

# BTGenBot: Behavior Tree Generation for Robotic Tasks with Lightweight LLMs

Riccardo Andrea Izzo  
Politecnico di Milano  
Milan, Italy  
riccardo.izzo@mail.polimi.it

Gianluca Bardaro  
Politecnico di Milano  
Milan, Italy  
gianluca.bardaro@polimi.it

Matteo Matteucci  
Politecnico di Milano  
Milan, Italy  
matteo.matteucci@polimi.it

**Abstract**—This paper presents a novel approach to generating behavior trees for robots using lightweight large language models (LLMs) with a maximum of 7 billion parameters. The study demonstrates that it is possible to achieve satisfying results with compact LLMs when fine-tuned on a specific dataset. The key contributions of this research include the creation of a fine-tuning dataset based on existing behavior trees using GPT-3.5 and a comprehensive comparison of multiple LLMs (namely llama2, llama-chat, and code-llama) across nine distinct tasks. To be thorough, we evaluated the generated behavior trees using static syntactical analysis, a validation system, a simulated environment, and a real robot. Furthermore, this work opens the possibility of deploying such solutions directly on the robot, enhancing its practical applicability. Findings from this study demonstrate the potential of LLMs with a limited number of parameters in generating effective and efficient robot behaviors.

**Index Terms**—robotics, robot behavior, behavior tree, large language models

## I. INTRODUCTION

In recent years, robots have become an integral part of our everyday lives, permeating various sectors such as logistics, manufacturing, and healthcare. The increasing prevalence of robots in these diverse fields shows the necessity to go beyond a simple set of functionalities and achieve adaptable, flexible, and effective task planning. This is necessary to tackle the challenges of modern robotics, such as dynamic and unstructured surroundings, unpredictable situations, and advanced interaction with humans and the environment. To achieve this, various representations have been proposed to describe robot tasks [6] and multiple planning languages [3] have been designed to let experts write complete and descriptive definitions of high-level tasks.

An example of these representations is behavior trees (BTs). Behavior trees have gained significant traction in the field of robotics, offering a structured and scalable approach to managing complex robot behaviors. Originating from the video game industry, BTs have found relevance in robotics due to their ability to handle high-level decision-making and control in dynamic environments. A key player in this transition is the BehaviorTree.CPP library<sup>1</sup>. This library has become the de facto standard in the Robot Operating System 2 (ROS2) ecosystem, largely due to its inclusion in Navigation2 [12].

This integration has facilitated the development of sophisticated navigation behaviors, contributing to the broader adoption and standardization of BTs in robotics software design.

However, even after a long history of efforts, there is no definitive solution to equip robots with flexible and adaptable task planning, since it requires complex knowledge, such as the robot’s actions, the dynamic nature of the environment, object relations, and affordances.

Recently, large language models (LLM) trained on large amounts of data have emerged as a promising tool to manage this challenge and deal with the complex knowledge required for task planning. These models have demonstrated remarkable success in robot planning from a set of natural language instructions. Additionally, an LLM can represent commonsense priors (e.g., visit the colder locations first) and comprehend spatial relationships (e.g., move behind the sofa). However, such large models present a significant limitation: their substantial hardware resource requirements prevent their direct deployment on robots.

In this work, we explore an alternative approach, focusing on compact large language models, specifically those with 7 billion parameters. We posit that these smaller models, while less resource-intensive, can generate behavior trees for robots with comparable effectiveness to their larger counterparts. Our main contributions are: (i) a dataset of 600 behavior trees paired with a natural language description of their tasks, (ii) a system to evaluate and validate the performance of the generated BT over nine different tasks including navigation and manipulation, and (iii) a comparative analysis of three different LLM and their fine-tuned versions. All the material used in this work, including the dataset, the complete prompts of both evaluation phases, and the source code of the validator are available at <https://github.com/AIRLab-POLIMI/BTGenBot>.

## II. RELATED WORK

A pre-trained foundation model (PFM) refers to a complex neural network-based model trained on extensive open-source datasets, boasting billions of parameters [23]. This robust architecture serves as a versatile backbone that can be further fine-tuned for diverse tasks across various domains. Recently, several PFMs specifically trained on massive text datasets (i.e., large language models) have been proposed such as the most recent one GPT-4 [1] by OpenAI or LLaMA by Meta AI [18].

<sup>1</sup><https://www.behaviortree.dev/>

In this work, we rely on LLaMA released by Meta AI, a family of foundation models trained on an open-source dataset with trillions of tokens. In particular, we employed LLaMA-2 [19], trained on 2 trillion tokens, and with a context length of 4000 tokens, double the length of its predecessor. Three model sizes are available: 7, 13, and 70 billion parameters.

**Behavior tree generation.** An example of behavior tree generation using LLM is the work done in [11]. In this paper, the authors exploit a transformer-based LLM fine-tuned from the Stanford Alpaca 7B model [17] to generate behavior trees from a text description. They further fine-tuned this model with natural language descriptions generated using *text-davinci-003* from behavior trees created in the previous step. In summary, they use two different LLMs to generate both the behavior trees and the task description. While interesting in proving that LLMs can be used to generate BTs, this approach has several limitations. Fine-tuning an LLM with a dataset generated by the LLM itself causes a data bias only partially mitigated by the use of an auxiliary LLM to generate the descriptions. Moreover, self-generated data can be noisy and, in this configuration, errors propagate from the data generation phase to the training phase. Differently from the solution proposed in [11], we create our instruction-following dataset by collecting behavior trees from open-source robotics projects. This allows us to fine-tune our model with BTs already tested and effectively used in real projects. Moreover, we use GPT-3.5 only to generate descriptions for each corresponding behavior tree.

Finally, concerning the validation of the above approach, the only metric used is the visual evaluation by a group of human experts. The evaluators were presented with a pair of behavior trees and had to indicate which one was generated. This evaluation, while interesting and useful to assess the general capabilities of the model, is rather limited. This method does not consider the correctness from a syntactical and semantic point of view, or any performance metric, such as the generation time and the hardware resources used. In contrast, we tested our system with a collection of metrics described in Section III.

**Code-based planning.** Another approach to defining robot behaviors using LLM revolves around the direct generation of executable code. An excellent example is Code as Policies [10]. In this work, the authors release a robot-centric formulation of language model generated programs (i.e., LMPs), based on hierarchical code generation prompting, that allows the generation of new policy code via recursively defining undefined functions. LMPs take advantage of few-shot prompting to generate different subprograms and can be generated hierarchically with chain-of-thought prompting. Even if their approach requires no additional training, their system is based on GPT-3 with 175B parameters. Moreover, they use open-vocabulary object detection models like ViLD [5] and MDETR [9] to compose perception-to-control feedback logic. Another work that relies on prompt engineering to create a Pythonic planner is ProgPrompt [16]. They introduce a programmatic LLM prompt structure that enables

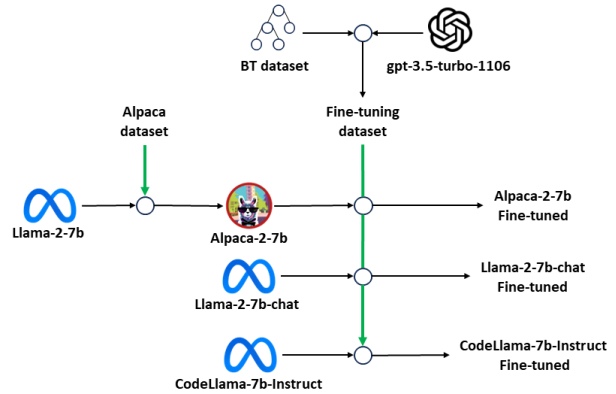


Fig. 1. Fine-Tuning Process

plan generation functional across situated environments and robot capabilities. Their fine-tuned version of GPT-2 can generate plausible action plans in the context of robotic task planning, this is achieved by providing the model with a Pythonic program-like specification of the available actions and objects in an environment as well as executable example programs. In this way, they introduce situated awareness in LLM-based robot task planning via prompting.

Both works described in [10] and [16] benefit from LLMs with over 100B parameters, while our approach instead leverages smaller models up to only 7B parameters. This allows our method to be replicable locally on consumer hardware without relying on larger, more complex models available via APIs. Additionally, the use of behavior trees gives more flexibility in terms of expressive ability, represents a plug-and-play solution as in [13], and is commonly used in recent open-source robotics projects as demonstrated by [8].

### III. METHOD

For the experiments in this work, we considered as reference models the state-of-the-art open-source large language models provided by Meta. We selected *Llama-2-7b* [19], the foundation model, *Llama-2-7b-chat* [19], designed for conversational tasks, and *CodeLlama-7b-Instruct* [15], with a focus on code-generation. All of these models are based on the transformer architecture pioneered in [20]. Following the approach proposed in [11], we fine-tuned the foundation model *Llama-2-7b* with the Alpaca dataset, with the procedure reported in [17] using the Alpaca-LoRA repository [21]. For the remainder of this paper, we will refer to this fine-tuned version as Alpaca. Additional fine-tuning of these models allows for customization to meet specific requirements, such as domain specialization, in our case the generation of behavior trees. A summary of the various fine-tuning steps is presented in Figure 1, and a more detailed description is provided in Section III-C.

#### A. Dataset Format

Each entry of our instruction-following dataset is composed of three parts: “instruction”, “input” and “output”. While the

original work of [17] states that “input” is optional, in our case, we combine the XML version of the behavior tree provided in “output” with the description of the task in the “input” to create a definition that matches the prompts used later during inference.

You will be provided a summary of a task performed by a robot, and your objective is to express this task as a behavior tree in XML format.

The behavior tree is a simple sequential task for a robot. It first instructs the robot to move to a specific point (GoPoint) and then to interact with a particular object (GoObject). The robot will execute these tasks in sequence, moving to the specified destination before interacting with the designated object.

```
<root main_tree_to_execute = "MainTree" >
  <BehaviorTree ID="MainTree">
    <Sequence name="root_sequence">
      <GoPoint goal="{destination}" />
      <GoObject target="{object}" />
    </Sequence>
  </BehaviorTree>
</root>
```

This is an example of an entry in the dataset. In gray, the “instruction” element is common to all the samples in the dataset, and it is fixed and immutable. It is followed by the “input” element, in blue, where a description in natural language is provided of the behavior tree. Finally, the “output” element represents the behavior tree in XML format. All the behavior trees used in the dataset are sourced via the work of [4]. In this work, the authors provide a collection of roughly 600 behavior trees collected from various open-source projects in the field of robotics. This dataset is particularly valuable because compatible with BehaviorTree.CPP, the de facto standard of ROS2, and the same format used by [11]. Furthermore, what distinguishes this dataset is the quality of the behavior trees, which have been developed for actual applications and tested on robots and thus have already been validated in real-world scenarios. The natural language description (i.e., the “input” element) of each entry of the dataset has been generated automatically using GPT-3.5, with a procedure described in the next section.

### B. Dataset Generation

Recent studies such as [22] and [2] highlight the exceptional capabilities of modern LLM, notably GPT, in several domains including question answering, text generation, and code generation. GPT models are suitable for text generation tasks [22] and can be exploited to generate a description of the provided input. Considering that, we used OpenAI available APIs to complete our instruction-following dataset. In particular, the “input” element (i.e., the description of the behavior tree) has been generated using the *gpt-3.5-turbo* model with default parameters and a context length of 2048 tokens. We prompted the model to generate a natural language description of a behavior tree in XML format received as input. The one-shot prompt used with *gpt-3.5-turbo* model to generate the description of the provided behavior tree is the same discussed in [17] and it is composed of the following elements: “system”, “user”, “assistant” and again “user”. The “system” element is shared between all prompts and represents

the general context given to the model: “You will be provided a behavior tree in XML format, and your task is to summarize the task performed by this behavior tree”. The above prompt represents only the starting point, additional information is given including the maximum number of words and the required compatibility with BehaviorTree.CPP library and the fact that the description must represent an overall summary of the task clearly described in natural language. This method ultimately leverages the inherent linguistic capabilities of GPT models to create a description of the behavior tree. To assess the quality of the generated descriptions, we sampled a subset of the results (i.e., roughly ten behavior trees) and evaluated visually how well the description matched the input BT. The result of our evaluation was positive, therefore we proceeded to generate the whole dataset using the procedure described before.

### C. Fine-Tuning Process

The models selected for the fine-tuning process are *Llama2-7b*, *LlamaChat*, and *CodeLlama-Instruct-7b*. The fine-tuning process can be structured in two steps, the first one involves the *Llama2-7b* model that has been fine-tuned with the original Alpaca dataset as reported in [7]. This step prepared the base model for instruction-following tasks, differently from the other models that are already fine-tuned for this task. The second step prepared the models for the generation of the behavior trees. To perform this task, we further fine-tuned the models using the dataset described in III-A. LLMs fine-tuning is a computationally intensive task, therefore, we employed the Parameter-Efficient-Fine-Tuning (PEFT) approach, in particular, Low-Rank Adaptation (LoRA) described in detail in [7]. As reported in [14], this is particularly useful to avoid retraining the entire model from scratch. In this way, we can keep the model frozen and add an adapter at the end of the model consisting of a few learnable parameters, and layers.

Regarding LoRA parameters, most of them were set at their default value, except for the target modules. In the work [7], it is noted that the transformer architecture has four weight matrices in the self-attention module ( $W_q, W_k, W_v, W_o$ ). To enhance generalization capabilities with our limited dataset and to optimize performance, it was necessary to unfreeze the MLP layers. We expanded the corresponding initial set of weight matrices [q\_proj, k\_proj, v\_proj, o\_proj] by adding the additional weight matrices of the MLP module, these include [gate\_proj, up\_proj, down\_proj]. This approach led to a more robust and effective training process.

The fine-tuning process was accomplished with two NVIDIA RTX Quadro 6000 with a total of 48GB of video memory. All the basic hyperparameters have been used except for a batch size of 256 and a micro-batch size of 4, we used a learning rate of  $3e-4$  and a validation set size of 5%. Given that our dataset is composed of less than a thousand samples, we increased the number of epochs to achieve satisfying results.

## IV. EVALUATION

To assess and compare the performance of the three base models (i.e., Alpaca, LlamaChat, and CodeLlama) and our fine-tuned versions, we conducted different tests using nine task descriptions of behavior trees.

### A. Task definition

All three models were tested, with the input being the provided description of the behavior tree. The expected output is a behavior tree in XML format, compatible with BehaviorTree.CPP library, which reflects the provided description. The tasks used are the following:

- 1) **Navigation.** The robot is tasked to reach a series of locations provided as coordinates in a specific order.
- 2) **Navigation with priority.** The system receives a list of locations and a list of corresponding readings (e.g., temperature). Given a threshold, the robot must visit all locations prioritizing those with a reading above the threshold.
- 3) **Navigation with fallback.** The robot must navigate through a series of waypoints. During the navigation, a waypoint may become unreachable. In this case, the destination must be skipped and the robot should proceed to the next waypoint.
- 4) **Navigation with arm activity.** An extension of the Navigation task, at each location the robot activates the on-board manipulator.
- 5) **Exploration.** In this situation, the robot navigates continuously. Periodically, the robot receives a new location and checks if the exploration routine is completed.
- 6) **Manipulator exploration.** A task performed by a manipulator. The system cycles through multiple joint configurations until a target object is found. When found, the manipulator approaches the target.
- 7) **Active vision and picking.** A robotic arm observes an object, estimates a grasping position, and performs a "pick and place" routine.
- 8) **Material processing.** A robot manipulator triggers material processing by pressing buttons in the correct sequence. The robot is also in charge of evaluating the status of the processing.
- 9) **Multi-station assembly.** A mobile manipulator moves between multiple stations to collect components and assemble an object.

### B. Evaluation approach

The evaluation has been accomplished in two separate phases: the first one represents a preliminary selection among the models while the second one effectively validates the models and the generated behavior trees with more rigorous metrics. In the first phase, we evaluated the models on a subset of simpler tasks, namely tasks from 1 to 7. In the second phase, where the general capabilities of the models are already been assessed, all nine tasks are considered. The metrics considered in the first evaluation phase are:

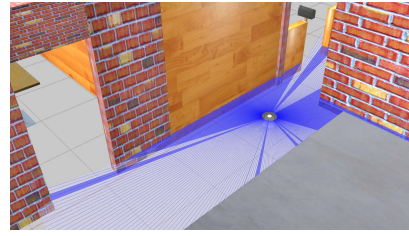


Fig. 2. Simulation Environment using TurtleBot3 on Gazebo



Fig. 3. Robot used to test the generated behavior trees

- **Time:** assess the time performance of the models, this is crucial when the entire pipeline, from the LLM to the actual robot, is considered.
- **Syntactic Correctness:** evaluating the syntactic accuracy using Groot2, an IDE to create and debug behavior trees provided by BehaviorTree.CPP. This allows us to check the syntactic correctness of the XML schema and the overall tree.
- **Semantic Evaluation:** without considering the syntactic correctness, we evaluate the semantic accuracy. This means the ability of the LLM to generate a solution that understands and solves the problem at hand. This evaluation is performed by a group of human experts.

Moving on to the second evaluation phase, we considered the following metrics:

- **BT Correctness:** validating the behavior trees using a custom-developed behavior tree validator
- **Simulation robot:** verifying the successful execution of the task on a simulated environment, in particular, the navigation task has been tested on the TurtleBot3 by ROBOTIS. See Figure 2.
- **Real robot:** verifying the successful execution of the task on the physical robot shown in Figure 3.

### C. Preliminary selection

Table I shows the time comparison of execution time while doing inference on the model, the tests have been conducted on an NVIDIA RTX Quadro 6000. Considering the models without fine-tuning. When using a zero-shot approach, CodeLlama overall is faster than LlamaChat, having a similar mean but less standard deviation. In general, Alpaca is slower and task

TABLE I  
TIME COMPARISON

	Alpaca	LlamaChat	CodeLlama
Base, Zero-Shot			
Mean	-	3m 10s	2m 52s
Std. Dev.	-	1m 29s	1m 57s
Fine Tuned, Zero-Shot			
Mean	-	1m 13s	1m 15s
Std. Dev.	-	56s	58s
Base, One-Shot			
Mean	1m 42s	3m 54s	1m 49s
Std. Dev.	46s	38s	1m 16s
Fine Tuned, One-Shot			
Mean	2m 10s	1m 25s	1m 21s
Std. Dev.	1m 20s	58s	1m 11s

TABLE II  
SYNTACTIC EVALUATION (PHASE 1)

	Alpaca	LlamaChat	CodeLlama
Base, Zero-Shot			
Fine Tuned, Zero-Shot	14%	71%	86%
Base, One-Shot			
Fine Tuned, One-Shot	86%	86%	86%

2 and task 7 do not terminate, therefore, it was not possible to compute the mean execution time. One-shot prompts are slightly slower since they have to deal with a more complex input, CodeLlama and LlamaChat have comparable mean and standard deviation similar to their zero-shot counterparts. It is worth noting that LlamaChat becomes rather consistent (i.e., low standard deviation) between tasks when using a one-shot prompt. In this case, Alpaca receives a significant performance boost, since it can complete all tasks, however it remains slower in most cases.

Regarding syntactic correctness, we define a behavior tree to be syntactically correct if successfully recognized by Groot2. Groot2 checks the correctness of the overall tree for what concerns Nodes, Decorators, and parameters. All the outputs of three base models and the corresponding fine-tuned version with our dataset, both with zero-shot and one-shot prompting, have been evaluated using Groot2. The results of this evaluation are summarised in Table II.

In terms of accuracy, base models with a zero-shot prompt achieve very limited results, with only CodeLlama being able to generate a syntactically correct behavior tree in a very limited number of cases. Adding examples in the prompt (i.e., one-shot) significantly boosts performance, with CodeLlama managing to create a syntactically correct behavior tree for each task. A similar result is achieved for fine-tuned models. In this case, even with a zero-shot prompt, all models can generate syntactically correct behavior trees for most tasks. Surprisingly, providing an example does not drastically increase accuracy as before, with the extreme case of CodeLlama

TABLE III  
SEMANTIC EVALUATION (PHASE 1)

	Alpaca	LlamaChat	CodeLlama
Base, Zero-Shot			
Fine Tuned, Zero-Shot	14%	14%	71%
Base, One-Shot			
Fine Tuned, One-Shot	28%	57%	100%

TABLE IV  
SYNTACTIC CORRECTNESS (PHASE 2)

	LlamaChat	CodeLlama
Zero-Shot	88,9%	66,7%
One-Shot	88,9%	88,9%*

\* Note: max\_new\_tokens parameter limited to 1000, larger values increase accuracy

achieving a worse result than before. This result shows how impactful the choice of examples can be in the generation of the final result.

Table III summarises the analysis done by human experts on the semantics of the behavior tree. In this context, by semantic, we mean the ability of the system to interpret the task defined in the prompt and create a solution in an XML format that represents the correct flow of execution. When evaluating the semantics of a behavior tree, we do not take into account any syntactic constraint.

As for the previous performance metrics, Alpaca is the least successful of all models. Values are low for all configurations. In the case of LlamaChat and CodeLlama, we observe an interesting behavior. When using a zero-shot prompt, performance increases when the model is fine-tuned. However, when adding an example in the prompt, performance decreases when using a fine-tuned model. This result shows again how impactful prompt design can be in guiding the LLM to achieve the correct solution. Nonetheless, semantic evaluation is performed only for the sake of completeness, since it disregards the syntactic correctness of the tree.

From this initial evaluation phase, we can conclude that in general, fine-tuned models perform better than base models in generating behavior trees in an XML format. They are faster and produce a correct solution more consistently, even with more complex tasks. The few exceptions are related to the use of examples in the prompt, and these situations are explored more in detail in the second evaluation phase. Additionally, we established that Alpaca, while achieving relatively good syntactic correctness when provided with examples, has poor performance across the board. Given these considerations, in the second phase, we will evaluate only LlamaChat and CodeLlama fine-tuned, both without and with examples.

#### D. Validation

During the second evaluation phase, we employed our custom-developed behavior trees validator to assess the overall

TABLE V  
VALIDATION (PHASE 2)

	LlamaChat			CodeLlama		
	ZS	OS	OS+SA	ZS	OS	OS+SA
Task 1		✓	✓		✓	✓
Task 2		✓	✓			
Task 3			✓		✓	✓
Task 4		✓	✓		✓	✓
Task 5		✓	✓			
Task 6			✓			
Task 7						
Task 8			✓			
Task 9						

Success rate for behavior trees generated with zero-shot (ZS), and one-shot (OS) prompts. Additionally, the results of the corrected BTs are included (OS+SA).

correctness of the nine LLM-generated trees. This validator is designed to examine and confirm the overall correctness of the generated behavior trees. Additionally, further tests have been conducted within a simulation environment, in particular, we used the TurtleBot3 by ROBOTIS<sup>2</sup> for navigation tasks leveraging the capabilities of Nav2. For this reason, we developed a simple action client capable of sending navigation goals. Each action specified in the previous tasks was wrapped within a corresponding node within the behavior tree. With this approach, we aimed to ensure the reliability and functionality of the generated behavior trees in practical scenarios, both in simulated environments and real-world scenarios.

In this phase, we limited our analysis to the fine-tuned version of CodeLlama and LlamaChat. The two LLMs are prompted in the same way as the previous phase, however, we redesigned the examples in the one-shot prompts to be more in line with the target task. In practice, this means providing in the example at least one instance of each action being used to hint to the LLM the structure (i.e., name and type) of the parameters.

You will be provided a summary of a task performed by a robot, and your objective is to express this task as a behavior tree in XML format.

The behavior tree represents a mobile robot tasked to visit two locations: (7,1) and (4,8). The available actions are: "MoveTo"

```
<root BTCPP_format="4">
  <BehaviorTree>
    <Sequence>
      <MoveTo x="7" y="1"/>
      <MoveTo x="4" y="8"/>
    </Sequence>
  </BehaviorTree>
</root>
```

The behavior tree represents a mobile robot tasked to visit a sequence of locations: ((0,0), (2,3), (4, 7), (5, 11)). The available actions are: "MoveTo"

For example, this is the prompt used for Task 1. On top, in grey, there is the system prompt used to define the general task of the LLM. Followed, in blue, by the instruction provided as an example. This is a simplified version of the target task where only two locations are visited. Then, the corresponding

behavior tree where the format of the action is exemplified. Last, in green, there is the description of Task 1.

As before, first, we evaluate the syntactic correctness of the behavior trees generated by each model. Table IV shows the results obtained. LlamaChat, both with zero-shot and one-shot prompts, is rather consistent and achieves syntactic correctness for most of the behavior trees. CodeLlama has problems obtaining correct results when no example is provided, possibly because, by being focused on code generation the model was exposed to more diverse types of behavior trees. Nonetheless, with a one-shot prompt, CodeLlama only fails to generate a syntactically correct behavior tree only in one case. It is worth noting that the failure is caused by reaching the maximum number of available tokens during generation. This problem appears when the LLM decomposes the behavior tree in multiple subtrees causing an unnecessarily lengthy solution. In general, multiple iterations of the same prompt can be used to achieve a syntactically correct result.

Table V shows the results obtained after testing the behavior trees using our validation system. We performed the validation considering three different outputs. One is the direct output obtained by using zero-shot prompts, then the output of one-shot prompts as previously described, and lastly, the output of one-shot prompts corrected using a basic static analysis. The result of the static analysis on the behavior tree created using a zero-shot prompt is not reported since it provides no benefit.

```
<root BTCPP_format="4">
  <BehaviorTree>
    <Sequence>
      - <Fallback>
        <Sequence>
          - <Action ID="moveToNewConfiguration"
            goal_pose="{goal_pose}" />
          + <Action ID="moveToNewConfiguration" />
          <Sequence>
            <Action ID="CheckForTarget"
              target="{target}" />
          + <Action ID="CheckForTarget" />
          - <Fallback>
            <Sequence>
              - <Action ID="ApproachTarget"
                target="{target}" />
              + <Action ID="ApproachTarget" />
            </Sequence>
            <Action ID="MoveBack" />
          - </Fallback>
        </Sequence>
      - <Action ID="MoveBack" />
    </Fallback>
  </Sequence>
</BehaviorTree>
</root>
```

This is an example of how behavior trees are corrected via static analysis. The corrected behavior trees are obtained by removing the extra parameters from the actions, and by removing unrecognized actions. In some cases, these changes lead to empty control sequences that are then removed.

The validation process shows that behavior trees generated with a zero-shot prompt consistently fail to achieve the target task. Most of the time this is because there is a mismatch between the provided parameters and the expected format.

<sup>2</sup><https://emmanuel.robotis.com/docs/en/platform/turtlebot3/overview/>

When provided with examples, both models obtain better results. Simpler tasks that contain basic execution flows are correctly generated. In more complex tasks that include a form of control flow (e.g., “During navigation a location may become unreachable, if this happens, skip it”), the behavior tree fails because the LLM fails to understand the request or because actions are added to try to capture this control flow. In this second case, the behavior tree is successful after the action is removed via static analysis. In sum, the best results are achieved by LlamaChat when prompted with an example and after a cleanup of the behavior tree using static analysis. When inspecting the results, CodeLlama is more effective at creating complex and articulated behavior trees, often including subtrees, and comments. However, it fails to understand the request provided in the more complex prompts.

## V. CONCLUSION

We introduce a novel approach in robotics, enabling large language models with at most 7 billion parameters to generate executable behavior trees tailored for robot behaviors. We conducted a comparative analysis using Alpaca, LlamaChat, and CodeLlama, to identify the most suitable model for the task of generating ready-to-use behavior trees. To enhance the generation capabilities of these base models, we created a new instruction-following dataset specifically designed for fine-tuning behavior tree generation. The generated behavior trees have been evaluated in terms of syntactic accuracy, assessed with Groot2, and semantic accuracy, using our custom-developed validator. Furthermore, their performance was assessed within a simulation environment and in real-world deployment on a physical robot. Our assessment shows how fine-tuned models perform better than base models. In particular, LlamaChat is the best model overall, although CodeLlama still demonstrated notable performance. This work showcases the efficiency of compact large language models in directing autonomous agents like robots. Furthermore, behavior trees offer a flexible solution thanks to their inherent modularity, enabling seamless expansion through the incorporation of additional prompts and hence additional robot functionalities. Finally, our approach offers a plug-and-play solution, to transition from the LLM to direct execution on the robot.

The main limitation of using an LLM to generate robot behavior is the challenging tasks of evaluating and validating the correctness of the solution. In our work we managed to do so using a custom-developed validator, however, it is limited to behaviors where the solution is already known. In the future, we plan to deploy our system directly on the robot, therefore, we will focus on how to provide an automatic validation of the generated behavior tree. An option can be the use of milestones in the task to provide intermediate constraints and validate simpler behaviors. Alternatively, an auxiliary LLM can be used to regenerate the description from the XML and compare it with the original prompt.

The final aim of this work is to provide a direct interface between a user and the robot. Using this interface, the user can describe a task using natural language and the robot

will execute it without the need for direct support. Solving this challenge has significant potential to improve human-robot interaction and the capabilities of autonomous robots in a variety of fields such as service robotics, autonomous industrial inspections, and last-mile logistics.

## REFERENCES

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrkke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.
- [3] Swaib Dragule, Sergio García Gonzalo, Thorsten Berger, and Patrizio Pelliccione. Languages for specifying missions of robotic applications. *Software Engineering for Robotics*, pages 377–411, 2021.
- [4] Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Andrzej Wasowski, and Swaib Dragule. Behavior trees and state machines in robotics applications. *IEEE Transactions on Software Engineering*, 49(9):4243–4267, 2023.
- [5] Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv preprint arXiv:2104.13921*, 2021.
- [6] Huihui Guo, Fan Wu, Yunchuan Qin, Ruihui Li, Keqin Li, and Kenli Li. Recent trends in task and motion planning for robotics: A survey. *ACM Computing Surveys*, 2023.
- [7] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [8] Matteo Iovino, Edvards Scukins, Jonathan Styruud, Petter Ögren, and Christian Smith. A survey of behavior trees in robotics and ai. *Robotics and Autonomous Systems*, 154:104096, 2022.
- [9] Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1780–1790, 2021.
- [10] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.
- [11] Artem Lykov and Dzmityr Tssetserukou. Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model. *arXiv preprint arXiv:2305.19352*, 2023.
- [12] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725. IEEE, 2020.
- [13] Petter Ögren and Christopher I Sprague. Behavior trees in robot control systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:81–107, 2022.
- [14] George Pu, Anirudh Jain, Jihan Yin, and Russell Kaplan. Empirical analysis of the strengths and weaknesses of peft techniques for llms. *arXiv preprint arXiv:2304.14999*, 2023.
- [15] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- [16] Ishika Singh, Valtis Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.
- [17] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. <https://crfm.stanford.edu/2023/03/13/alpaca.html>*, 3(6):7, 2023.

- [18] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [19] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [21] Eric Wang. Alpaca-lora. <https://github.com/tloen/alpaca-lora>, 2023.
- [22] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.
- [23] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.