Creating and Demonstrating a Dataset for Swarm Mission Generation from Natural Language

Julian Jandeleit
julian.jandeleit@uni-konstanz.de
Master Project Cyber-physical Systems (WS 2023/2024)
Cyber-physical Systems Group
Department of Computer and Information Science, University of Konstanz

January 15, 2024

Abstract

Generating a controller for robot swarms from a high-level description of a swarm mission is a nontrivial task. This project introduces a modular framework for constructing a dataset that pairs swarm mission descriptions in natural language with their corresponding configurations for a swarm simulator. The primary objective is to facilitate the finetuning of Large Language Models (LLMs) to use the finetuned model for conversion and thus address the micro-macro gap. The dataset architecture is based on modular mission elements that are composed together to form a diverse dataset. We implemented mission elements for arenas, lights, robots, and objectives, resulting in 80000 unique descriptions for eight types of scenarios.

To demonstrate our dataset, we finetune the LLM Mistral 7B to generate mission configurations for ARGoS from natural language descriptions. We evaluate the results and find a limit in the complexity of internal computations successfully done by the finetuned model and point out how the limit could be increased. We achieve an average bleu score of 0.88 after 400 steps of training and find that the model is capable of generating valid mission configurations. The model even fills in unspecified details, making it potentially feasible to train a model to generate controllers directly or be used by non-domain experts.

1 Introduction

Many tasks for single robot systems can be replaced by several simpler and cheaper robots when the robots collaborate as a swarm. Common applications include search and rescue, environmental monitoring, or communication and logistics. There, collective robotics has the opportunity to build more robust and adaptive systems that can scale comparatively easily with

the number of robots and size of the task (Schranz et al., 2020). Usually, a swarm consists of identical robots that share both hardware and software without a global coordinator. Thus, swarm behavior needs to emerge from local interactions alone. Describing a multi-agent system like robot swarms on a global level and deriving behaviors of individual agents is a challenging approach and can be called micro-macro gap (Hamann, 2018).

The micro-macro gap can be approached starting from the microscopic perspective, modeling an individual robot and deriving the global behavior e.g. from a statistical model originated in physics (Hamann and Wörn, 2008). However, this does not provide a straightforward way to achieve any desired swarm behavior by macroscopic declaration alone. Kuckling et al. (2018) generate swarm behavior by composing elementary behaviors into a behavior tree using an evolutionary algorithm. This way, only a library of basic behaviors needs to be present at the microscopic level, while the desired macroscopic behavior can be specified as the objective function for the evolutionary algorithm. Nevertheless, reward functions and configurations still need to explicitely be given by a domain expert.

Recently, the transformer architecture (Vaswani et al., 2017) has lead to many advancements in the field of natural language processing. While originally designed for language translation, it was quickly adapted to be used for other tasks like text classification and question-answering (Kenton and Toutanova, 2019), achieving better scores than humans on some tasks (Taloni et al., 2023).
Training transformers on a corpus of text results in building contextualized word representations, a high dimensional vector that gets assigned to every word. This way, a pre-trained transformer can internally represent a text sequence as a sequence of features describing its semantic meaning. By training a transformer model that has enough parameters with enough text, the internal features can be leveraged to generate text output that even solves more complex tasks than translation (Zhao et al., 2023). The process of adapting to a specific task is called fine-tuning and these models are then called Large Language Models (LLMs).

Large Language Models have already been used to solve tasks in robotics. Cao and Lee (2023) uses LLMs to convert behavior trees across domains, like car manufacturing to PC assembly. Bucker et al. (2023) demonstrates the use of transformers to modify existing trajectories of a robotic arm in real world by encoding both language, image and geometry information. Lykov and Tsetserukou (2023) finetunes a LLM to generate a 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, that then can be used to optimize the controller of a robot. This previous research opens up the question if Large Language Models can also be applied to solve the micro-macro gap.

By generating configuration for the automated design method by Kuckling et al. (2018) with LLMs, we could obtain a link from the declarative description of a swarm behavior at a macroscopic level to the robot controller generated by the evolutionary algorithm at the microscopic level. A user would specify the swarm mission in natural language. This description is passed into an LLM that was fine-tuned to output the configuration file for ARGoS (Pinciroli et al., 2012). There, the actual controller can be obtained using evolutionary search, establishing the correspondence and link from description to controller. The necessary configuration is the environment and setting the swarm operates in, as well as the parameters for the objective function to generate the desired behavior. Finetuning an LLM for this task requires a dataset consisting of mission descriptions in natural language and the configuration for this mission in XML as expected by the optimizer.

In this work, we describe and implement a framework for how such a dataset of missions and their descriptions can be created. We demonstrate and evaluate how it can be used to generate configuration for automated design using Large Language Models and outline its use for generating swarm behavior directly from natural language.

2 Dataset Construction

Creating a dataset for swarm missions and their descriptions requires a common data model where the configuration and the description of the mission can both be derived from. Bozhinoski and Birattari (2021) describe a swarm modeling language *SML* that formalizes the components swarm missions can consist of. We adapt and restrict their model to only include the parts for the automated design software AutoMoDe-`Maple` by Kuckling et al. (2018), that we individually want to vary. While SML defines an environment that can contain many different types of elements, we restrict it to the arena. Different types of environment elements are explicitly stated at the mission level and objects required for the objective function are bundled with the objective. Here, we use the following model of a swarm mission:

- The **environment** where the mission takes place. It defines the size of the area, as well as the walls surrounding it. It is also called **arena**.
- The **lights** that are placed inside the arena.
- The **robots** or **swarm** placed inside the arena.
- The **objective**, the desired behavior the swarm should perform, and related objects necessary for achieving the objective.

The dataset we build consists of a set of these swarm missions as pairs of a configuration for AutoMoDe and several fitting natural language descriptions.

2.1 Data Model for Swarm Missions

We model a swarm mission as a recursive data structure of mission elements. We define a mission element ME that has the following properties:

- Parameters P, containing all variables necessary to fully define ME.
- A function *sample* that creates an instance of ME by randomly assigning reasonable values to P
- A function *describe* that uses parameters in P to return a set of valid descriptions for ME.
- A function *configure* that uses P to instantiate a class that can be serialized to an XML configuration.
- A set of children C, containing mission elements, so that the functions *sample*, *describe* and *configure* can consist of the combination of the respective children's functions.

Now, a swarm mission M is a mission element ME with a specific set of children C_M that reflects the variable components in the mission descriptions for AutoMoDe-`Maple`. We define

$$C_M = \{ME_{Arena}, ME_{Lights}, ME_{Robots}, ME_{Objective}\}$$

, where ME_T is a mission element such that *configure* returns an object that represents the component T of a mission and can be serialized to an XML configuration.

Note that C_M can change depending on the use case and actual simulation framework being used. This data model allows us to introduce diversity in our dataset. Each mission element can sample multiple different sets of parameters. Further, each has its distinct set of descriptions from which one gets sampled. On the mission level, there is diversity by instantiating different sets C_M. For example, the arena element ME_{Arena} could be an element that creates walls arranged in a circle, or an element that surrounds the environment with walls creating a rectangle. Both kinds are a valid ME_{Arena}, as long as their *configure* function can serialize to an *Arena* representation in the XML.

Let $|ME_T|$ be the number of existing kinds of mission elements serializing to T (e.g. $|ME_T| = 2$ for $T = Arena$ in our example of circular and rectangular walls). Let $N_{describe}$ be the mean number of different descriptions every existing mission element ME_T generates. Then there are $N_{configuration} = |ME_{Arena}| \cdot |ME_{Lights}| \cdot |ME_{Robots}| \cdot |ME_{Objective}|$ different types of mission scenarios and approximately

$$N_{description} = N_{configuration} \cdot N_{describe}{}^{|C_M|}$$

different descriptions, not including the different values the parameters can embody.

When interpreting the resulting dataset as features and labels for training a machine learning algorithm like generative language models, a model needs to understand the connection between a description and the respective configuration. So $N_{description}$ is the number of unique data points in the dataset if each combination of possible descriptions is sampled once.

2.2 Data Model Implementation

We implemented the following mission elements:

- **Arena:**
  - **CircularArena** with parameters $P = (height, radius, num\_walls)$, representing an environment that is enclosed by *num_walls* walls of height *height* arranged in a circle with radius *radius*.
  - **RectangularArena** with parameters $P = (length, width, height)$, representing an environment that is enclosed by 4 walls of height *height* composed to an rectangle of length *length* and width *width*.
- **Lights:**
  - **UniformLights** with parameters $P = (lights)$ and $lights = \{l_i \mid l_i = (x_i, y_i, intensity_i)\}$, where $|lights|$ lights are distributed uniformly inside the environment.
- **Robots:**
  - **CenteredSwarm** with parameters $P = (radius, num\_robots)$, representing a swarm where *num_robots* robots are placed uniformly inside a radius of *radius* from the origin.
- **Objective:**
  - **Aggregation** with parameters $P = (target, ground_1, ground_2)$ which places two circles $ground_i$ in the arena. One circle is black and the other is white. The objective of the swarm is to aggregate at $ground_{target}$.
  - **Connection** with parameters $P = (start, range, ground_1, ground_2)$. This objective also places a black and a white circle on the ground. The robots should form a line connecting both circles, beginning from $ground_{start}$. Two robots are connected within the range *range* of each other.
  - **Foraging** with parameters $P = (start, target, ground_1, ground_2)$. Here again, a white and black circle is placed inside the arena. The objective of the robots is to simulate foraging by repeatedly moving from $ground_{start}$ to $ground_{target}$.

- **Distribution** with parameters $P = (range, width, length)$. The swarm has to dispense into a rectangle with dimensions *width* by *length* while staying connected. Two robots count as connected within a range of *range*.

Sampling As the parameters of the different mission elements are chosen on different conceptual levels, the sampling function has varying complexity. While the *CenteredSwarm* element defines only the number of robots and their radius, the *UniformLights* element needs to compute the exact location for each light during sampling. Additionally, the sampling is done in a way that results in reasonable parameter configurations. For example, the positions of lights are only sampled inside an area of the environment that fits inside the arena. This results in a variety of levels of abstraction in the final dataset that the LLM has to learn.

Descriptions Every mission element has a set of 10 different descriptions. Some descriptions incorporate all element parameters, while others intentionally leave out some variables. In cases where the parameters contain very specific information like the *UniformLights* element, some descriptions also contain only summarized reports of variables like the maximum and minimum intensity of the lights. This should introduce more diversity in the kind of descriptions present in the dataset. It is intended to help the LLM to generalize from simply translating explicit values to the more abstract concept of describing a scenario that should be represented as XML configuration.

Configuration The design methods of the AutoMoDe family including AutoMoDe-`Maple` run ARGoS to perform simulations. ARGoS requires an `.argos` file containing specific XML as the configuration of the mission. Some parts of this XML are boilerplate required to set default configuration like the physics engine or parts that do not change within the different missions, like configuration for the type of robot that is used. Here, this is the e-puck platform (Mondada et al., 2009). We insert placeholders into a template `.argos` file to mark where the respective configuration needs to be placed.

Each *configure* returns an object that can be serialized to XML, containing all required parameters. The result of the complete missions M *configure* method also returns such an object. It contains all parameters necessary to represent the complete mission with all child mission elements as XML. It has a fixed defined datatype so that it can be serialized in a way that matches the placeholders defined in the `.argos` template. The XML only contains the variable parts of the templates and is called *parameter configuration*. The parameter configuration is transformed into the ARGoS

configuration by replacing each placeholder in the template with the respective element in the parameter configuration. The resulting `.argos` file can now be run by AutoMoDe-`Maple`. Note that the conversion from the parameter configuration is only dependent on itself. So for an LLM it is enough to generate *parameter configuration* to generate a working `.argos` file.

Dataset Sampling Finally, the dataset is obtained by first constructing a mission object M. M is sampled by first randomly choosing a mission element ME_T for each type of child in C_M. The mission parameters get sampled by calling *sample* on M which in turn samples the parameters for each child. Then the parameter configuration gets derived using *configure*. The result are the correct configuration parameters in XML. A description gets randomly chosen by the descriptions created using *describe*. Remember that *configure* and *describe* rely on the sub-solutions built by the respective functions of Ms child elements.

This process gets repeated $N_{dataset}$ times, creating a dataset with $N_{dataset}$ data points. Each data point consists of a sampled mission M, its parameter configuration, and a randomly chosen description. With two different environment types and 4 different objectives, we get a dataset size of $N_{description} = (2 \cdot 1 \cdot 1 \cdot 4) \cdot 10^4 = 80000$ unique mission descriptions. Each dataset row contains independently sampled parameters for its mission.

3 Large Language Model Finetuning

In this section, we describe how we use the dataset created in the previous section to finetune a Large Language Model on the task of generating configuration parameters from the description of a swarm mission.

We use the Large Language Model Mistral 7B (Jiang et al., 2023) in its *instruct-v0.2* version as the basis for our finetuning. It is available under an Apache License and has similar and higher performance than the larger Llama 2 13B model in many benchmarks. We load Mistral into memory using 4-bit quantization. To make finetuning feasible on smaller GPUs, we make use of the LoRA Technique (Hu et al., 2021), that reduces the number of trained parameters by only training rank decomposition matrices that get injected into each layer of the model. The pipeline is implemented in pytorch (Paszke et al., 2019) using the transformers library (Wolf et al., 2019). We use the LoRA implementation provided by the PEFT library (Mangrulkar et al., 2022) and train using the supervised fine-tuning module in the library TRL (von Werra et al., 2020).

To use our dataset for supervised training, it needs to be preprocessed. The description gets concatenated with the string *Generate the XML configuration for this mission* to mark the kind of desired result independent of

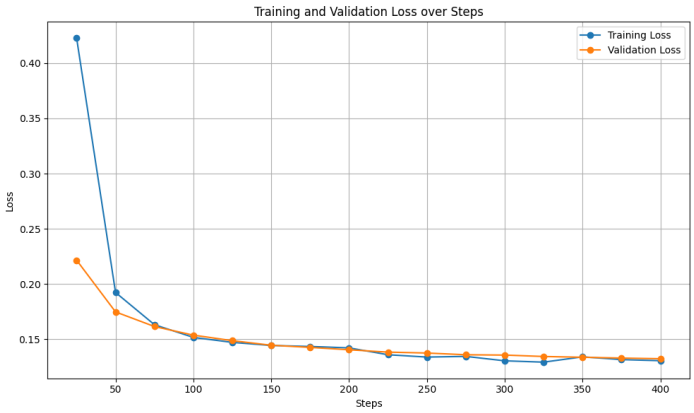

Figure 1: Training and validation loss by checkpoint.

the actual description. The Mistral LLM assumes its instruction to be enclosed in special tokens `[INST]` and `[/INST]`. Additionally, the string should start with the special token `<s>`. Mistral will generate new tokens until the special token `</s>` is generated or a maximum number of tokens is reached. For training, the target output is expected to already be part of the train. So after preprocessing, we have data points converted to text in the following form:

```
<s>[INST]{DESCRIPTION}\nGenerate the xml configuration for this mission.[/INST]{CONFIGURATION}</s>
```

Here {`DESCRIPTION`} and {`CONFIGURATION`} represent the original columns in our dataset.

We finetune the Mistral model on our preprocessed dataset for 400 steps with a learning rate of $2 \cdot 10^{-4}$, the optimizer *paged_adamw_32bit* and weight decay of 0.001. The number of actually trained parameters is reduced to 0.56% by using LoRA. The hardware is an Intel Xeon CPU which runs at 2.00 GHz and a NVIDIA Tesla P100 GPU with 16 GB VRAM. Training takes 7.5 h to complete. Checkpoints of the finetuned model are saved every 25 training steps for later evaluation.

For evaluation, we generate configuration parameters for 14 new descriptions each checkpoint. In this section, we will evaluate the results with three different methods quantitatively and also perform a manual qualitative assessment. Additionally, the supervised trainer reports training and validation loss for these checkpoints. The default metric is mean squared error (MSE). It is reported in Figure 1.

We see that the training loss quickly decreases. While validation loss also decreases, it starts lower and aligns with the training loss after about 100 training steps with a loss of roughly 0.15. After 200 steps, the training

loss starts to be slightly lower than the validation loss, indicating overfitting. Nevertheless, the validation loss is monotonically decreasing and aligns with the training loss again at step 400 at its minimum of 13.24. This might indicate that a lower minimum can be reached without overfitting by training the model for more than 400 steps.

3.1 Quantitative Evaluation

The generated configuration parameters first need to be converted to valid `.argos` files. This is an entirely deterministic procedure as described in section 2. However, if the LLM output contains syntactical errors, like missing or wrong tags, attributes, etc., this conversion will fail. For 14 generated missions at each checkpoint, we have a total of $14 * 16 = 224$ generated missions to be evaluated. Of those, 40 mission configurations were not successfully converted to `.argos` files. The share of invalid configuration parameters at each checkpoint is shown in Figure 2.

We then run every successfully converted mission in ARGoS 3 beta 48[1]. A run will fail if there is a syntactical error in the ARGoS configuration. ARGoS will also fail if specific errors on the semantic level are encountered. For example, robots could enter space outside of the defined environment by spawning at undesired locations or leaving the scene due to wrongly placed walls. There are a total of 77 missions that ARGoS did not run successfully. Notably, 78% of those missions include the *CircularArena* mission element. The shares of failed ARGoS runs and *CircularArena*s each checkpoint are shown in Figure 2. There are a total of 107 missions that were successfully generated by the finetuned LLM over all checkpoints. It is a total of 48% of all generated missions. The last checkpoint has 57% of all missions that successfully could be simulated in ARGoS. Finally, we calculated the bleu score (Papineni et al., 2001) for each generated response. It was introduced for machine translation and is commonly used to evaluate generated output by language models. It measures how many n-grams (sequences of n tokens) in the generated output appear in the reference output, which here is the correct configuration parameters XML, generated during dataset sampling in Section 2.2. An identical output has a bleu score of 1.0. For our evaluation data, we achieve an average score of 0.88 with variance of 0.024. The bleu score is reported by checkpoints shown in Figure 2 as well.

3.2 Qualitative Evaluation

After manual investigation, there are several reasons for generated configuration parameters that cannot be converted. In the earlier checkpoints, there often is an XML tag missing that specifies the dimensions and dimensions

[1]For running a mission, we packaged an ARGoS and AutoMoDe installation in an Ubuntu docker container, running on a Fedora host machine.

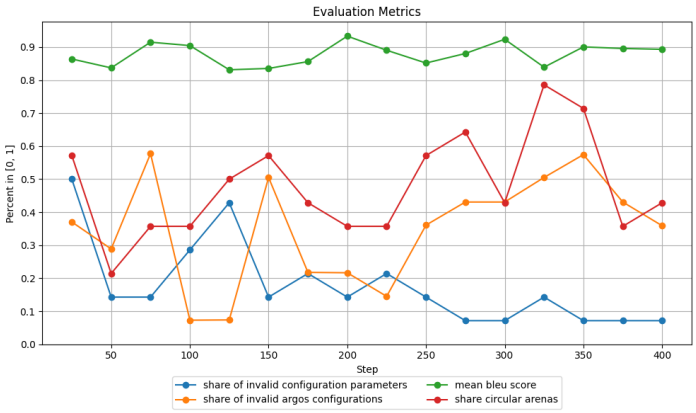

Figure 2: This Figure shows that the share of invalid configuration parameters decreases with the number of training steps to below 10%. This is expected as this is what the LLM was directly trained on. While the bleu score is already relatively high after 25 steps, it continues to increase slightly with fluctuations. The share of `.argos` file that was not able to run successfully varying largely and is high, when the share of circular arenas is high. The share is calculated relative to the number of configuration parameters that were successfully converted.

of the environment. Without this tag, the ARGoS configuration cannot be generated and fails. In missions that have many elements like lights and walls placed inside the environments, the conversion also fails. The conversion fails because the generated XML is not complete. When generating responses from a LLM you specify a maximum number of tokens to be generated. If the closing token `</s>` is not generated within this number of tokens, the generation will stop. For this evaluation, the maximum number of tokens was set to 2500, which may not be enough for some missions. However, setting a higher value may result in longer execution times. The 224 mission configurations were generated in 8 hours in our execution environment. So, if more computation time is available, the maximum number of tokens can be increased. This way, the number of invalid configuration parameters after 400 training steps, can be minimized.

When visually investigating the runs of a mission in ARGoS, it becomes clear why some ARGoS missions fail, especially those that contain circular walls. In the descriptions, the exact positions and orientations of the walls are not specified, as discussed in section 2. Placing walls of a rectangle correctly requires 4 correct positions and 4 correct orientations, while 2 walls are symmetrical each. Also, the walls are aligned with the coordinate system. This is no longer the case for circular arranged walls. Also, the correct length of each wall needs to be calculated by the LLM, as well as the correct

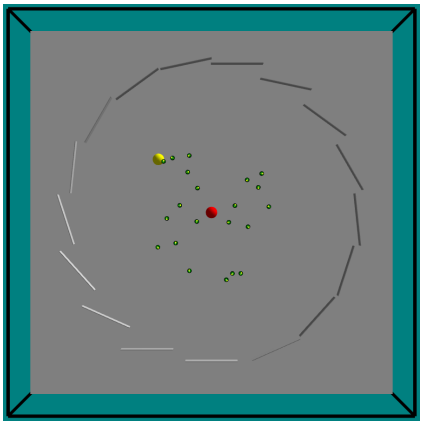

Figure 3: An example where a circular arena was not generated correctly (Image shows the initial positions of the robots.). This is the description the model was prompted with: *In this setting, a circular arena with a radius of 2.64 meters is established. 1 lights are evenly positioned throughout the arena, providing illumination. Their intensities are 6.98. Placed within a 1.21-meter radius around the center are 21 robots. The swarm's goal is to cover an area of 0.55 by 1.29 meters while ensuring connectivity.*

orientations to form a circle. In most cases, the LLM can correctly compute the positions of the walls. However, in all manually investigated cases that failed, the orientation of the walls was wrong. This also holds for missions containing rectangular-shaped walls that failed.

Figure 3 shows an example of the most common scenario encountered when manually evaluating failed runs. We can see that there are 21 robots correctly placed inside the arena (green). Further, there is a light in the arena (yellow). The red sphere represents a default light with no effect on the robots, required to successfully run ARGoS, and is injected when the configuration parameters are converted to the ARGoS configuration. The walls are correctly placed in a circular shape. However, their size as well as orientation are not specified correctly, leading to gaps between the walls where robots can pass through. This way robots can leave the defined environment marked in grey, leading to a crash and failed run.

Although not present in the image in Figure 3, we can see by manually inspecting the converted XML that the objective of the swarm mission was encoded correctly:

```
<objective type="distribution">
  <objective-params area="0.55,1.29" connection_range="0.25"/>
</objective>
```

You can see that the LLM correctly specified the mission type by deriving it from the description. It also has chosen a sensible connection range without a value being present in the description. This means the fine-tuned model is capable of deriving related information as well as filling in missing data. In this example, *connection_range* is missing data as it was not described in the instruction of the LLM. The instruction is presented in Figure 3. The effect of filling in missing data is also visible in the example presented in Figure 4. It shows a stereotypical configuration generated by the fine-tuned model that can successfully run in ARGoS. Here, the missing data is the radius for both circles. It is not mentioned in the mission description presented in Figure 4. The description only mentions the positions of each circle. The model generates the following XML nodes for the circles:

```
<circle position="1.74,-2.00" radius="0.45" color="white"/>
<circle position="0.89,1.43" radius="0.45" color="black"/>
```

You can see that both circles are assigned a radius of 0.45. It appears that the model has learned that circles need the parameter *radius* even if not mentioned in the mission description. When comparing the circles with the environment, you can see that the size was chosen reasonably so that the circles fit inside.

Note that in this mission configuration, the walls are rotated correctly and placed at the boundaries of the environment shown in gray.

4 Discussion

The evaluation shows that the finetuned model can successfully generate configuration files for the ARGoS simulator from a description of the mission in natural language. The model can derive and supplement missing information even if not present in the text. Also, the model can even perform simple computations, for example for rotating and aligning walls to a rectangle of specified dimensions. However, there seems to be a limit to the complexity of computations the model can perform to sufficient precision, for example when creating a circular arena. This limit may be a property of the base model and the number of available parameters. It could also be a property of the reduced number of training parameters and limited number of 400 training steps. If this is the case, choosing a larger base model and training it for more steps should increase the limit to a degree where the finetuned model may be able to correctly place walls to form a circle.

The complexity limit for internal computations can potentially also be increased by changing the training method and loss computation. As the LLM outputs a sequence of text that should represent XML in our case, it produces output that is often repetitive, like having to individually write

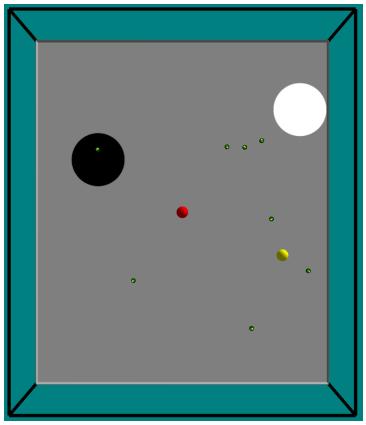

Figure 4: A working configuration generated from the following description (Image shows the initial positions of the robots): *In this setting, a rectangle is formed with dimensions 4.92 x 5.77 x 2.62. The arena features 1 lights: (-0.73, -1.70, 4.43). Within a 2.36-meter radius from the center, 8 robots are uniformly distributed. In the floor space, you'll discover two distinct areas: a circle at [1.74, -2.00] in white, and another circle at [0.89, 1.43] in black. The primary objective for the robots is to aggregate at the black circle.*

an XML tag for each light that is present in the scene. These tags require multiple tokens without introducing much new relevant information. The important new information lies in the individual placements of each light, wall, etc. This creates an instance of class imbalance, where the important information is underrepresented in the target output so that the model can achieve relatively high scores only by comparing the generated output with metrics like MSE, ignoring the actual values inside the XML attributes. A solution could be to change the training to optimize a reward function that is composed of metrics comparing the structure of the generated XML and the values inside the attributes with equal weight.

As the model can fill in many parameters, it can probably be used by non-domain experts to specify swarm missions for ARGoS, if the types of supported missions are communicated to the user. By optimizing the robot controllers in AutoMoDe, we obtain a dataset of mission descriptions and behavior trees implementing the required behavior. These correspondences might be enough to train a Large Language Model to directly generate robot controllers for specified swarm missions, effectively bridging the micro-macro gap, starting from the macroscopic perspective. In this application scenario, it would make sense to filter invalid configurations in advance.

The types of missions that can be generated by the finetuned model of course depend on the mission elements present in the dataset and can be

easily modified due to the modular architecture of our dataset framework.

5 Conclusion

In this project, we introduced a framework for creating a dataset of swarm mission descriptions in natural language and their configurations for a simulator. The motivation is to use the dataset to finetune a Large Language model that bridges the micro-macro gap from the swarm level to the level of the robot controller. The robot controller can be obtained by optimizing a behavior tree for a given configuration with AutoMoDe.

The dataset is constructed in a modular and extensible way. Each mission consists of specific mission elements, whose instances are interchangeable. Each mission element specifies how it is represented in XML and how it can be described using natural language. As the mission is a mission element itself, it can be also represented as both, by recursively incorporating its children's sub-solutions. We implemented a mission consisting of an arena, lights, robots, and an objective. We implemented two types of arenas and 4 types of objectives.

We demonstrated the dataset by using it to finetune the Large Language Model Mistral 7B. The evaluation showed that the finetuned model can successfully generate mission configurations for ARGoS from a description of the mission in natural language. The evaluation also showed that the model can fill in unspecified details and compute derived values like the exact placement of walls. However, there is a limit to the complexity of internal calculations that can be done by the model to sufficient precision. We hit this limit when requiring the model to arrange walls to form a circle. We discussed the results and pointed out how to overcome this limitation by increasing training time and parameters. Also, we propose an alternative training reward that captures the importance of individual parameter values.

Finally, finetuning a Large Language Model to overcome the micro-macro gap, appears promising, and the dataset we created lays the foundation for further research. The dataset is available online[2]. Future investigations will reveal whether this method allows non-domain experts to configure swarm missions or empowers LLMs to generate appropriate robot controllers directly without the need for evolutionary optimization in between.

[2]The dataset can be accessed at `https://github.com/julianjandeleit/swarm_descriptions/tree/v1.0.0`.

References

Bozhinoski, D., Birattari, M., 2021. Towards an integrated automatic design process for robot swarms. Open Research Europe 1.

Bucker, A., Figueredo, L., Haddadin, S., Kapoor, A., Ma, S., Vemprala, S., Bonatti, R., 2023. Latte: Language trajectory transformer. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 7287–7294.

Cao, Y., Lee, C., 2023. Robot behavior-tree-based task generation with large language models. arXiv preprint arXiv:2302.12927.

Hamann, H., 2018. Swarm robotics: A formal approach. Vol. 221. Springer.

Hamann, H., Wörn, H., 2008. A framework of space–time continuous models for algorithm design in swarm robotics. Swarm Intelligence 2, 209–239.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Wang, L., Chen, W., et al., 2021. Lora: Low-rank adaptation of large language models. In: International Conference on Learning Representations.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al., 2023. Mistral 7b. arXiv preprint arXiv:2310.06825.

Kenton, J. D. M.-W. C., Toutanova, L. K., 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT. pp. 4171–4186.

Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M., 2018. Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Swarm Intelligence: 11th International Conference, ANTS 2018. pp. 30–43.

Lykov, A., Tsetserukou, D., 2023. Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model. arXiv preprint arXiv:2305.19352.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., Bossan, B., 2022. Peft: State-of-the-art parameter-efficient fine-tuning. `https://github.com/huggingface/peft`, accessed: 2024-01-11.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., Martinoli, A., 2009. The e-puck, a robot designed for education in engineering. In: Proceedings of the 9th conference on autonomous robot systems and competitions. Vol. 1. IPCB: Instituto Politécnico de Castelo Branco, pp. 59–65.

Papineni, K., Roukos, S., Ward, T., Zhu, W.-J., 2001. Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02. ACL '02. Association for Computational Linguistics.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., et al., 2012. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm intelligence 6, 271–295.

Schranz, M., Umlauft, M., Sende, M., Elmenreich, W., April 2020. Swarm robotic behaviors and current applications. Frontiers in Robotics and AI 7.

Taloni, A., Borselli, M., Scarsi, V., Rossi, C., Coco, G., Scorcia, V., Giannaccare, G., 2023. Comparative performance of humans versus gpt-4.0 and gpt-3.5 in the self-assessment program of american academy of ophthalmology. Scientific Reports 13 (1), 18562.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.

von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S., 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., Rush, A. M., 2019. Huggingface's transformers: State-of-the-art natural language processing.

Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Arenas, M. G., Chiang, H.-T. L., Erez, T., Hasenclever, L., Humplik, J., et al., 2023. Language to rewards for robotic skill synthesis. arXiv preprint arXiv:2306.08647.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al., 2023. A survey of large language models. arXiv preprint arXiv:2303.18223.