** from the global perspective, where robots have to move together to a target location in a coordinated motion. The authors study several prompting strategies and compare LLM chosen actions to actions chosen by reinforcement learning and find comparable performance and good versatility of the LLM regarding input prompt. However, they also find that prompting the model for direct output instead of giving it space to elaborate and reason dramatically reduces performance and diversity of the LLMs answers.

Another study ([Li et al., 2024](#)) investigates **multi-agent flocking**. Here, each agent uses its own independent LLM as a controller to decide movement. They employ prompt engineering by structuring the input into four parts with distinct natural language instructions. In the `agent_role` section, the model is informed that it is an agent in a 2D space. The `game_description` details the overall scenario and flocking instructions, explicitly reminding the model to “Keep in mind Boids flocking rules” ([Reynolds, 1987](#)). In the `round_description`, the current 2D position and those of other agents are provided. In the `output_format`, the model is instructed to first provide reasoning, followed by a concise output in the format “Position: [x, y]”. Depending on the LLM model used, syntactical success rates vary from

95% to 20%. In failure cases, the model did not adhere closely enough to the instructions to parse a new position. Qualitatively, about 60% of tests failed, with agents converging to the same spot instead of maintaining a consistent distance. The authors conclude that current LLMs, as individual decision-makers, are infeasible for flocking due to limited spatial reasoning, although future improvements may overcome this limitation.

These results align with the work of [Chen et al. \(2023\)](#) on **multi-agent consensus seeking**. In their study, each agent is assigned an LLM, prompted with the scenario, and asked to provide both a “Reasoning” and a “Position” in its response. The agent states its new state along with the states of others, encoded as an  $n$ -tuple in brackets (with  $n = 1$  for basic consensus and  $n = 2$  for a 2D aggregation experiment). The agents’ task is to agree on a numeric state. Several prompting variants and network topologies were evaluated; consensus was typically achieved via an averaging strategy. However, this same strategy—if applied to tasks like flocking—leads to failures, as noted in [Li et al. \(2024\)](#).

A full framework for robot collaboration is proposed by [Mandi et al. \(2024\)](#) in their work on **RoCo**. They consider a scenario where robotic arms collaborate in a shared workspace on a common task with incomplete knowledge, communicating through dialogue. Prompted with the *task context*, *round history*, *agent capability*, *communication instructions*, *current state observations*, and *plan feedback*, the robots’ LLMs discuss until a final summary for the next step is generated. This summary contains goal positions for each robot, which are then fed into a centralized motion planner. Unlike previous methods that directly output goal positions, RoCo’s LLMs discuss sub-tasks for “common-sense objects” (e.g., picking up a green block) that are deemed semantically simple for LLMs. These objects are detected and described by a separate pre-trained object detection model, which is noted as the primary bottleneck. RoCo was evaluated on a custom benchmark dataset of six tasks in simulation. Comparisons with a centralized LLM planner with full knowledge show cases of comparable performance and instances where the LLM agents devise a collaborative strategy that the central planner does not. However, there are also cases where the central LLM performs better than RoCo. Additionally, the authors demonstrate that an LLM agent can be replaced by a human, enabling human-robot collaboration via language. After evaluation, they conclude that RoCo shows promising generalization, though with limitations and room for improvement.

While LLMs have been applied to swarm robotics specifically, the approaches discussed here do not try to fine-tune a LLM on global level descriptions and individual level robot controllers. Especially, they do not focus on creating a unified model for different kinds of collective behavior, which is done in this thesis.

**Summary** This literature review demonstrates the scope of previous work on designing robot behavior—from formal specification and macroscopic design to approaches leveraging Large Language Models. Formal modeling improves the understanding and precise control of swarm behavior but requires considerable expert effort. Formal design approaches yield highly performant controllers after scenario-specific optimization, yet these controllers do not necessarily generalize to other tasks, necessitating repeated investment in fitness function design and resources and simulation environments. Swarm languages allow for fine-grained specification of macroscopic goals and microscopic behavior while reducing the algorithm design workload through decomposition of global descriptions. However, a generally usable framework for real-world controllers remains elusive since the link between elementary behaviors still must be manually established.

Enabled by the generalization and reasoning capabilities of pre-trained LLMs, these models have been applied to single-robot systems to control robots and generate controllers through natural language specification. Limited work has been done in multi-robot settings, with most approaches focusing on prompt engineering and micro-level dialog-based cooperation, while achieving local swarm behavior with

LLM agents remains challenging. Nevertheless, the generalization abilities of LLMs might eventually allow them to serve as a general translator from macroscopic instructions to microscopic swarm controllers for arbitrary tasks. To the best of my knowledge, no unified LLM framework currently exists that accomplishes or even attempts this kind of translation.

## 3 Experiments Pipeline

This section gives an overview of the pipeline implemented for the experiments. The pipeline is organized in the following general structure:

1. Define a set of swarm robot missions with parameters that can vary to create a dataset for finetuning.
2. Generate swarm controller for each sample.
3. LLM training like supervised finetuning on natural language description and generated swarm controller.

The section is organized in these three parts. The method to create a dataset for fine-tuning as described in Section 3.1 was conceived during my preparatory masters project (Jandeleit, 2023). In the scope of my thesis, the method was adapted to integrate with AutoMoDe-Maple (Kuckling et al., 2018). AutoMode was executed to work on the missions created for the dataset as described in Section 3.2. Finally, a pre-trained LLM is fine-tuned on that dataset, which is described in Section 3.3.

### 3.1 Dataset Collection

A dataset of mission descriptions and suitable controllers is necessary to finetune a LLM on generating swarm behavior.

To avoid overfitting during fine-tuning, a sufficient dataset size is required. Such a dataset can be acquired by first compiling a set of mission attributes that vary in certain parameters across missions. Then, a specific mission scenario can be sampled from these attributes. Assuming there is a function that transforms these attributes into a configuration for a collective behavior design software such as AutoMoDe, a robot controller and respective performance score can be obtained. Assuming there is a second (probabilistic) function that converts a scenario to a natural language description, you get a correspondence from formal descriptions to behavior tree, its performance score and textual description.

If the different mission attributes are independent, the number of configurations multiply. And the total amount of different datapoints that can be generated by using this method of structural variation of parameters can thus be counted by

$$N_{\text{configurations}} = N_{\text{possible values of each attribute}}^{N_{\text{number of attributes}}}$$

The same holds for the number of different descriptions that can be generated from a single scenario. So the total variation of the dataset can be described as

$$N_{\text{descriptions}} = N_{\text{possible values of each attribute}}^{N_{\text{number of attributes}}} \cdot N_{\text{possible text descriptions of each attribute}}^{N_{\text{number of attributes}}}$$

However, do note that the variation regarding all possible descriptions might be of a different quality, because the different descriptions do not alter the inherent kinds of missions in the dataset, which additionally might be of redundant nature if datapoints overlap to much.

This approach has already been implemented in my preparatory master's project *Generating a Dataset of Swarm Mission Configurations for Training Large Language Models* (Jandeleit, 2023).

To the best of my knowledge, there currently is no other public dataset of swarm missions and controllers, especially for AutoMoDe and the simulator used by AutoMoDe, ARGoS3 specifically. In the following, a description of the aforementioned method is given.

### — 3.1.1 Mission Elements Data Model

A mission is described as a recursive data structure of *mission elements* which are formally defined below.

Let  $ME$  be a variable. I define  $ME$  as a *mission element* if it has the following properties:

- A tuple  $P$  of parameters. These are all variables necessary to fully specify  $ME$ .
- A function **sample** that randomly assigns reasonable values to  $P$  and  $C$  according to a sample space, thus creating an *instance*  $ME$ .
- A function **describe** that transforms  $P$  and  $C$  to return a *set* of natural language descriptions for  $ME$ .
- A function **configure** that transforms  $P$  and  $C$  to a controller design software description (XML in this case of AutoMoDe).
- A tuple of  $C$  of *mission elements* as children. The functions **describe** and **configure** are required to independently aggregate their children's respective function if  $C$  is not empty. That makes these functions purely aggregative in nature. On the contrary, for leaf nodes without children, the functions' values purely rely on the parameters.
- A type  $T$ .
- A tuple  $C^t$  of children types. The length must be the same as  $C$  and for each  $me_i$  and type  $t$  of  $me_i \in C$  it must hold that  $t = C_i^t$  when  $i$  is the index identifying that child.

This model allows a tree of more and more specific mission elements that can be combined independently. The independent aggregation requirement is necessary to achieve a high number of different datapoints. In practice, this model is implemented as two layers as shown in Figure 3. The root node aggregates its children to a complete mission. Its children are leaf nodes without own children. The different types the root node's children can have are the different parts a mission can consist of, described as *mission attributes* in Section 3.1. The parameters  $P$  of a *mission element* captures how much each specific *mission attribute* can vary.

Inspired by the swarm modeling language SML (Bozhinoski and Birattari, 2022), missions are split into independent parts, representing the *mission attributes* mentioned above. Their model is adapted such that only parts suitable for the underlying controller design software, which is AutoMoDe-Maple in this case, are used. Implemented *mission attributes* are:

- The **arena** where the mission takes place. It defines the size of the area, as well as the walls surrounding it.
- The **lights** placed inside the arena.
- The robots as a **swarm** placed inside the arena.
- The **objective** as the desired behavior the swarm should perform.

So every mission consists of 4 children, sampled independently. Nevertheless, there are multiple kinds of *mission elements* possible for each kind of attribute. In fact, there are multiple kinds of objectives implemented like *Aggregation* and *Foraging*.

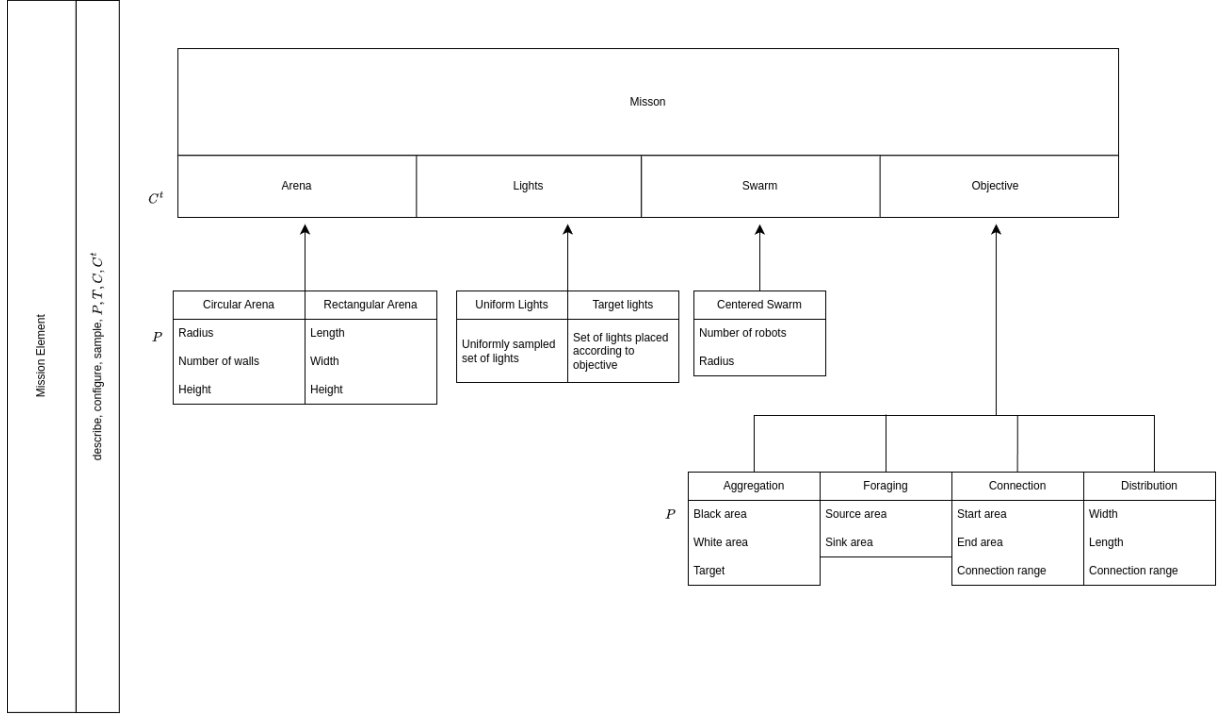


Figure 3: This figure illustrates how the generated dataset is made up of mission elements. It highlights the most relevant children types also called *mission attributes* and variable parameters.

For the experiments I utilize this previously described method to create a dataset for finetuning. Further, I extend the existing implementation by fixing bugs and extend it to not only provide randomly sampled environments but also guided implementations, where lamps are placed according to the objective. More details about the implementation can be found in the following Section 3.1.2.

### — 3.1.2 Scenario Specification

The swarm controllers for finetuning are created using AutoMoDe by Kuckling et al. (2018). It uses ARGoS (Pincioli et al., 2012) as its simulator. So the dataset of missions is designed with the abilities of that method in mind.

The dataset is obtained using the method created in the preparatory project by Jandeleit (2023). It is described in the previous Section 3.1.1 This section describes mission elements for several *mission attributes* that can be defined and sampled individually. Figure 3 contains a visualization of the implemented elements. The following mission elements are defined:

**The arena attribute** constrains the environment and represents the walls surrounding it, constraining the movement of the robots:

- **Circular arena:** A *radius*, *number of walls* and *height* is sampled. The arena consists of straight wall elements of the specified number, arranged in a circular shape to form a circle of a given radius. The walls have a certain height but it is a leftover and not used here. During the sampling process, the radius is sampled uniformly within 1.0 and 5.0 meters, and the number of walls within 3 and

25 pieces. The available space inside the arena for use in other elements is considered to be its inscribed square.

- **Rectangular arena:** The *length*, *width* and *height* are specified. The arena will be a rectangle of 4 walls placed to form a rectangle of a given shape. The height is not used in the experiments either. During sampling, length and width are independently sampled uniformly within 1.0 and 3.0 meters.

The **swarm attribute** describes how and where the robots are initially placed. A single element mission element, **centered swarm**, is defined. Robots are sampled within a circle of given *radius* around the center of the arena. The *number of robots* parameter specifies how many robots are sampled and ranges from 5 to 25. The radius for the swarm is sampled uniformly from the arenas radius within  $[\max(0.25, \frac{\text{radius}}{2}), \text{radius}]$ . Note, that the actual placement of the robots is not specified during sampling. New placements are generated each simulator run within the given constraints.

There are two different kinds of mission **objectives** specified. Two have been chosen because it has been shown that AutoMoDe-Maple is able to generate controllers for them (Kuckling et al., 2018):

- **Aggregation** is specified by defining two *ground areas* and defining which one of them is the aggregation target. The ground areas are circles of different size and a certain color (*black* or *white*). They are distributed uniformly inside the arena and their sizes range from 0.25 to half the size of the arena. The colors of the two circles are sampled exclusively so that there is always a circle of each color present. The goal of this mission is that all robots meet at the specified circle.
- **Foraging** is specified by two circles as with the aggregation objective. However, the circles are now sampled with a maximum of  $\frac{1}{4}$  of the arena's size. One circle is assigned to be the *source* circle and the other is the *sink* circle. The goal of this mission is to bring objects from source to target, also called sink in the following text.

To increase the diversity in the dataset and evaluate the performance on new missions, two previously untested missions in AutoMoDe are specified:

- **Connection** is also specified by two ground circles the same as in aggregation. One area is the *start circle* and the other is the *end circle*. Additionally, a *connection range* is sampled within  $[0.05, 0.5]$ . The robots are supposed to form a line from start to end, keeping a distance between each other according to the connection range.
- **Distribution** is specified by a bounding box rectangle of *width* and *length* of an area. Additionally, a *connection range* is specified. The dimensions are sampled ranging from half of the arena to its full size and the connection range is sampled within  $[0.05, \min(\text{width}, \text{length})]$ . The goal of this objective is that the robots distribute in an area of the specified dimensions, keeping distances in accordance with the connection range.

For the experiments in this thesis, the connection range attributes have been reinterpreted to a sampling grid in the fitness functions. In the experiments it is often distinguished between the different kinds of objectives and called mission type according to the objective for brevity.

Two kinds of **light** elements are implemented:

- **Uniform lights** specifies a *set* of lights. Each light is described by a *position* in *x* and *y* coordinates and an intensity. The number of lights that are sampled for each scenario lies between 0 and 4. The position is sampled uniformly within the arena and the intensities lie within  $[2.0, 8.0]$ .
- **Target lights** is specified in the same way as uniform lights. However, the sampling is different. Here, the lights are not sampled uniformly within the arena, but according to the objective to provide

the robots orientation. This is because the robots only have a local view of the environment and cannot detect the circles, walls, or other robots from afar. The intensities of all lights are sampled within  $[2.0, 12.0]$ . For aggregation, a single light is placed at the target circle. For distribution, four lights are placed at the corners of a rectangle according to a factor of the target area range. The factor is chosen as the reciprocal of a number within  $[1.0, 3.0]$ . For connection and foraging objectives, each circle gets a light.

Note, that there is always another light of a different color (red) with intensity 5.0 at the center of the scene. This was required because the ARGoS simulator always needs a light to be present when the light sensor is enabled. As this is not a restriction of the dataset generation method, it was chosen not to be modeled in the dataset according to encapsulation principles in software design. Instead, that light is hard coded to be present at the simulation stage.

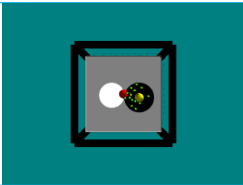
<b>Last Frame</b>	
<b>Description</b>	A rectangular area, with a length of 1.65 meters, width of 1.66 meters, and height of 2.19 meters, is established. A light shows the way to the black area. 16 robots are evenly spaced around the central point, spanning a radius of 0.45 m. The goal is for the robots to aggregate at the black circle. There are two areas on the floor: a circle at $[-0.08, -0.35]$ with a radius of 0.33 meters in black, and another circle at $[-0.03, 0.26]$ with a radius of 0.28 meters in white.
<b>Behavior Tree</b>	<pre>--nroot 3 --nchildroot 4 --n0 0 --nchild0 2 --n00 6 --c00 5 --p00 0.6627 --n01 5 --a01 3 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 0 --p10 0.9127 --n11 5 --a11 4 --att11 4.3675 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 5 --p20 0.6156 --n21 5 --a21 2 --p21 0 --n3 0 --nchild3 2 --n30 6 --c30 5 --p30 0.0786 --n31 5 --a31 1 --p31 0</pre>

Table 1: This table shows an example scenario as depicted in the dataset. The description describes the sampled scenario. The scenario data itself is converted so that the depicted behavior tree can be generated by AutoMoDe. The image shows the last frame after running this controller with ARGoS. A full example that also includes the ARGoS configuration is given in the appendix.

The fields every mission element is specified by can be converted to an XML representation. That XML representation can be converted to a configuration for the ARGoS simulator. A sampled instance of elements of all types can now be called a scenario. Each variant of a mission attribute is selected with the same probability during sampling. An example with a complete depiction of an ARGoS configuration can be found in the appendix.

Every mission element gets assigned an associated natural language description during sampling. There are 10 different templates implemented for each element. The templates are natural language descriptions of the respective element and contain placeholders for its fields. Not all templates use all of the element's fields. This implies that the natural language description can be incomplete. The target lights element has 5 descriptions that are independent and 5 descriptions targeted to the objective. The sampled descriptions get concatenated in the fixed order: arena, lights, swarm, and objective to obtain a description for the overall mission scenario. Table 1 above shows an exemplary description.

The implementation is done in Python and is available<sup>1</sup> as open source under the MIT license as reference and for further work.

The mission elements have already been implemented in Jandeleit (2023). However, the target lights

<sup>1</sup>The code from this thesis is available at <https://github.com/StudentWorkCPS/behavior-tree-llm>.



element has been added in the scope of this thesis. Also, the implementation of all elements has been optimized further and errors found have been removed.

## 3.2 Reference Controller Design

This section describes how the reference controllers for the sampled dataset are obtained. In contrast large parts of the previous Section 3.1, everything has been implemented in the scope of this thesis again.

### — 3.2.1 Fitness Optimization with AutoMoDe

The robot controllers for each scenario are generated by AutoMoDe-Maple in the form of behavior trees. They can be passed to the ARGoS simulator as command line arguments. This already is a text-based format as used by conventional LLMs. Thus, this is the representation used in this thesis. An example can be found in Table 1.

For the optimization process, a fitness function that measures a controller’s performance needs to be defined and implemented into the C++ runtime environment. They are defined such that higher values imply better performance. The functions are implemented into a library that is referenced in the ARGoS configuration. As there are four different kinds of missions, a custom fitness function needs to be specified for each objective indepentently.

The fitness function for **Foraging** is implemented as in Kuckling et al. (2018). To simulate robots picking up and releasing items, each robot is assigned an internal state indicating whether a robot is carrying an item. Once the robot enters the source area and is not yet carrying anything, it is marked as having picked up an item. Similarly, when entering the target area with an item, it gets marked as released. Let  $N$  be the number of robots in the scene and  $c_i$  the number of released items for robot  $i$ , then

$$F_{\text{foraging}} = \sum_{i=1}^N c_i$$

is the fitness function for a foraging mission.

The fitness function for **Aggregation** is slightly different from the definition in Kuckling et al. (2018) because the mission objective is defined with a slight variation. Instead of two areas of the same color, the aggregation mission is defined here with two areas of distinct colors. Thus, if  $t_i = 1$  indicates that robot  $i$  is on the target area and  $t_i = 0$  otherwise, the fitness function for an aggregation mission is defined as

$$F_{\text{aggregation}} = \frac{\sum_{i=1}^N t_i}{N}.$$

The fitness functions for the remaining mission types have been derived from scratch. For **Connection**, the line connecting both ground circle centers is computed. Points on that line are sampled with a distance of the specified connection range. Each robot  $i$  is assigned its closest sample point  $p_i$  and  $d_i$  denotes the euclidean distance to that point. If  $nb(p)$  is an indicator function that is one if more than

one robot is assigned to the same sample point  $p$ , then

$$F_{\text{connection}} = -0.1 \sum_{i=1}^N nb(p_i) - \sum_{i=1}^N d_i$$

is the fitness function for a connection mission. The intuition is that the robots should be on the line connecting both circles, so the total distance between the sample points should be minimized. At the same time, the robots should physically connect both ground circles by distributing equally on that line, so the number of robots sharing the same closest line point is to be minimized as well. The weighting factor of 0.1 was determined by visual inspection of the results.

The fitness function for **Distribution** is defined in an analogous way. A rectangle of the specified shape is computed around the center of the robot swarm. In this area, points with a mutual distance according to the specified connection range are sampled. Again, the robots are assigned to their closest sampled point. The fitness function for a distribution mission is thus defined as

$$F_{\text{distribution}} = - \sum_{i=1}^N nb(p_i) - \sum_{i=1}^N d_i,$$

differing from  $F_{\text{connection}}$  only by the different sampling strategy according to the specified area and the absence of the weighting factor that was not deemed necessary here. The intuition is that the robots should spread out equally within the specified area. Note, that the area is not defined absolutely with respect to the environment but by the current swarm center. Thus, the robots can decide where to distribute and, in theory, keep moving, similar to flocking, as long as the overall area occupied by the swarm stays the same as defined in the respective scenario.

The fitness functions and necessary computations have been implemented in the scope of this thesis. Several variants have been tried in an educated guessing approach and these have been chosen because they yield reasonable results with AutoMoDe-Maple in an optimization time adequate enough to produce behavior trees in a higher quantity. Behavior trees for 2700 distinct scenarios are generated. 2564 of them resulted in usable controllers. The others are missing because AutoMoDe crashed and did not return a controller in some cases. AutoMoDe was executed with the following parameters: a maximum experiment budget of 40000 and a maximum of 50 iterations.  $\mu$  is set to 25. Each run simulates the environment for 1200 timesteps.

### — 3.2.2 Implementation Details

The behavior trees are generated on the hardware of the high performance computing (HPC) system BwUniCluster2.0<sup>2</sup>. Tooling for this has been developed in the scope of this thesis. The installation of AutoMoDe itself is isolated with the use of containerization. They are specified with Docker, automatically building and installing the additional library for the fitness functions (called “LoopFunctions” by ARGoS). BwUniCluster2.0 does not support Docker (Merkel, 2014), but uses Enroot (NVIDIA, 2025) instead. Before uploading, the Docker image is exported as an Enroot image.

Because of the shared nature of the BwUniCluster2.0, processing tasks are not run directly, but submitted as a script to a workload manager, Slurm (Yoo et al., 2003) in this case. A collection of individual scenarios and respective AutoMoDe runs is combined together and submitted as a Slurm job. This means that within the same job, multiple AutoMoDe optimizations are run in sequence. As there is

<sup>2</sup>The author acknowledges support by the state of Baden-Württemberg through bwHPC.

a maximum number of jobs that can be submitted per user, the behavior tree generation is split into batches of typically 300 or 600 scenarios at a time. These are further distributed to 50 Slurm jobs. The required scripts for the Slurm jobs are generated from the sampled scenarios using meta-programming and templating in Python. Slurm requires resource requirements and estimations to be set in the script. The number of CPUs is set to 8, the memory to 2500 MB. The estimated wall clock running time for the complete Slurm script is computed by the number of scenarios per script and by assuming an average of 150 minutes per AutoMoDe run, with an additional 5 minutes to install the AutoMoDe Enroot container. These values have been determined by trial and error. AutoMoDe needs to be installed for each job independently because Slurm isolates its computations from the environment. That is the case when the environment variables `ENROOT_DATA_PATH`, `ENROOT_CACHE_PATH` and `ENROOT_RUNTIME_PATH` are set manually to a directory local to the Slurm job and exported. Initial trials with a shared installation resulted in almost all Slurm jobs crashing as well as warnings because of a high number of reading accesses in the non-optimized shared filesystem. All Slurm tasks are then submitted and run in parallel. Notably, a CPU number larger than 8 caused frequent crashes during controller generation. With this setup, behavior trees for 600 scenarios are generated in typically about 16 hours. This corresponds to 5 or more days of total CPU time. For the final dataset used in the experiments, behavior trees for a total of 2700 scenarios are generated.

In each optimization, AutoMoDe-Maple (Kuckling et al., 2018) is started by invoking `irace` with the parallel 16 option. AutoMoDe prints the final behavior tree to the console. This output is captured and parsed for the tree in a bash script. The script writes the scenario with its index and controller to a shared file across all Slurm jobs. That file represents the output of the behavior tree generation step and is downloaded to a local computer for further processing. After downloading the generated controllers from the HPC cluster, they are merged with the original dataset containing all the scenario information, such as the natural language description. For evaluation, fitness scores are computed for each scenario. This is done using the same Docker image that is transferred to the cluster. However, instead of optimizing a fitting behavior tree, the behavior tree now is evaluated only. The performance is based on the fitness scores it achieves during several evaluation runs. Ten executions on each scenario are performed and the average fitness is added to the dataset. Evaluating a dataset of 600 scenarios takes about 100 minutes on consumer-grade hardware.

### 3.3 LLM Training

Based on the scenario dataset and the AutoMoDe behavior trees, a large language model is fine-tuned to produce behavior tree swarm controllers on its own. The input is the natural language description of the scenario, concatenated with the instruction

*Generate the behavior tree that achieves the objective of this mission.*

on a new line. The output is the behavior tree in its command line argument version as used by AutoMoDe for ARGoS. After training with the fixed instruction in the final line, the instruction is also required during inference. Without this line, the trained model was found to return a natural language response instead of the command line behavior tree it was trained with. An example of a natural language description and output behavior tree is shown in Table 1

The pretrained base model that is used for fine-tuning is Mathstral in version 0.1 with 7 billion parameters. It is a version of the Mistral 7B model (Jiang et al., 2023) fine-tuned on math and instruction following. It was chosen because it is available as open source under the Apache license and for its good performance given its relatively small size. The authors claim it outperforms a 13 billion parameter Llama 2 model

and the Mistral model was already successfully applied to robotics by [Ao et al. \(2024\)](#). Because 7 billion parameters is a large model still, dedicated hardware is required. Using quantization methods ([Dettmers et al., 2021](#)), the model is loaded on the NVIDIA Titan RTX GPU with 24GB of VRAM.

For interacting with the LLM during the experiments, a pipeline is developed that abstracts most of the complexity involved based on the primary applications in this thesis. The framework is written in Python and builds on top of the Huggingface ([Wolf, 2020](#)) framework for transformer-based models. The pipeline supports supervised fine-tuning with techniques for parameter efficient fine-tuning (PEFT) ([Houlsby et al., 2019](#), [Mangrulkar et al., 2022](#)) like QLoRA ([Dettmers et al., 2024](#)) reinforcement learning using DPO ([Rafailov et al., 2024](#)) and model inference. The pipeline aids in the consistent and correct handling of the LLM models, including loading models from file. Due to quantization and QLoRA, which only trains adapters instead of the full model, the fine-tuned model must be loaded correctly. Even slight changes or oversights can cause the loaded model to produce random output. This problem was solved by creating the unified pipeline for the experiments.

During supervised fine-tuning, 20% of the dataset is used for validation. The training is conducted for 12 epochs with a learning rate of  $2 \cdot 10^{-5}$  and a cosine learning rate scheduler. For memory reasons, the batch size had to be reduced to 1 scenario per training batch. Reinforcement learning with DPO (refer to Section 2.3.3) was performed with similar settings. All training parameters can be found in the appendix. Initially, bracketing techniques (e.g., instructing the LLM to enclose the final controller inside “—BTSTART—” and “—BTEND—”) were attempted, but that resulted in inconsistent output with the model including several tags and repeating itself endlessly. After the appropriate combination of special tokens in the model’s tokenizer, training parameters, and dataset size was found, that issue was resolved and the tags were deemed unnecessary. An example of that can be found in the appendix.

After training, the fine-tuned model is evaluated similarly to the AutoMoDe controllers. For a set of scenarios, depending on the conducted experiment, the model is prompted to generate a controller for a scenario using the same prompt structure as used during training. No special inference technique is used by default. The output is then passed into ARGoS and executed 10 times, with the average fitness reported as the controller’s performance.

## 4 Experiments and Results

This section contains the experiments done to evaluate the viability of LLMs for generation of behavior trees for swarm robot missions.

### 4.1 AutoMoDe Baseline

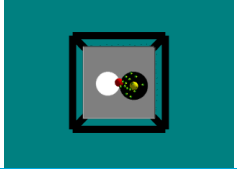
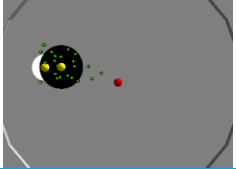
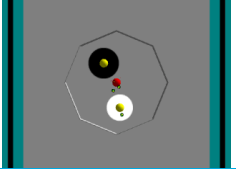
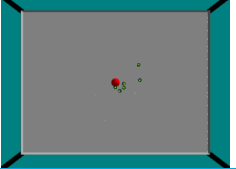


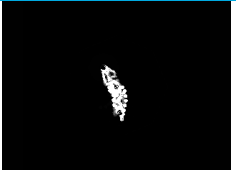
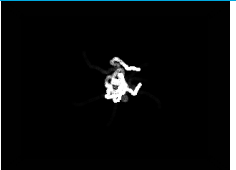
	Aggregation	Foraging	Connection	Distribution
Last Frame				
Trace				
Description	A rectangular area, with a length of 1.65 meters, width of 1.66 meters, and height of 2.19 meters, is established. A light shows the way to the black area. 16 robots are evenly spaced around the central point, spanning a radius of 0.45 m. The goal is for the robots to aggregate at the black circle. There are two areas on the floor: a circle at [-0.08, -0.35] with a radius of 0.33 meters in black, and another circle at [-0.03, 0.26] with a radius of 0.28 meters in white.	The circular arena, having a radius of 2.90 meters, is constructed with 18 walls. The first light is located at (0.36, 1.33), guiding foragers, while the second light is at (0.35, 1.71), providing additional visibility for the other circle. Placed within a 1.91-meter radius around the center are 24 robots. Two circles are present—one at [0.36, 1.33] with a radius of 0.51 meters, colored in black, and another at [0.35, 1.71] with a radius of 0.31 meters in white. The robots' task is to transport items from the white starting area to the black circle.	The environment features a circle composed of 8 walls. A light is positioned at the target circle. Another light is positioned at the end. Evenly positioned around the origin are 5 robots within a radius of 0.76 meters. The objective for the robots is to connect both circles from white to black, maintaining a distance just under 0.31 m. There are two circles on the floor—one at [0.45, 0.30] with a radius of 0.36 meters, colored in black, and another at [-0.59, -0.08] with a radius of 0.31 meters in white.	The area is a rectangle with dimensions 4.36 x 3.29 x 2.13. 0 lights are distributed uniformly in the arena. There are 7 robots placed uniformly around the center within a radius of 1.21 meters. The objective for the swarm is to cover an area of 1.61 by 0.95, while staying connected to each other. The swarm counts as connected if every robot is transitively connected to each other robot in the swarm. Two robots are connected if their distance is at or below 0.11 m.
Behavior Tree	--nroot 3 --nchildroot 3 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.6193 --n01 5 --a01 0 --rwm01 3 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 2 --p10 0.6412 --n11 5 --a11 4 --att11 3.7667 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 3 --p20 6 --w20 17.8635 --n21 5 --a21 4 --att21 2.7507 --p21 0	--nroot 3 --nchildroot 3 --n0 0 --nchild0 2 --n00 6 --c00 0 --p00 0.7682 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 4 --p10 2 --w10 13.9819 --n11 5 --a11 5 --rwm11 4.8951 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 1 --p20 0.1116 --n21 5 --a21 0 --rwm21 1 --p21 0	--nroot 3 --nchildroot 1 --n0 0 --nchild0 2 --n00 6 --c00 0 --p00 0.7669 --n01 5 --a01 2 --p01 0	--nroot 3 --nchildroot 3 --n0 0 --nchild0 2 --n00 6 --c00 5 --p00 0.9318 --n01 5 --a01 4 --att01 2.3654 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.1776 --n11 5 --a11 1 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 5 --p20 0.2977 --n21 5 --a21 2 --p21 0

Table 2: This table shows the best scores achieved by AutoMoDe for the evaluation dataset of the main experiment. First, the last state is displayed after 1200 simulation steps. Then, a trace is displayed, highlighting points where changes occur between the last frames of the video. They are highlighted by exponential decay so that the most recent frame has the highest opacity. Then, the natural language description is displayed (although not used by AutoMoDe), as well as the AutoMoDe generated behavior trees. Line breaks are for visualization reasons only and not part of the behavior tree.

Table 2 shows an example for generated behavior trees from AutoMoDe and the final state of the simulation environment after execution. The performance of each generated controller for its scenario is estimated individually after training. During evaluation, the simulation is run repeatedly and the fitness computed according to the fitness function of the mission as described before. If not stated otherwise, each behavior tree is evaluated 10 times and the average reported as the score of the controller.

Figure 4 shows the distribution of the scores by mission type. Generally for all mission types, a score that is higher (closer to positive infinity) is better. But depending on the mission, the scores that are actually achievable depend on the missions fitness function and are often limited.

From the results it can be seen that the aggregation, distribution and connection missions have unimodal distributions with a peak close to zero. As the scores for those missions (except *Aggregation*) have a negative sign, a peak close to zero indicates good performance. The upper limit is influenced by the optimization process, the environment and the expressiveness of the available elementary behaviors and implemented behavior tree nodes. A non-minimal score close to zero implies a successful optimization process within these constraints. Thus, the AutoMoDe results are taken as the baseline performance for this dataset. This is supported by visual inspection of individual results. Examples of those results can be seen in Table 2.

In general, the robots seem to perform the expected behaviors. For *Aggregation*, the robots are placed at the target area, displaying little movement in the trace image. For *Foraging*, they are moving around the black area while occasionally moving to the white area along three distinct paths to bring items from the white area to the black one. For *Connection*, the robots can be seen in the trace image to be moving between both areas and for *Distribution*, they mostly are moving inside a limited area.

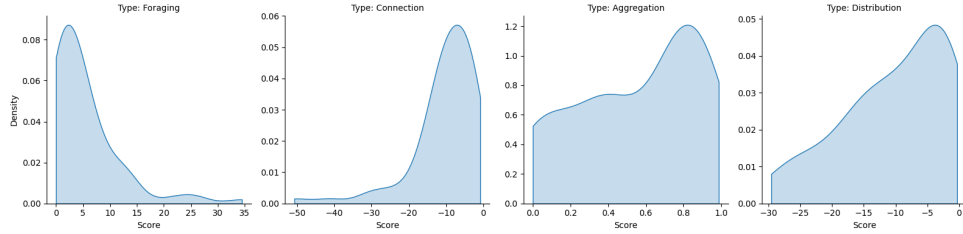


Figure 4: This figure shows the fitness score distribution achieved by AutoMoDe for the validation dataset of the main experiment according to each mission type. The corresponding absolute maximum fitness for each objective are 34.50 for *Foraging*,  $-0.80$  for *Connection*, 0.99 for *Aggregation* and  $-0.30$  for *Distribution*. It can be seen that the density has a maximum on the upper side of the achieved fitness for all mission types except *Foraging*.

The intuitive definition of  $F_{\text{aggregation}}$  allows for direct interpretation of its scores as the percentage of successfully aggregating robots. Notably, their distribution stretches over the range of all possible values from 0 to almost 1. This means that there are cases where optimal fitness has been achieved and all robots aggregated at the same circle but also that there are cases where no robot is at the target circle. Specifically, there is a low but non-zero amount of behavior trees at the score 0 directly. The performance depends not only on the generated controller alone but the environment as well. As AutoMoDe generally seems to optimize the scores, it can be assumed that the scores near 0 result from the environmental constraints rather than AutoMoDe. This can be seen in qualitative examples in Table 3, where circles are placed so far away that it is unreasonable for the robots to aggregate there in the limited execution time. Interestingly, AutoMoDe does not seem to systematically explore all of the environment in some cases and can be distracted by badly placed lights. Also, remember that the location of the ground circles are not known to the robots and need to be discovered first.

The scores for the foraging mission display a slightly bimodal distribution. Most scores are in the single digit range, but distinct from zero. This indicates that some, but not very many items have been transported in most cases. This can be attributed to the environmental constraints as discussed for the aggregation mission. Furthermore, there is a small second peak around the 25 score mark, implying that some environments are very suitable for foraging. This can be seen in Table 2 showing the best AutoMoDe results. In the foraging mission, circles are placed close to each other with lights showing the way. This kind of environment seems to be suitable for high performing foraging controllers.

	Scenario 109	Scenario 187	Scenario 222
Last Frame			
Trace			
Description	The arena has a radius of 4.87 m. In the arena, 4 lights are evenly spread out with intensities 3.93, 2.95, 6.18, 7.09. Uniformly distributed are 25 robots within a radius of 3.31 meters. There is a circle at [-0.36, -2.51] with a radius of 1.09 meters in black, and another circle at [-2.44, 3.15] with a radius of 0.68 meters in white.	A rectangular area, with a length of 3.47 meters, width of 7.06 meters, and height of 2.40 meters, is established. There are the following lights in the arena: ((-0.65, -0.63), (0.97, -0.11), (1.46, -1.18)). 20 robots are evenly placed around the center, covering a radius of 1.66 meters. The objective is for the robots to aggregate at the black circle. There are two designated areas on the floor: a circle at [0.45, 0.99] with a radius of 0.87 meters in white, and another circle at [2.55, -0.02] with a radius of 0.66 meters in black.	The arena has a radius of 4.14 m. 0 lights are distributed uniformly in the arena. 24 robots are evenly distributed around the origin within a radius of 2.21 m. The primary goal for the robots is to cluster at the black circle. There are two designated areas on the floor: a circle at [2.82, 2.29] with a radius of 0.39 meters in black, and another circle at [-2.86, 0.53] with a radius of 0.66 meters in white.
Behavior Tree	<pre>--nroot 3 --nchildroot 4 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.7582 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.5365 --n11 5 --a11 4 --att11 2.7074 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 0 --p20 0.8473 --n21 5 --a21 0 --rwm21 5 --p21 0 --n3 0 --nchild3 2 --n30 6 --c30 1 --p30 0.8541 --n31 5 --a31 3 --p31 0</pre>	<pre>--nroot 3 --nchildroot 4 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.1561 --n01 5 --a01 0 --rwm01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.6041 --n11 5 --a11 3 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 0 --p20 0.803 --n21 5 --a21 4 --att21 3.033 --p21 0 --n3 0 --nchild3 2 --n30 6 --c30 5 --p30 0.3648 --n31 5 --a31 3 --p31 0</pre>	<pre>--nroot 3 --nchildroot 4 --n0 0 --nchild0 2 --n00 6 --c00 5 --p00 0.3602 --n01 5 --a01 1 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 0 --p10 0.4497 --n11 5 --a11 2 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 5 --p20 0.4682 --n21 5 --a21 4 --att21 4.0885 --p21 0 --n3 0 --nchild3 2 --n30 6 --c30 4 --p30 9 --w30 18.3516 --n31 5 --a31 3 --p31 0</pre>

Table 3: This figure shows examples in the evaluation dataset of the main experiment, where AutoMoDe was not able to generate good performing behavior trees for *Aggregation* missions and does not properly explore the environment. The scenarios are displayed like in Table 2. The arenas are displayed zoomed out to fit the frame, their sized don’t necessarily need to be comparable.

## 4.2 LLM Experiments

Several experiments get executed to evaluate the performance of LLM-based swarm controller generators. They vary in training but all use supervised fine-tuning on some subset of the AutoMoDe controllers. The exception is the DPO experiment, where a reinforcement learning approach is taken to evaluate a more direct way of communicating the rewards to the LLM. The first experiment is the main experiment, evaluating the overall performance while the others are conducted to investigate special aspects of the training process.

### — 4.2.1 Main Experiment

In this experiment, the full dataset of available scenarios with controllers is used for joined supervised fine-tuning. To evaluate the impact of the dataset size, the dataset is split into equal sizes. Ten partitions of size 255 are formed. Another eleventh partition of size 250 is kept for evaluating the final model. From these partitions, training datasets of increasing size get formed. The base LLM gets fine-tuned on each of these partitions from scratch according to the SFT pipeline and its execution details as discussed in

Section 3.3. After training, each fine-tuned model gets evaluated on the test dataset. Figure 5 shows the achieved scores by dataset size.

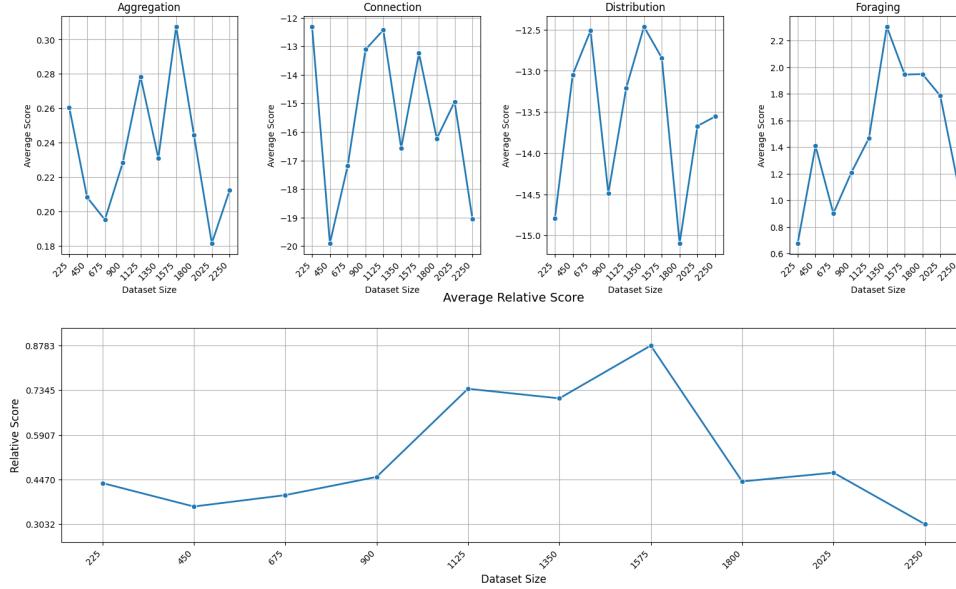


Figure 5: This figure shows the impact of the training dataset size on the scores achieved by the fine-tuned model. The upper 4 subfigures show the scores by mission type, while the bottom figure shows an aggregated score of all types. The aggregate score is obtained by first min-max scaling the individual mission scores because they have different scales. Then, the aggregated average relative score is the mean of the min-max scaled scores for each dataset size. It can be seen that the different missions are learned in a different way. For example, *Distribution* almost meanders between two extremes until a high dataset size is reached while *Foraging* shows a clear upward trend with one setback and is followed by a continuous decline. From the average it can be seen that the overall performance improves until a dataset size of 1575 rows is reached. There an average of about 88% of the individual top performances is achieved.

It can be seen that the fine-tuned LLM has learned the structure of the command line behavior tree representation and is able to produce controllers whose performance increases with dataset size. That these behavior trees are actually working swarm controllers that result in meaningful swarm behavior, can be seen in Table 4. From these qualitative examples it can be seen that the fine-tuned LLM can produce qualitatively similar swarm behaviors similar to AutoMoDe. However, while the scores for *Distribution* missions can compete with AutoMoDe, it can be questioned if the behavior shown is actually desired. After 1575 rows, the scores do not improve consistently anymore and overall performance starts to decline again. Interestingly, there is a lot of fluctuation in the individual missions results but not so much in the overall average. Potentially, the samples added with increasing dataset size, lead to a different local optimum because of different initialization. For all other means, the training process is equal. The datasets are sampled uniformly with equal share of each mission type, so there are no such class imbalances in the training sets.

When a behavior tree gets generated that is invalid, it gets detected when passed to the ARGoS simulator, which crashes in these occasions. These crashes are not included in Figure 5 yet. Figure 6 shows the relative number of crashes according to dataset size. The graph shows a similar general behavior to Figure 5. At smaller training sizes, there still is a considerable amount of invalid responses with low failure rates between sizes from 1125 to 1575, and then the failure rate increases again occasionally. However,



in contrast to the achieved scores, the ratio of errors does not go up as strongly and consistently again. And while some mission types like *Aggregation* and *Connection* showed comparatively high fitness scores with small training sets, the high number of invalid responses for smaller training sets shows that larger datasets are required for getting those consistently. In general, both measures agree that a dataset of 1575 entries seems to be a good size in this specific context. Probably, that is the result of a combination of the following factors: Model size, pre-training, fine-tuning environment, task complexity and dataset diversity.

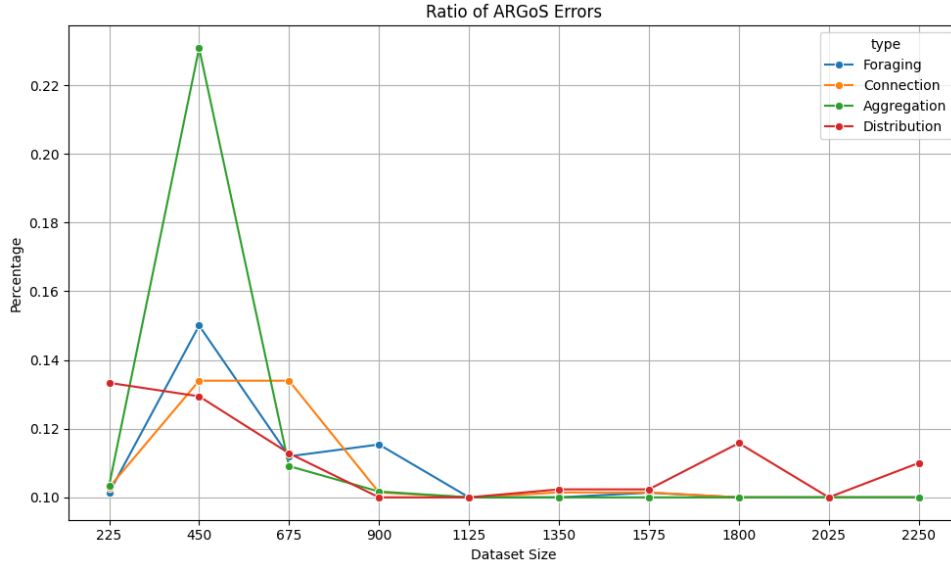


Figure 6: This figure shows the number of ARGoS crashes according to training dataset size and mission type. At lower dataset sizes, the generated controller results in a considerable number of crashes, surpassing 22% of all aggregation missions in its maximum. The number of crashes reaches their overall minimum at 1125 training samples and noticeably begins to fluctuate after the 1575 mark, especially when looking at the *Distribution* missions.

In accordance with the average relative score and ratio of valid responses, the model fine-tuned on the dataset with 1575 entries is considered to be the best result and studied further. A figure showing its loss curve during training can be found in the appendix. Figure 7 shows the performance distributions of the LLM-generated controllers when trained on the 1575 entry dataset in comparison to the AutoMoDe controllers.

In general, the LLM-generated controllers do not match the performance of AutoMoDe. In the *Foraging* case, AutoMoDe is able to achieve scores twice as much as the LLM can. While the LLM is able to achieve top scores for the *Aggregation* mission, it is unable to do so as frequently as AutoMoDe and thus has a significantly worse median.

On the other hand, the fine-tuned LLM has very similar distributions for the *Connection* and *Distribution* missions. Here, a significant difference cannot be claimed, as the 95% confidence intervals for the medians, as displayed by the notches, overlap. Furthermore, it can be seen that the shape of the distributions is similar between LLM and AutoMoDe results in all cases except for *Aggregation*. Even with a large difference between optimal performance like with the *Foraging* missions, the distribution of LLM scores looks like a scaled down version of the AutoMoDe scores. This suggests that the LLM finds fitting behaviors for environments where higher scores are possible but is not able to exploit the environment in such a way AutoMoDe is doing and thus not getting the same high fitness scores.

For the *Aggregation* missions this does not seem to be the case. Here, the LLM performance has two peaks

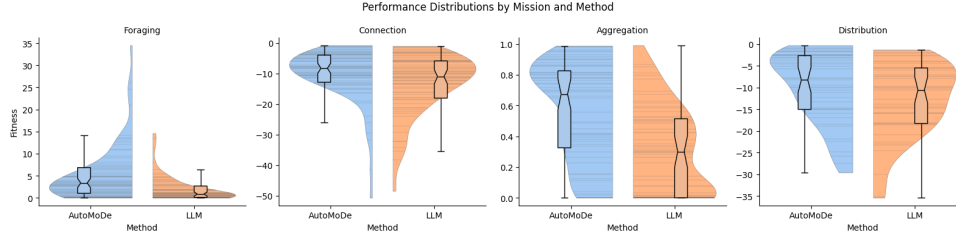


Figure 7: This figure shows the performance of the LLM fine-tuned on 1575 examples in comparison to the AutoMoDe generated baseline. Gray markings in the violin plots indicate performances of individual controllers. The overlaid notched boxplots show important descriptive statistics like the median, 25th and 75th percentile and the 1.5 inter-quartile range. The notches around the median, show its 95% confidence interval. In general, AutoMoDe performs better than the LLM fine-tuned on its results. Two kinds of similarities can be observed. *Connection* and *Distribution* missions show very similar distributions with overlapping notches, while *Foraging* and *Aggregation* show a significant margin between both methods' performances.

with the highest peak near zero, while AutoMoDe has a single peak around a fitness of 0.8. This indicates that there might be a systematic problem for the LLM as it is able to produce considerable results, but in a high number of cases almost achieves no fitness. This observation is supported by the black line around zero, indicating that a very high number of controllers achieved no fitness at an aggregation task. This line is also present for AutoMoDe but not nearly as distinct. So while there are some scenarios where getting a single robot on the target is difficult, it is possible for most scenarios. Especially for the *Connection* missions the LLM seems to have learned one or two dominant strategies, as high scores cluster a lot more around two black lines than for the AutoMoDe controllers.

So overall, the fine-tuned LLM produces successful behavior trees but does not match AutoMoDes performance in most cases.

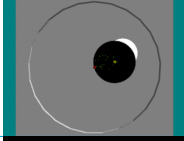
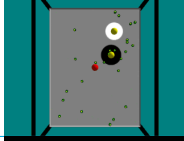

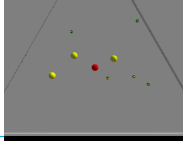
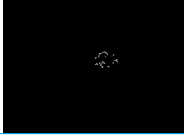



	Aggregation	Foraging	Connection	Distribution
Last Frame				
Trace				
Description	<p>The environment is a circle made out of 25 walls. 1 lights illuminate the space, located at: <math>((0.41, -1.48))</math>. 22 robots are evenly distributed around the origin within a radius of 3.21 m. The robots' task is to aggregate at the black circle. There are two floor areas, each defined by a circle. The first circle, located at <math>[0.41, -1.48]</math>, has a radius of 1.56 meters in black. The second circle, positioned at <math>[1.05, -2.09]</math>, has a radius of 1.09 meters in white.</p>	<p>The environment is constructed as a rectangular space with a length of 2.64 meters, width of 3.43 meters, and height of 2.49 meters. The area features 2 lights, positioned at the following coordinates: <math>((1.07, -0.57), (0.37, -0.48))</math>. Around the central point, 25 robots are positioned uniformly within a 1.28-meter radius. Observe a circle at <math>[1.07, -0.57]</math> with a radius of 0.26 meters, adorned in white, and another at <math>[0.37, -0.48]</math> with a radius of 0.30 meters, characterized by its black hue. The robots' foraging mission is to convey items from the black starting location to the white circle.</p>	<p>The environment features a circle composed of 8 walls. A light is positioned at the target circle. Another light is positioned at the end. Evenly positioned around the origin are 5 robots within a radius of 0.76 meters. The objective for the robots is to connect both circles from white to black, maintaining a distance just under 0.31 m. There are two circles on the floor—one at <math>[0.45, 0.30]</math> with a radius of 0.36 meters, colored in black, and another at <math>[-0.59, -0.08]</math> with a radius of 0.31 meters in white.</p>	<p>The environment is a circle made out of 3 walls. The arena features 3 lights: <math>(0.26, -0.56, 7.72)</math>, <math>(0.35, 0.62, 2.30)</math>, <math>(-0.24, 1.27, 3.38)</math>. 5 robots are evenly spaced around the central point, spanning a radius of 1.96 m. The mission's objective is for the robots to cover an area with a length of 4.77 meters and a width of 0.84 meters, all while staying connected. Two robots are considered connected if their distance is below 0.22 meters.</p>
Behavior Tree	<pre>--nroot 3 --nchildroot 2 --n0 0 --nchild0 2 --n00 6 --c00 0 --p00 0.9995 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.9999 --n11 5 --a11 1 --p11 0</pre>	<pre>--nroot 3 --nchildroot 2 --n0 0 --nchild0 2 --n00 6 --c00 0 --p00 0.9995 --n01 5 --a01 0 --rwm01 1 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 2 --p10 0.9999 --n11 5 --a11 2 --p11 0</pre>	<pre>--nroot 3 --nchildroot 2 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.9995 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 5 --p10 0.9999 --n11 5 --a11 4 --att11 4.6862 --p11 0</pre>	<pre>--nroot 3 --nchildroot 2 --n0 0 --nchild0 2 --n00 6 --c00 1 --p00 0.9995 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 5 --p10 0.9999 --n11 5 --a11 5 --rep11 4.9999 --p11 0</pre>

Table 4: This figure shows the best performing scenarios for each mission type in the evaluation dataset for the main experiment after training on 1575 scenarios. The format is the same as for the AutoMoDe results. However, this time the LLM generated behavior trees are used and the LLM’s answer is based on the description as displayed here. It can be seen that for *Aggregation*, the robots go to the black circle and move slowly once they are there. For *Foraging*, most robots show exploration behavior by ”reflecting” off of walls while some move between both areas every now and then. Most movement between both circles occurs at the start of the simulation which is less visible in the trace image compared to the end of the simulation. The *Connection* scenario is the same as displayed for the optimal AutoMoDe results, however the approach taken by the LLM is different. This time, a few robots stop at the white circle and only the other robots move between both areas. For *Distribution*, no motion of the robots is observed at all. It seems that for this scenario the initial placement was suitable for distributing the robots according to the target area on average. This approach was taken for AutoMoDe as well for some scenarios but not on this scenario. For the LLM this seems to be the general strategy for these kinds of missions. The scenarios indices in the dataset are 82, 245, 199 and 25 respectively.

#### — 4.2.2 Separation of Target Color

To investigate why the LLM produces a distribution for *Aggregation* missions that is bi-modal, a dedicated experiment is performed. Only the aggregation missions are taken from the original dataset and separated into distinct sets by color of the target area. This results in two datasets of size 450 for the target colors black and white each. A third control dataset of the same size is sampled, where black and white target colors are mixed uniformly.

The pre-trained base LLM gets fine-tuned on each of those datasets individually and the resulting models evaluated on their training set. Pre-trained refers to the original model weights and not the fine-tuning done in the main experiment. Figure 8 shows the results.

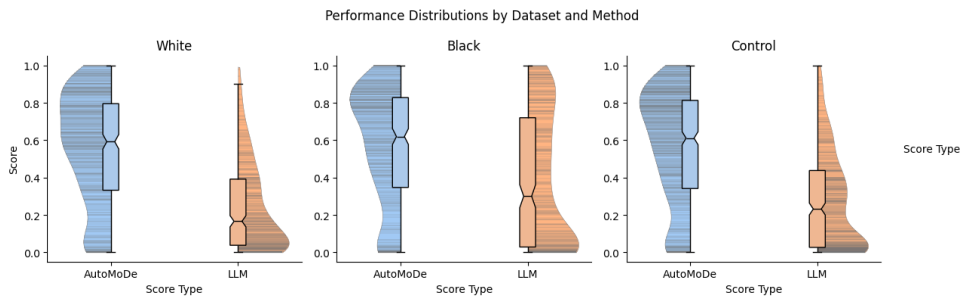


Figure 8: This figure shows the performance distribution after fine-tuning on aggregation only, either on black, white or both target colors. For comparison, the AutoMoDe performance on these scenarios is plotted as a baseline. It can be observed that the LLM has more troubles learning aggregation for white targets than black targets.

It can be seen that the LLM scores after training on black and white aggregation targets individually have different distributions. The LLM for black targets has a peak near zero but a big tail towards 1.0. The LLM for white targets only has the peak close to zero and a little tail. Its fitness scores do range up to 1.0 but become notably less frequent. This seems to translate to the results for the control LLM jointly trained on both target colors. Its fitness distribution has a similar shape to the distribution found in the main experiment in Figure 7 and also has the second peak around the 0.35 mark.

These results lead to two possible conclusions. The first possible conclusion is that aggregating on white spots is inherently more difficult than on black spots. As the ground color is handled by the system as an integer of either 1 or 2, this is unlikely. There is no obvious reason for the robot to prefer one number over the other. A difficulty of distinguishing between both has been falsified by this experiment as the effect is not only present in a combination of both scenarios but also after training on each color individually. The second possible conclusion is that there is a skew in the dataset for white colored targets, that makes it more difficult for the LLM to learn white aggregation targets. For example, during development, there was a bug where the aggregation color was assigned randomly in the natural language description, independent of the actual target, making it impossible for the LLM to decide the target color correctly in some cases. However, this bug has been found and fixed before this dataset was obtained and no similar systematic issue has been found after repeated investigation. Yet, this is the more immediate explanation.

### — 4.2.3 Placement of Environmental Lights

The influence of the environment at the LLM training is studied by independently fine-tuning on a dataset with only uniformly distributed lights and a dataset with lights placed according to the objective as described in Section 3.1.2. For this experiment, the main dataset is split into a uniform, guided and control dataset of size 1250 each. After fine-tuning on these datasets individually, each fine-tuned model gets evaluated on the dataset used for training. The results can be seen in Figure 9.

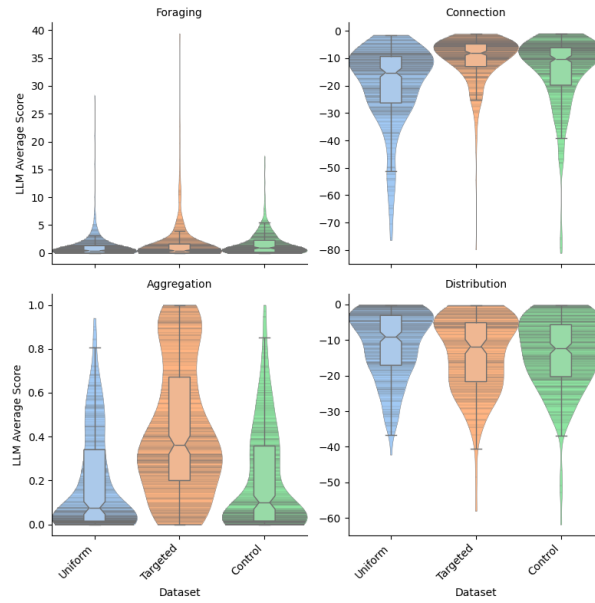


Figure 9: This figure shows the fitness score distributions according to the placement of lights in the environment and mission type. The biggest improvement of placing lights in the environment can be seen for aggregation type missions, while the lowest improvement can be seen for distribution type missions, resulting in a small decrease in median fitness.

The figure shows diverse results. For *Foraging*, guiding lights change the maximum fitness that can be achieved. This makes sense as lights placed on source and sink allow the robots to drive more directly to the desired places, resulting in more efficient foraging. In some cases, this seems to be captured by the swarm behavior generated by the LLM. Interestingly, the highest median and whisker of the box plot is found in the scores of the control dataset. Apparently, the foraging missions benefit from increased diversity of uniform and guided light placements on average even though the fine-tuned LLM is not able to take advantage of individual high potential scenarios as much.

In the *Connection* mission performances, the LLM benefited from the targeted lights at the areas overall. The fitness scores from the control trained model come close, but do not reach the same performance. The model for uniformly placed lights has the lowest median. Interestingly, the control model produced the highest number of controllers at the overall minimum of the scale.

The most visible distinction can be observed for *Aggregation* missions. Clearly, lights placed at the aggregation target is taken use of by the model, resulting in significantly higher fitness overall. The control LLM only shows an almost unnoticeable difference to the uniform scores, indicating that it has difficulties learning how to make use of the benefits of the targeted lights included in the mix.

For the *Distribution* missions, the best scores are achieved by the LLM trained on uniformly distributed lights only. The lights placed in a rectangular fashion, to indicate the size of the area, does not seem to help, affecting also in the control LLM that achieves a similar median, although the peak close to the optimum fitness is visually less pronounced there.

#### — 4.2.4 Effect of Natural Language

The effect of natural language is studied by changing the input to the model. In this experiment, the prompt is no longer the natural language description of the scenario. Instead, the prompt input is an XML string that directly encodes all the parameters of the different mission elements sampled during dataset sampling (see Section 3.1.2) instead of the natural language description of the scenario. The output still is the behavior tree. An example can be found in the appendix.

The idea is that a more structured formal language like XML might be easier to parse and extract concise instructions from, so that the fine-tuned model might better adapt to the given input. The base model is fine-tuned on a dataset of 2250 scenarios and evaluated on a dataset with 250 entries. The results get compared with the results from the main experiment after training on natural language descriptions of the same 2250 scenarios. Figure 10 shows the results.

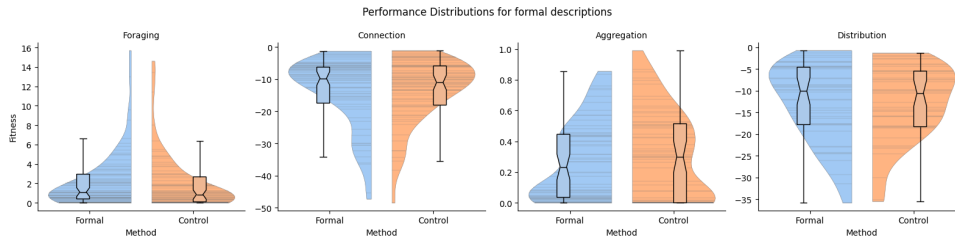


Figure 10: This figure shows the fitness score distributions after fine-tuning on XML input. They are marked as *Formal* and get compared with the results from the main experiments of the same training data size, indicated as *Control*. While the distributions are very similar in general, formal inputs seem to have a slight advantage for all but the aggregation type missions. For *Aggregation*, natural language descriptions have the visually most distinct differences in their fitness distributions.

Formal descriptions seem to have a slight advantage for mission types *Foraging*, *Connection* and *Distribution*. However, because of overlapping notches in the box plots, the difference cannot be found significant. Interestingly, for *Aggregation* missions, formal descriptions lead to an improved minimal performance but worse optimal performance at the same time. Furthermore, when fitness scores cluster at a specific fitness in one input type, there often is a corresponding cluster at the other input type. This is especially pronounced for *Distribution* missions, implying that the resulting scores are probably not a result of the description, but the underlying scenario itself.

To add to the Section 4.2.2 about the environment impact of the aggregation target color, the results for *Aggregation* missions are looked into more deeply. Figure 11 next shows its individual results separated by target color. Similar to the aforementioned previous experiment, white targets have generally lower fitness scores. This means that the lower scores are not a result of a mistake or bug in the natural language descriptions. They are not used in this experiment. This rather suggests a more ground level structural difference for these two targets, at least for LLMs and this dataset.

A second experiment is done where the prompt stays the natural language input, but a description of

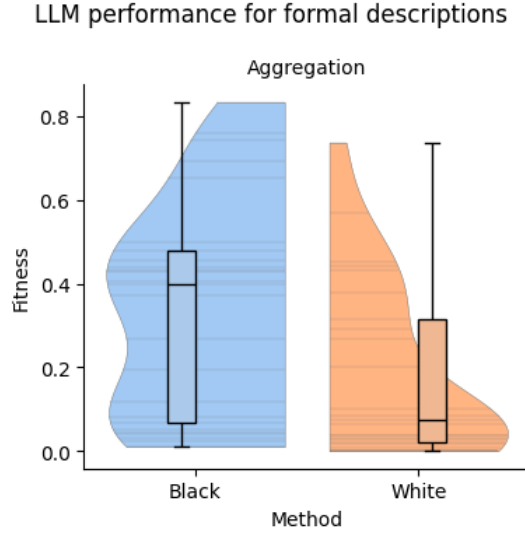


Figure 11: This figure shows the achieved scores for *Aggregation* missions on XML prompts, separated by target color. White aggregation targets have a considerably lower median by about 0.3 fitness and overall lower maximum fitness.

how the behavior tree syntax works is appended to every prompt in training and inference. The training dataset and evaluation datasets are the same as for the best model in the main experiment. The results are shown in Figure 12. The actual prompt and a comparison to the AutoMoDe baseline is given in the appendix. The results show that the natural language description of the abstract command line syntax of the behavior tree slightly improves the fitness scores. The median for *Aggregation* changes from 0.3 to 0.314, for *Connection* from  $-11.04$  to  $-10.7$ , for *Distribution* from  $-10.58$  to  $-10.52$  and for *Foraging* from 0.85 to 1.20, more than doubling the maximum fitness.

These results indicate that the description of behavior tree syntax in a natural language, which the model is familiar with from pre-training, helps to improve the results. This implies that the model has made a connection from its general understanding of natural language sentences to the swarm missions and behavior tree controllers in particular.

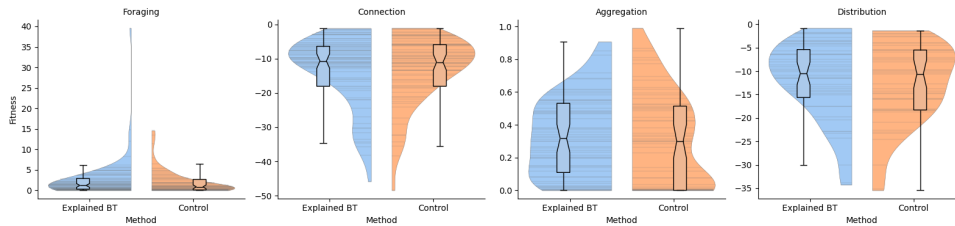


Figure 12: This figure shows a comparison of how appending a natural language description of the behavior tree syntax at the end of every prompt changes the quality of the results. It is compared to the model from the main experiment that is trained on 1575 examples (displayed as *Control*). The results show a slight improvement, especially for *Aggregation*, where the peak at score zero is less pronounced and *Foraging*, where the highest score for that mission overall is achieved.

#### — 4.2.5 Generalization Capabilities

The capabilities of the LLM to generalize swarm behavior is evaluated in two ways. First, the abilities to transfer from missions it is trained on to unseen mission types is investigated. For this experiment, the scenarios are split into a training dataset of 1895 entries, containing scenarios for *Foraging*, *Connection* and *Distribution* missions. The trained model is evaluated on a 669 scenario dataset, containing *Aggregation* missions only. The results are displayed in the Figure 13.

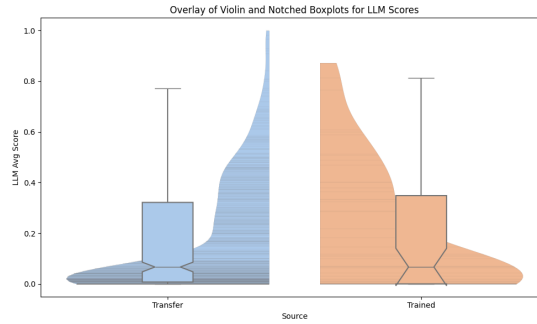


Figure 13: This figure shows the fitness distribution for Aggregation missions achieved by an LLM not trained on aggregation type missions. The distribution is marked as *Transfer*. For comparison, the aggregation performance of the main experiment after training on 675 scenarios, including *Aggregation*, is displayed as *Trained*.

The LLM is able to generate controllers with up to perfect fitness of 1.0 without being trained on aggregation type missions and in general shows comparable results to a normally trained model on 675 scenarios. Both have a median fitness of 0.667. While the median is significantly lower when compared to best result from the main experiment (compare to Figure 7), partial success can be claimed, because higher scores are reached repeatedly, only not consistently enough for a high median. This suggests that in some cases the model has made use of its pre-training to understand the intention of the aggregation descriptions and used its experience from fine-tuning on the other mission types to generate a fitting behavior tree. This can also be seen in qualitative examples from Table 5.

The second way to investigate the LLM’s generalization capabilities is by looking into the same mission type. It is investigated how well the best performing AutoMoDe controller performs in other scenarios of the same mission type and compared to the trained LLM as a whole. This is appropriate, as behavior tree generation from context is a zero-shot task for LLMs and does not require optimization for each scenario individually. Furthermore, the generalization capabilities attributed to LLMs pose one of their potential main benefits and is thus evaluated like that.

For this evaluation, the dataset created for Section 4.2.2 is reused. The highest performing AutoMoDe behavior tree on *Aggregation* missions with black target area is taken and evaluated on all other scenarios in the dataset. For comparison, the LLM trained on the same dataset is used. The results are displayed in Figure 14. It shows that the best AutoMoDe behavior tree achieves non-zero scores in many other scenarios, but has a significantly worse median than the LLM generated controllers for the same missions. It can be seen that the fine-tuned LLM achieves high fitness much more frequently. The AutoMoDe behavior tree likely only achieves very high fitness if the scenario is similar to the scenario the controller was optimized for. This shows that the LLM has learned to internalize the different kinds of controllers that are necessary for the different variants of aggregation missions and returns fitting behavior trees according to the respective scenario. This shows that LLMs can improve the generalization capabilities of AutoMoDe generated controllers.




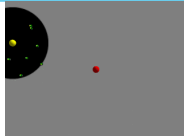




	Scenario 324	Scenario 191	Scenario 465
Last Frame			
Trace			
Description	<p>The circular arena, constructed with 19 walls, has a radius of 3.17 m. The following lights are present in the arena at coordinates: <math>((-0.36, 0.14))</math>. Evenly positioned around the origin are 14 robots within a radius of 1.20 meters. The goal is for the robots to aggregate at the black circle. There are two areas on the floor: a circle at <math>[-1.47, 1.91]</math> with a radius of 0.53 meters in white, and another circle at <math>[-0.36, 0.14]</math> with a radius of 1.03 meters in black.</p>	<p>With a radius of 4.36 meters, the circular arena is made up of 9 walls. At the black circle is a light. There are 15 robots placed uniformly around the center within a radius of 2.19 meters. In the arena, you'll find two areas: a circle at <math>[2.58, -1.20]</math> with a radius of 0.36 meters in white, and another circle at <math>[0.78, 2.47]</math> with a radius of 1.06 meters in black. The robots' goal is to group together at the black circle.</p>	<p>A rectangular area, with a length of 4.11 meters, width of 5.52 meters, and height of 1.29 meters, is established. 1 light illuminates the space, located at: <math>((-0.33, 0.43))</math>. 14 robots are evenly distributed around the origin within a radius of 1.87 m. The objective is for the robots to aggregate at the black circle. There are two designated areas on the floor: a circle at <math>[-2.19, -1.28]</math> in white, and another circle at <math>[-0.33, 0.43]</math> in black.</p>
Behavior Tree	<pre>--nroot 3 --nchildroot 2 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.9115 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.2905 --n11 5 --a11 3 --p11 0</pre>	<pre>--nroot 3 --nchildroot 3 --n0 0 --nchild0 2 --n00 6 --c00 0 --p00 0.9199 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.0955 --n11 5 --a11 0 --rwm11 2 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 0 --p20 0.5555 --n21 5 --a21 2 --p21 0</pre>	<pre>--nroot 3 --nchildroot 2 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.9119 --n01 5 --a01 2 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 1 --p10 0.3905 --n11 5 --a11 3 --p11 0</pre>

Table 5: This figure shows exemplary LLM-generated behavior trees for *Aggregation* missions when only trained on other mission types. The display format is the same as the main experiment (Table 4), but the scenarios are now from the evaluation dataset for the generalization experiment. These examples of successful generalization show that the fine-tuned LLM is able to find the aggregation target, especially when a lamp is placed on its center. Most robots stay on target, with a small outlier failing to find the target in scenario 191. It can be seen that in these scenarios the robots do not stand still on the target but keep moving around it. Especially in scenarios 191 and 465, as soon as individual robots leave the target, they swerve and quickly return, showing an intention to stay on target. Overall, these examples show that the LLM can successfully generalize to new mission types in individual cases.

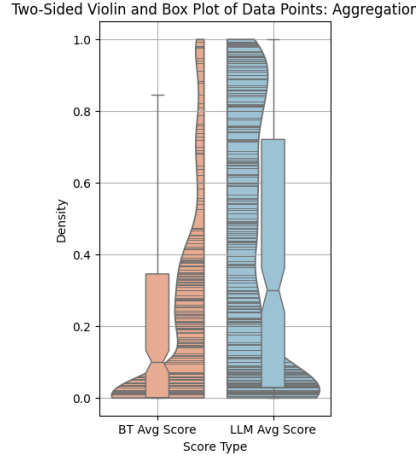


Figure 14: This figure shows the best performing aggregation controller generated by AutoMoDe evaluated on all aggregation missions with the same target color. The results are displayed as *BT Avg Score* and get compared to the LLM scores after fine-tuning on aggregation missions. A significant difference can be observed. This finding is supported by a Wilcoxon test (Wilcoxon, 1945) which computes a probability value of  $9.45 \cdot 10^{-26}$ . This means that the null-hypothesis, that the distributions are the same, needs to be rejected, implying they are indeed significantly different. Thus, LLMs significantly improve AutoMoDes generalization capabilities.

#### — 4.2.6 Reinforcement Learning

As an alternative approach to classical supervised fine-tuning, reinforcement learning (RL) is investigated. Motivation for this approach is the score based approach used in training. This way, the model can be taught not only how good solutions look like but the quality of the individual solutions can be communicated via score as well. The aspiration is to further improve the results from the main experiment with this method.

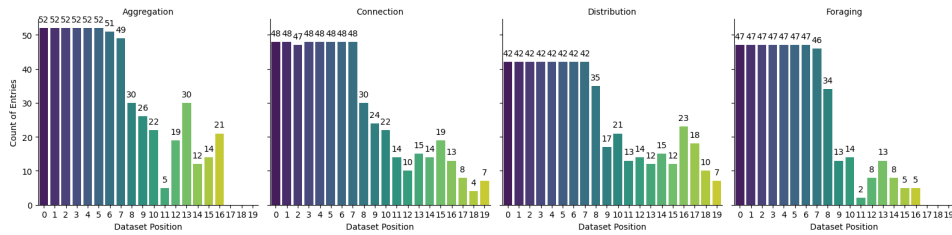


Figure 15: This figure shows the number of scenarios by mission and dataset iteration, where valid behavior trees have been generated. It can be seen that after 8 iterations of RL, the number of valid responses decreases rapidly. Sometimes only a single digit number of scenarios remain per mission type. 200 scenarios are evaluated per iteration in total.

Direct preference optimization (DPO) by Rafailov et al. (2024) is used as the reinforcement learning algorithm for this experiment. The implementation is from the TRL library (von Werra et al., 2020). For DPO the dataset needs to be shaped in pairs of a rejected response and accepted response paired with their respective score. The input prompt stays the natural language description of each scenario. To obtain this dataset, a different method is used than for the other experiments. The trained model from Section 4.2.1 is used to generate two different responses for newly sampled scenarios. As opposed to

the text generation for evaluating models, a sampling based approach is used instead of classical greedy decoding. There, not the most likely word is chosen each iteration but is sampled from the model's softmax (Hinton, 2015) output distribution according to a temperature which is set to 0.14 in this case. This way, different controllers can be generated for the same input scenario. The behavior trees get evaluated in ARGoS to obtain their score as done in the other experiments. The scores then get min-max normalized by mission type and classified into rejected and accepted according to which behavior tree achieves higher the score.

The dataset with the accepted/rejected responses and normalized scores is then passed to the DPO algorithm for fine-tuning. Initially, the fine-tuned model of the main experiment is further being trained. A detailed overview of the exact DPO parameters can be found in the appendix. This procedure gets repeated with the DPO fine-tuned model that now generates the next dataset for fine-tuning until a maximum number of iterations is reached.

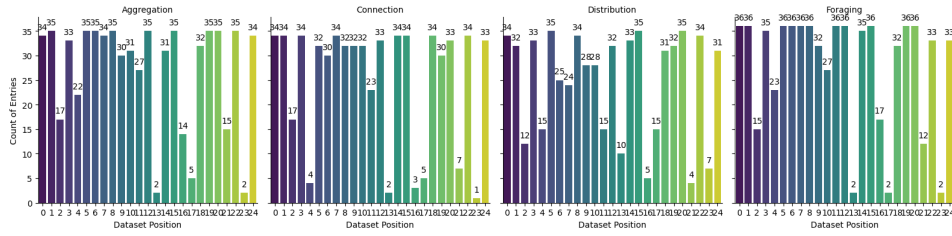


Figure 16: This figure shows the number of scenarios with valid generated behaviors by mission and dataset iteration when supervised fine-tuning is performed before DPO. While the number of valid responses drops occasionally, it does not stay low and remains stable overall. In this example, 25 epochs of RL have been performed and 150 scenarios are sampled each iteration.

However, this approach has been found not to be stable. At some point, the number of valid behavior trees that are generated collapses. This can be seen in Figure 15. To circumvent this issue, the RL training pipeline is altered. Before training with DPO, supervised fine-tuning is performed for a single epoch. Figure 16 shows that this stabilizes the pipeline.

The results for training for 15 iterations on 150 scenarios each are shown in Figure 17. In contrast to the preliminary experiments shown in Figures 15 and 16, the DPO learning rate has been reduced by several orders of magnitude to  $5 \cdot 10^{-9}$ . This has helped in reducing the number of erroneous behavior trees even more.

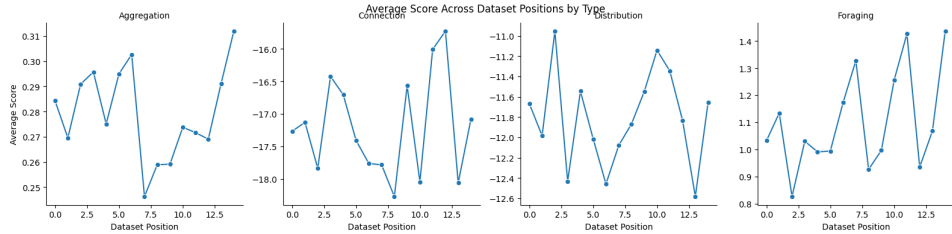


Figure 17: This figure shows the average fitness score achieved in each iteration of DPO training by mission type. It can be seen that the process is very volatile and only small gains are achieved.

After training, the trained model gets evaluated on the same dataset as the base model from the main experiment. The average scores can be seen in the following table:

Type	DPO	SFT Base
Aggregation	0.313732	0.307450
Connection	-13.615785	-13.242303
Distribution	-12.579867	-12.842702
Foraging	1.784000	1.944595

Table 6: Evaluation results after DPO training compared to the SFT base model from the main experiment. The table shows the mean of the scores achieved on the evaluation dataset.

While the DPO trained model shows some progress in Figure 17, its magnitude is small. This is reflected in the evaluation results, where the model is only able to increase the average for *Aggregation* missions by 0.07. For *Foraging*, the average score becomes even slightly worse after DPO training.

The big volatility in training can potentially be explained by a training rate that is too high. However, the training rate was reduced already to the current level during preliminary experiments and high volatility is still present. Another potential reason is the diversity between the datasets. As the datasets get sampled individually every iteration, they are not the same, leaving place for different optimal averages that are possible for each dataset. This could be circumvented with higher dataset sizes because of the law of large numbers, but this is costly time-wise. With the currently time-wise unoptimized implementation, it takes about 1.5 hours to generate and evaluate samples for 150 scenarios.

Another experiment has been run with a different base model. This time, the base model to be optimized with RL is not the best model from the main experiment, but a worse model using all of the available training scenarios. From Table 7, it can be seen that DPO was able to lift these base model performances to a similar level to the optimal model from the main experiment. This suggests that DPO is working correctly and that there is a structural boundary in the dataset that represents a major obstacle for LLM learning in the form of a local optimum that is difficult for them to pass.

Type	DPO (difficult start)	SFT Base
Aggregation	0.278498	0.088818
Connection	-19.139695	-18.761727
Distribution	-10.100879	-11.454326
Foraging	1.860759	0.317500

Table 7: DPO results starting from a difficult base model compared to the SFT base in Table 6. The table shows the mean of the scores achieved on the evaluation dataset.

## 5 Discussion

In general, the experiments show that the LLM has learned how to generate working behavior trees for robot swarm missions. In all four mission types, a wide range of fitness scores have been achieved, although there remains a noticeable difference between mission types and in comparison to the AutoMoDe baseline. For *Distribution* and *Connection* missions, the score distributions are very similar, whereas for *Foraging* the shape is similar but squashed. Very high scores achieved by AutoMoDe result in LLM scores that are much lower. For *Aggregation* missions, the LLM struggles with variants where the target is white instead of black. This effect is not due to poor or ambiguous natural language descriptions, as the experiment on formal inputs shows the same behavior. The fact that there is little difference between formal and natural language descriptions suggests that LLMs can be used to interact with robot swarms using natural language successfully, although the unexplained target color issue indicates that an error that has not been found cannot be dismissed completely.

While the LLM-based controllers are not optimal compared to AutoMoDe, they do adjust their responses according to the actual scenario, providing better generalization than a single controller optimized for a specific scenario. The model appears to have learned some general rules about swarm robotics, as evidenced by its ability to produce working controllers for *Aggregation* missions even without specific training. In many cases, the model likely leverages its pre-training to understand the expected behavior and combines that with what it has learned about working behavior trees in other missions. Visual inspection of high-performing generalization cases reveals that a light at the target is a common feature, suggesting that the model has internally associated the light with the mission objective or the ground areas—a finding supported by experiments where fine-tuning on intentionally placed lights to guide the robots had the most significant impact for *Aggregation* missions.

An important observation is the transferability of the LLM-generated controllers across mission types. Experiments indicate that the model can generate working controllers for missions it was not specifically trained on, as shown on generating aggregation controllers when trained only on Foraging, Connection, and Distribution missions. This suggests that the LLM has internalized a more abstract and general representation of swarm behavior, which could be exploited further for real-world adaptability and cross-domain applications.

One notable outcome of the experiments is that while working controllers can be generated by LLMs, they often lack the nuanced understanding required for truly optimal behavior trees. This comparison may seem unfair, however, since AutoMoDe spends several tens of thousands of experiments to optimize each scenario, whereas the LLM produces responses instantly with constant computation time per token and relies only on about two thousand training examples. Moreover, the base LLM was trained on data from the World Wide Web, which is vastly different from the examples for collective behavior used in the experiments. This further underlines how limited training data can serve as a sufficient basis for understanding.

Yet, as the overall performance decreased after a dataset size of 1575 scenarios, simply increasing the amount of data does not seem beneficial. This observation appears to represent an intrinsic limit of what can be achieved with the current training dataset, as even the reinforcement learning approach seems constrained by this same limit.

In addition, while the fine-tuned model does adjust its responses for the respective scenario, it is not doing so sufficiently to consistently achieve optimal scores. The responses tend to behave like an average solution for each scenario, a phenomenon that aligns with the findings by [Chen et al. \(2023\)](#), who observed that LLMs seem to prefer an averaging strategy. Interestingly, the reinforcement learning experiment exhibits similar behavior despite being able to communicate actual scores in its feedback. Perhaps a more conscious exploration of the solution space, akin to the approach used in evolutionary algorithms, could

overcome this local optimum. It is also worth considering that the similar results observed for AutoMoDe and the fine-tuned LLM on *Distribution* and *Connection* missions may indicate that the AutoMoDe behaviors for these missions represent an easier, average or greedy optimum solution that the LLM can more readily approximate compared to more complex cases.

Another implication of the experiments is the variability present in the responses. Occasionally, the model is able to produce behavior trees that are optimal, yet at other times the resulting controllers achieve minimal performance. Especially after little training and with sampling enabled, the model sometimes produces outputs that are invalid or only loosely related to the training data. Although this issue of reliability improves with increased dataset size, it remains a visible problem during reinforcement learning, suggesting that further work could explore ways to minimize the number of failed responses.

Finally, there are additional techniques for LLM training that were not explored in this thesis. Recent successes in models like [Guo et al. \(2025\)](#), which employ chain-of-thought ([Wei et al., 2022b](#)) reasoning, suggest that this approach can dramatically improve language model performance. Furthermore, the parameters for model training in this work were estimated by trial and error and then left fixed. A more structured approach, such as incorporating a hyper-parameter optimization framework, could potentially lead to further improvements in the results.

An important aspect that warrants further emphasis is the computational efficiency and potential dataset biases. The LLM’s ability to instantly generate controllers contrasts sharply with AutoMoDe’s extensive optimization process, a significant advantage for real-time or adaptive applications. Moreover, the persistent performance drop for white targets may point to underlying biases or subtle imbalances in the training data that deserve more detailed analysis. Addressing these issues could not only improve performance but also enhance the practical applicability and scalability of LLM-based controllers in swarm robotics.

For building an application for one-shot swarm interaction from natural language, it might be beneficial to revisit different prompting strategies and methods to add more natural language versatility and descriptions to the training dataset. With a different base model that supports images, it might even be useful to add a map of the scenario or provide visual feedback of the actual behavior in execution. Further work regarding the models understanding of collective systems and generation of optimal behavior could focus on the general dataset diversity and training strategy to promote results more optimal for each scenario. Potentially, it might even be helpful to train a very small language model from scratch only on the collective behaviors. Most of the pre-trained knowledge does not seem to be used from the experiment results, although encouraging so does improve them. Nevertheless, that method would have the chance to be trained more efficiently because of the smaller parameter count. However, a significantly larger and more diverse dataset would likely be necessary and different prompting, that represents the behavior tree in a less abstract way, or methods like chain-of-thought might have a more immediate effect.

## 6 Conclusion

This thesis has used AutoMoDe to create a dataset of scenarios with 4 mission types for swarm robots and working controllers in a structured way. This dataset was used to fine-tune a pre-trained large language model for creating behavior tree robot controllers for these missions. Different experiments have been conducted to investigate the impact of several characteristics like the environment, mission type, generalization capabilities, and language formalization.

The results show that the LLM can successfully generate robot controllers for all missions, but its performance stays behind that of AutoMoDe. This is especially true for the mission types *Aggregation* and *Foraging*. The reason is likely the restricted diversity of the training dataset. It would be interesting for further work to investigate if a significantly larger and more diverse dataset could help bridge that performance gap.

Nevertheless, the fine-tuned model shows some degree of generalization, especially within the same type of mission. Further, successful controllers for mission types outside of the training dataset have been generated as well with the example of *Aggregation*. A dedicated experiment on a formalized version of the input description has shown that natural language descriptions are a viable option and do not perform worse than formal ones. Instead, a good description of the behavior tree allows the LLM to make use of its pre-training to understand the structure and improves the results.

In conclusion, LLMs can be trained to generate working behavior trees for different swarm robot missions and learn the swarm behavior good enough to generalize to unseen but similar missions. Major challenges remain with achieving the same performance as state-of-the-art approaches like AutoMoDe and the consistency with which good results are generated. Further work might introduce a more diverse and significantly larger dataset that might be helpful to overcome this gap.

## Appendix

### LLM Training Loss

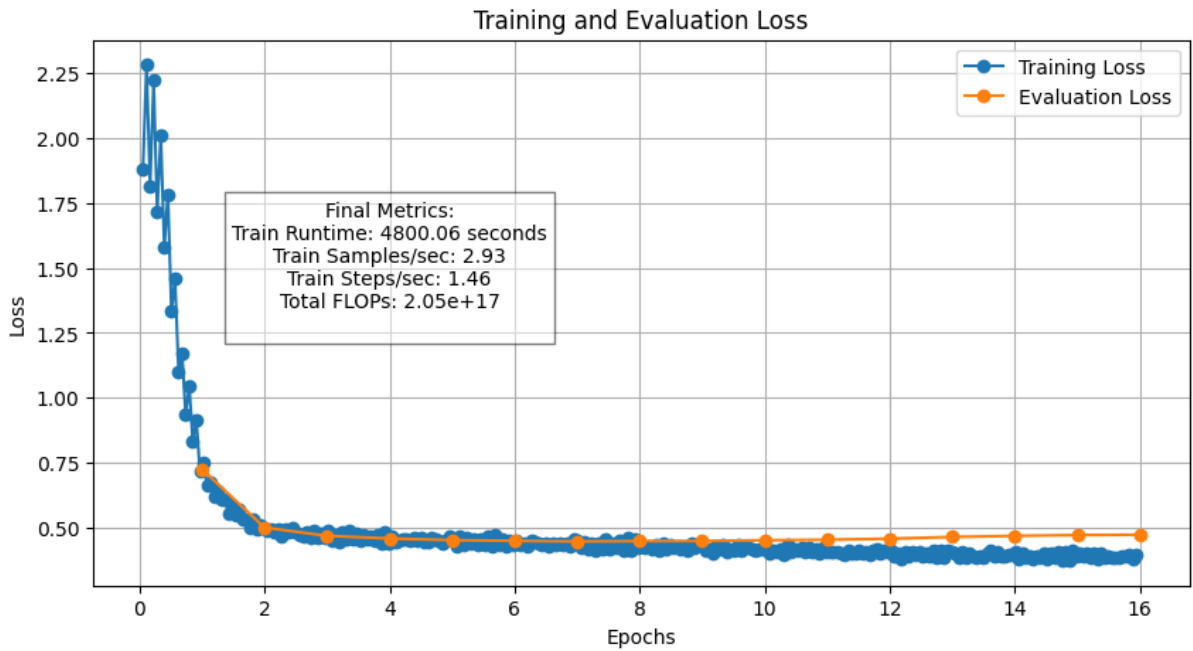


Figure 18: Training loss during supervised fine-tuning for the main experiment on the 1575 entries dataset.

### Full Dataset Entry

Here, an example for a full entry of the generated dataset is shown. First the outcome after simulation is displayed, then the natural language description, then the AutoMoDe-optimized behavior tree in its command line representation, then the formal description. Finally, the configuration used for AutoMoDe in simulation is shown.

The example is scenario 74 from the main experiments evaluation dataset.

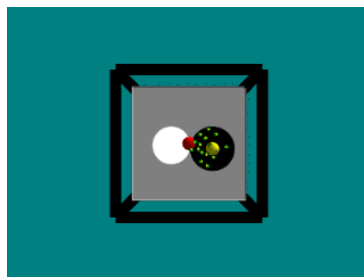


Figure 19: Last frame after executing the optimized behavior tree



### Natural Language Description:

A rectangular area, with a length of 1.65 meters, width of 1.66 meters, and height of 2.19 meters, is established. A light shows the way to the black area. 16 robots are evenly spaced around the central point, spanning a radius of 0.45 m. The goal is for the robots to aggregate at the black circle. There are two areas on the floor: a circle at  $[-0.08, -0.35]$  with a radius of 0.33 meters in black, and another circle at  $[-0.03, 0.26]$  with a radius of 0.28 meters in white.

Listing 1: Command-line behavior tree (line breaks are only inserted for displaying only)

```
--nroot 3 --nchildroot 3 --n0 0 --nchild0 2 --n00 6 --c00 2 --p00 0.6193 --n01 5 --a01 0
--rwm01 3 --p01 0 --n1 0 --nchild1 2 --n10 6 --c10 2 --p10 0.6412 --n11 5 --a11 4 --
att11 3.7667 --p11 0 --n2 0 --nchild2 2 --n20 6 --c20 3 --p20 6 --w20 17.8635 --n21 5
--a21 4 --att21 2.7507 --p21 0
```

```
<?xml version="1.00" ?>
<config>
<swarm-elems>
<entity quantity="16" max_trials="100">
<e-puck id="epuck">
<controller config="automode_bt"/>
</e-puck>
</entity>
</swarm-elems>
<env-elems>
<light id="light_0" position="-0.08,-0.35,0.00" orientation="360,0,0" color="yellow" intensity="
11.30" medium="leds"/>
<box id="wall_0" size="0.01,1.66,0.10" movable="false">
<body position="0,-0.83,0.00" orientation="90.00,0.00,0.00"/>
</box>
<box id="wall_1" size="0.01,1.65,0.10" movable="false">
<body position="0.83,0,0.00" orientation="0.00,0.00,0.00"/>
</box>
<box id="wall_2" size="0.01,1.66,0.10" movable="false">
<body position="0,0.83,0" orientation="90.00,0.00,0.00"/>
</box>
<box id="wall_3" size="0.01,1.65,0.10" movable="false">
<body position="-0.83,0,0" orientation="0,0.00,0.00"/>
</box>
<arena-attrib size="1.66,1.65,2.19"/>
</env-elems>
<objective-elems>
<spawnCircle position="0,0,0" radius="0.45"/>
<circle position="-0.08,-0.35" radius="0.33" color="black"/>
<circle position="-0.03,0.26" radius="0.28" color="white"/>
<objective type="aggregation">
<objective-params target-color="black" radius="0.33"/>
</objective>
</objective-elems>
</config>
```

Listing 2: Formal description with XML

```
<?xml version="1.00" ?>
<argos-configuration>
<framework>
<experiment length="120" ticks_per_second="10" random_seed="0"/>
</framework>
<loop_functions library="/root/AutoMoDe-loopfunctions/build/loop-functions/custom-loopfunctions/
libcustom_loopfunc.so" label="template">
<spawnCircle position="0,0,0" radius="1.63"/>
<circle position="1.06,0.08" radius="0.48" color="white"/>
```

```

<circle position="1.78,1.27" radius="0.52" color="black"/>
<objective type="connection">
<objective-params conn_start="black" conn_end="white" connection_range="0.27"/>
</objective>
</loop_functions>
<controllers>
<automode_controller_bt id="automode_bt" library="/root/AutoMoDe/build/src/libautomode_bt.so">
<actuators>
<epuck_wheels implementation="default" noise_std_dev="0.05"/>
<epuck_rgb_leds implementation="default" medium="leds"/>
<epuck_range_and_bearing implementation="medium" medium="rab" data_size="4" range="0.70"/>
</actuators>
<sensors>
<epuck_proximity implementation="default" show_rays="false" noise_level="0.05" calibrated="true"/>
<epuck_range_and_bearing implementation="medium" medium="rab" data_size="4" noise_std_deviation="
1.50" loss_probability="0.85" calibrated="true"/>
<epuck_light implementation="default" show_rays="false" noise_level="0.05" calibrated="true"/>
<epuck_ground implementation="rot_z_only" noise_level="0.05" calibrated="true"/>
<epuck_omnidirectional_camera implementation="rot_z_only" medium="leds" show_rays="false"/>
</sensors>
<params bt-config="--nroot_3--nchildroot_1--n0_0--nchild0_2--n00_6--c00_5--p00_0.26--n01_5_
--a01_1--p01_0"/>
</automode_controller_bt>
</controllers>
<arena size="3.83,3.60,1.19" center="0,0,0">
<floor id="floor" source="loop_functions" pixels_per_meter="300"/>
<light id="light" position="0.00,0.00,0.00" orientation="0,0,0" color="red" intensity="5.00" medium
="leds"/>
<distributed>
<position method="uniform" min="-1.00,-1.00,0" max="1.00,1.00,0"/>
<orientation method="gaussian" mean="0,0,0" std_dev="360,0,0"/>
<entity quantity="15" max_trials="100">
<e-puck id="epuck">
<controller config="automode_bt"/>
</e-puck>
</entity>
</distributed>
<light id="light_0" position="-0.32,1.36,0.00" orientation="360,0,0" color="yellow" intensity="7.00
" medium="leds"/>
<light id="light_1" position="-0.69,0.20,0.00" orientation="360,0,0" color="yellow" intensity="5.40
" medium="leds"/>
<light id="light_2" position="-1.32,-0.16,0.00" orientation="360,0,0" color="yellow" intensity="
3.82" medium="leds"/>
<box id="wall_0" size="0.01,3.83,0.10" movable="false">
<body position="0,-1.80,0.00" orientation="90.00,0.00,0.00"/>
</box>
<box id="wall_1" size="0.01,3.60,0.10" movable="false">
<body position="1.92,0,0.00" orientation="0.00,0.00,0.00"/>
</box>
<box id="wall_2" size="0.01,3.83,0.10" movable="false">
<body position="0,1.80,0" orientation="90.00,0.00,0.00"/>
</box>
<box id="wall_3" size="0.01,3.60,0.10" movable="false">
<body position="-1.92,0,0" orientation="0,0.00,0.00"/>
</box>
</arena>
<physics_engines>
<dynamics2d id="dyn2d"/>
</physics_engines>
<media>
<led id="leds" grid_size="1,1,1"/>
<range_and_bearing id="ircom"/>
<range_and_bearing id="rab"/>
</media>

```

```

<visualization>
<!-- Visualization configuration -->
</visualization>
</args-configuration>

```

Listing 3: Configuration for ARGoS3 simulator

## Training with Behavior Tree Explanations

The text that gets appended to the prompt for including the behavior tree explanations is changed to this:

```

Generate the behavior tree that achieves the objective of this mission.\nThis is how the desired behavior tree syntax is structured:The first element is always the node identifier (like --nroot or --n0), followed by the type of node. 1 and 1 define selector nodes, 2 and 3 sequence nodes, 4 is a decorator node, 5 an action and 6 a condition. Different versions of the same node type are specified by --c[ nodeID] parameter for conditions and --a[nodeID] parameter for actions. Conditions range from 0-6 with the order black floor, gray floor, white floor, neighbors count, inverted neighbors count, fixed probability and light. Depending on the condition, they have additional attributes like probability (p ) or neighbors and epsilon (p, w). Actions range from 0-5 in the order exploration, stop, phototaxis, anti-phototaxis, attraction and repulsion. They all have a success probability parameter p. Additional parameters for some actions are steps range (rwm), attraction (att) and repulsion rep.\nNow generate the robot controller.

```

Listing 4: Behavior tree explanation prompt

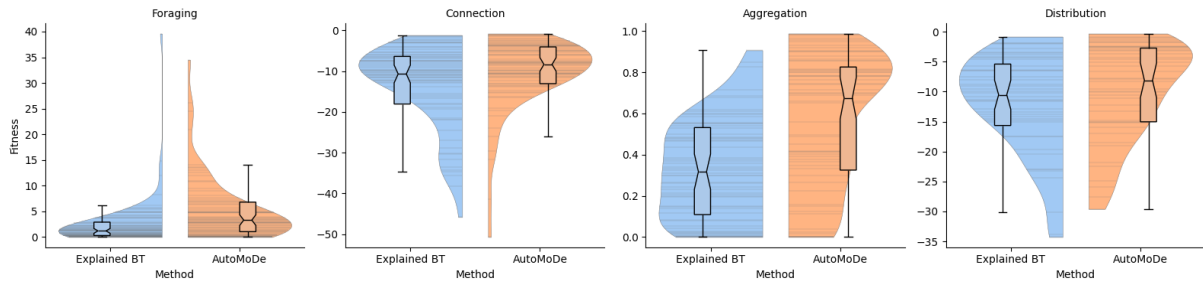


Figure 20: Fitness scores after training on prompts that include an explanation of the behavior tree syntax in comparison to the AutoMoDe baseline. It can be seen that with that training, the LLM produces the highest score for *Foraging* overall.

## Inconsistent LLM Output

```

Generate the behavior tree that achieves the objective of this mission.[/INST] |BTSTART||BTEND|--nroot 3 --
nchildroot 3 --n0 0 --nchild0 2 --n00 6 --c00 3 --p00 5 --w00 12.205 --n01 5 --a01 2 --p01 0 --n1 0 --
nchild1 2 --n10 6 --c10 2 --p10 0.4406 --n11 5 --a11 0 --rwm11 6 --p11 0 --n2 0 --nchild2 2 --n20 6 --
c20 2 --p20 0.0833 --n21 5 --a21 1 --p21 0|BTEND|--nroot 3 --nchild3 2 --n30 6 --c30 2 --p30 0.0509 --
n31 5 --a31 1 --p31 0|BTEND|</s>

```

Listing 5: Behavior Tree Generation Command

## LLM SFT Training Parameters

```
bf16: false
dataset_text_field: text
eval_steps: 1
eval_strategy: epoch
evaluation_strategy: epoch
gradient_accumulation_steps: 2
greater_is_better: false
group_by_length: true
learning_rate: 2.0e-05
load_best_model_at_end: true
logging_steps: 25
lr_scheduler_type: cosine
max_grad_norm: 0.15
max_seq_length: 1000
max_steps: -1
metric_for_best_model: eval_loss
num_train_epochs: 12
optim: paged_adamw_32bit
output_dir: logs
per_device_train_batch_size: 1
report_to: none
save_steps: 0
save_strategy: epoch
warmup_ratio: 0.1
weight_decay: 0.123
```

Listing 6: Training parameters for supervised fine-tuning in the main experiment

## LLM DPO Training Parameters

```
output_dir: "DPO"
report_to: "none"
model_adapter_name: "train_adapt"
ref_adapter_name: "reference_adapt"
per_device_train_batch_size: 1
per_device_eval_batch_size: 1
logging_dir: "logs"
logging_steps: 10
gradient_accumulation_steps: 1
eval_accumulation_steps: 1
max_prompt_length: 300
max_completion_length: 300
learning_rate: self.learning_rate
num_train_epochs: 3.0
```

Listing 7: Training parameters for DPO experiment (Learning rate is varied but typically set to  $5 \cdot 10^{-9}$ )

## References

- H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight Jr, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 2000. 1, 7
- G. Aguzzi, R. Casadei, and M. Viroli. Macroswarm: a field-based compositional framework for swarm programming. In *International Conference on Coordination Languages and Models*. Springer, 2023. 7
- J. Ao, F. Wu, Y. Wu, A. Swikir, and S. Haddadin. Llm as bt-planner: Leveraging llms for behavior tree generation in robot task planning. *arXiv preprint arXiv:2409.10444*, 2024. 14, 25
- J. Bachrach, J. Beal, and J. McLurkin. Composable continuous-space programs for robotic swarms. *Neural Computing and Applications*, 2010. 7
- J. Beal and J. Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems*, 2006. 1, 7
- M. Ben-Ari and F. Mondada. *Elements of Robotics*. Springer Nature, 2017. 1
- G. Beni and J. Wang. Swarm intelligence in cellular robotic systems. In *Conference Proceedings*, 1993. doi: 10.1007/978-3-642-58069-7\_38. 3
- M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. *Experimental methods for the analysis of optimization algorithms*, 2010. doi: 10.1007/978-3-642-02538-9\_13. 11
- D. Bozhinoski and M. Birattari. Towards an integrated automatic design process for robot swarms. *Open Research Europe*, 2022. 7, 8, 18
- M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 2013. 4, 5
- A. Brohan, N. Brown, J. Carbajal, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint*, 2023. 2, 13
- A. Buckner, L. Figueredo, S. Haddadin, et al. Latte: Language trajectory transformer. In *IEEE International Conference on Robotics and Automation*, 2023. 2, 13
- S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraula, and E. Bonabeau. *Self-organization in Biological Systems*. Princeton university press, 2001. 1
- Y. Cao and C. G. Lee. Robot behavior-tree-based task generation with large language models. *arXiv preprint*, 2023. 2, 14
- H. Chen, W. Ji, L. Xu, and S. Zhao. Multi-agent consensus seeking via large language models. *arXiv preprint arXiv:2310.20151*, 2023. 15, 42
- P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 2017. 13
- M. Colledanchise and P. Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics*, 2016. 9
- M. Colledanchise and P. Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018. 10

- W. J. Conover. Practical nonparametric statistics. *Probability and Statistics*, 1999. [11](#)
- V. Crespi, A. Galstyan, and K. Lerman. Top-down vs bottom-up methodologies in multi-agent system design. *Autonomous Robots*, 2008. [5](#)
- V. Darley. Emergent phenomena and complexity. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, 1994. [3](#), [5](#)
- T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021. [25](#)
- T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefer, and D. Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023. [13](#)
- T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 2024. [13](#), [25](#)
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. [1](#), [12](#)
- P. G. F. Dias, M. C. Silva, G. P. Rocha Filho, P. A. Vargas, L. P. Cota, and G. Pessin. Swarm robotics: A perspective on the latest reviewed concepts and applications. *Sensors*, 21(6), 2021. ISSN 1424-8220. doi: 10.3390/s21062062. [1](#), [4](#)
- M. Dorigo and E. Şahin. Guest editorial. *Autonomous Robots*, Sep 2004. ISSN 1573-7527. doi: 10.1023/B:AURO.0000034008.48988.2b. [3](#)
- D. Driess, F. Xia, M. S. Sajjadi, et al. Palm-e: An embodied multimodal language model. *arXiv preprint*, 2023. [13](#)
- A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2015. [6](#)
- J. Ericksen, M. Moses, and S. Forrest. Automatically evolving a general controller for robot swarms. In *IEEE Symposium Series on Computational Intelligence*, 2017. [6](#)
- H. Fan, X. Liu, J. Y. H. Fuh, W. F. Lu, and B. Li. Embodied intelligence in manufacturing: leveraging large language models for autonomous industrial robotics. *Journal of Intelligent Manufacturing*, 2024. [14](#)
- E. Ferrante, E. Duéñez-Guzmán, A. E. Turgut, and T. Wenseleers. Geswarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013. [6](#)
- D. Floreano and C. Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008. [6](#), [8](#)
- G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 2014. [1](#), [6](#), [7](#), [8](#)
- G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari. Automode-chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, 9(2), Sep 2015. ISSN 1935-3820. doi: 10.1007/s11721-015-0107-9. [9](#)

- M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 1937. 11
- D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 43
- H. Hamann. *Swarm robotics: A formal approach*. Springer, 2018. 1, 3, 4, 5, 6, 7
- H. Hamann and H. Wörn. A space- and time-continuous model of self-organizing robot swarms for design support. In *First International Conference on Self-Adaptive and Self-Organizing Systems*, 2007. 1, 5
- H. Hamann and H. Wörn. A framework of space-time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, 2008. 1, 5
- G. Hinton. Distilling the knowledge in a neural network. *arXiv preprint*, 2015. 40
- S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997. 12
- N. Houlsby, A. Giurghi, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*. PMLR, 2019. 25
- R. A. Izzo, G. Bardaro, and M. Matteucci. Btgenbot: Behavior tree generation for robotic tasks with lightweight llms. *arXiv preprint arXiv:2403.12761*, 2024. 14
- J. Jandeleit. Creating and demonstrating a dataset for swarm mission generation from natural language, 2023. URL [https://github.com/StudentWorkCPS/swarm-descriptions-llm/blob/main/project\\_report.pdf](https://github.com/StudentWorkCPS/swarm-descriptions-llm/blob/main/project_report.pdf). Project report. 17, 19, 21
- A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. D. L. Casas, and W. E. Sayed. Mistral 7b. *arXiv preprint*, 2023. 14, 24
- N. Khurshid. *Towards Specifying Swarm-Based Systems using Categorical Modeling Language: A Case Study*. PhD thesis, Concordia University, 2011. 7
- J. Kuckling, A. Ligot, D. Bozhinoski, and M. Birattari. Behavior trees as a control architecture in the automatic modular design of robot swarms. In *International conference on swarm intelligence*. Springer, 2018. 2, 8, 10, 11, 17, 19, 20, 22, 24
- J. Kuckling, K. Hasselmann, V. van Pelt, C. Kiere, and M. Birattari. Automode-editor: A visualization tool for automode, 2021. Accessed: 2025-02-01. 10
- P. Li, V. Menon, B. Gudiguntla, D. Ting, and L. Zhou. Challenges faced by large language models in solving multi-agent flocking. *arXiv preprint arXiv:2404.04752*, 2024. 14, 15
- A. Lykov and D. Tsetserukou. Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model. In *2nd International Conference on Foundation and Large Language Models*. IEEE, 2024. 2, 14
- C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, and P. Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023. 13
- Z. Mandi, S. Jain, and S. Song. Roco: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024. 15
- S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods, 2022. URL: <https://github.com/huggingface/peft>. 25

- A. Martinoli et al. *Swarm intelligence in autonomous collective robotics: From tools to the analysis and synthesis of distributed control strategies*. PhD thesis, Citeseer, 1999. 5
- A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014. 9
- M. Mataric and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems*, 1996. 6
- D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014. 23
- T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint*, 2013. Introduces Word2Vec, a model for learning word embeddings. 12
- F. Mondada, M. Bonani, X. Raemy, et al. Idk epuck. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 2009. URL <http://www.e-puck.org/>. 9
- M. Mounsif, K. Zehnder, Y. Motie, and Z. Adam-Gaxotte. Swarmind: Harnessing large language models for flock dynamics. In *2023 10th International Conference on Soft Computing & Machine Intelligence (ISCMi)*, 2023. doi: 10.1109/ISCMi59957.2023.10458573. 14
- S. Nagavalli, N. Chakraborty, and K. Sycara. Automated sequencing of swarm behaviors for supervisory control of robotic swarms. In *IEEE International Conference on Robotics and Automation*. IEEE, 2017. 6
- NVIDIA. Enroot: A simple tool for containerized environments, 2025. URL <https://github.com/NVIDIA/enroot>. Accessed: 2025-02-01. 23
- M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In M. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, June 2018. doi: 10.18653/v1/N18-1202. 1, 12
- C. Pinciroli and G. Beltrame. Buzz: An extensible programming language for heterogeneous swarm robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2016. 1, 6, 7
- C. Pinciroli, V. Trianni, R. O’Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, et al. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 2012. 6, 8, 19
- P. Pueyo, E. Montijano, A. C. Murillo, and M. Schwager. Clipswarm: Generating drone shows from text prompts with vision-language models. *arXiv preprint*, 2024. 14
- A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. 12, 13
- A. Radford, J. W. Kim, C. Hallacy, et al. Learning transferable visual models from natural language supervision. In *Conference Proceedings*, 2021. CLIP model combines text and images. 14
- R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2024. 13, 25, 39
- C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987. 14



- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 1986. 6
- E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-30552-1. 3
- M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich. Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, 2020. 4
- F. Schweitzer. *Brownian Agents and Active Particles*. Springer Series in Synergetics. Springer Berlin, Heidelberg, 2007. doi: <https://doi.org/10.1007/978-3-540-73845-9>. 5
- B. Siciliano and O. Khatib. *Robotics and the Handbook*. Springer, 2016. 1
- J. Snape and D. Manocha. Navigating multiple simple-airplanes in 3d workspace. In *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010. 14
- K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 2002. 6
- A. Taloni, M. Borselli, V. Scarsi, C. Rossi, G. Coco, V. Scoria, and G. Giannaccare. Comparative performance of humans versus gpt-4.0 and gpt-3.5 in the self-assessment program of american academy of ophthalmology. *Scientific Reports*, 2023. 1
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. Proposes Transformer Architecture, consisting only of attention layers. 1, 12, 13
- L. von Werra, Y. Belkada, L. Tunstall, E. Beeching, T. Thrush, N. Lambert, S. Huang, K. Rasul, and Q. Gallouédec. Trl: Transformer reinforcement learning. <https://github.com/huggingface/trl>, 2020. 39
- C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989. 6
- J. Wei, Y. Tay, R. Bommasani, C. Raffel, et al. Emergent abilities of large language models. *arXiv preprint*, 2022a. Discusses emergent abilities in large language models. 12
- J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 2022b. 43
- F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1945. 39
- T. Wolf. Transformers: State-of-the-art natural language processing. *ACL Anthology*, 2020. 25
- N. Wu, B. Green, X. Ben, and S. O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020. 12
- A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39727-4. 23
- W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, et al. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning*. PMLR, 2023. 2
- W. X. Zhao, Z. Liu, H. Sun, Z. Lan, Y. Wang, S. Feng, R. Ren, H. Li, Z. Wang, H. Yuan, Y. Xie, Y. Zhang, J.-R. Wen, and D. Yin. A survey of large language models. *arXiv preprint*, 2023. 2, 11