BTGenBot-2: Efficient Behavior Tree Generation with Small Language Models

Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci
Department of Electronics, Information, and Bioengineering
Politecnico di Milano, Milano
(riccardo.izzo, gianluca.bardaro, matteo.matteucci)@polimi.it

***Abstract*—Recent advances in robot learning increasingly rely on LLM-based task planning, leveraging their ability to bridge natural language with executable actions. While prior works showcased great performances, the widespread adoption of these models in robotics has been challenging as 1) existing methods are often closed-source or computationally intensive, neglecting the actual deployment on real-world physical systems, and 2) there is no universally accepted, plug-and-play representation for robotic task generation. Addressing these challenges, we propose BTGenBot-2, a 1B-parameter open-source small language model that directly converts natural language task descriptions and a list of robot action primitives into executable behavior trees in XML. Unlike prior approaches, BTGenBot-2 enables zero-shot BT generation, error recovery at inference and runtime, while remaining lightweight enough for resource-constrained robots. We further introduce the first standardized benchmark for LLM-based BT generation, covering 52 navigation and manipulation tasks in NVIDIA Isaac Sim. Extensive evaluations demonstrate that BTGenBot-2 consistently outperforms GPT-5, Claude Opus 4.1, and larger open-source models across both functional and non-functional metrics, achieving average success rates of 90.38% in zero-shot and 98.07% in one-shot, while delivering up to 16× faster inference compared to the previous BTGenBot.**

***Index Terms*—Robot Learning, Large Language Models, Behavior Trees**

I. Introduction

The process of autonomously generating task sequences directly executable by robots has sparked interest in the robotic community following the emergence of foundation language models such as GPT-4 [1], Gemini [2], and Llama [3]. These models have demonstrated impressive reasoning capabilities [4], being able to represent commonsense knowledge [5], grounding language to physical objects [6], and adapting plans based on contextual information [7]. Consequently, various representations have been exploited to generate robotics tasks using Large Language Models (LLMs), such as Planning Domain Definition Language (PDDL) [8], Linear Temporal Logic (LTL) [9], Python code [10], [11] and Behavior Trees (BTs) [12], [13]. In particular, BTs have gained popularity due to their scalable and structured approach to managing complex robotic tasks, with the BehaviorTree.CPP library becoming a standard component in the Robot Operating System 2 (ROS2) ecosystem, primarily because of its inclusion in the Navigation2 stack [14]. Despite these advances, most prior works focus on the integration of closed-source models, such as OpenAI's GPT [1], into their pipeline, neglecting the actual deployment on resource-constrained robots and relying on APIs that are not accessible in offline scenarios. Although some studies achieve impressive results, two major limitations prevent their practical adoption: 1) current models are *closed-source*, or with limited transparency into the model architecture, training procedures, and data mixture, and 2) the dependency on computationally intensive models that cannot be feasibly executed on actual robotic hardware. We argue that advancing robotics research requires the development of open-source and efficient models building on open-source LLMs [3], [15]. This enables the released model to be deployed locally on real robots without relying on external APIs or additional components. Additionally, the field still lacks standardized benchmarks for evaluating LLM-driven BT generation in robotics. To this end, we introduce BTGenBot-2, a 1B parameter open-source Small Language Model (SLM) that establishes a new state-of-the-art in generating executable and ROS2-compatible BTs. As illustrated in Figure 1, BTGenBot-2 is built upon a pretrained SLM, further fine-tuned with compute-efficient methods [16], [17] on a custom synthetic instruction-following dataset. Unlike the previous approach, BTGenBot-2 enables zero-shot BT generation, is resilient to errors at both inference and runtime, and is compact enough to run on consumer-grade GPUs while preserving state-of-the-art performance. We further introduce a comprehensive benchmark of 52 tasks spanning navigation and manipulation, organized into three levels of difficulty, and implemented in NVIDIA Isaac Sim[1]. Our experimental results show that BTGenBot-2 outperforms BTGenBot [18], a 7B-parameter state-of-the-art model for BT generation, and proprietary models such as OpenAI's GPT-5 and Anthropic's Claude Opus 4.1 across functional and non-functional metrics. Our main contributions are the following:

1) We release a new synthetic instruction-following dataset of 5,204 pairs of executable BTs and natural language task descriptions, filling a gap in planning datasets by providing a foundation for generalisable BT generation.
2) We introduce BTGenBot-2, a fully open-source SLM for on-device BT generation within the ROS2 stack
3) We design two novel failure detection mechanisms to handle errors at both inference and runtime, significantly enhancing robustness.
4) We propose the first standardized benchmark for evaluating LLM-based BT generation, enabling reproducible

[1]https://developer.nvidia.com/isaac/sim

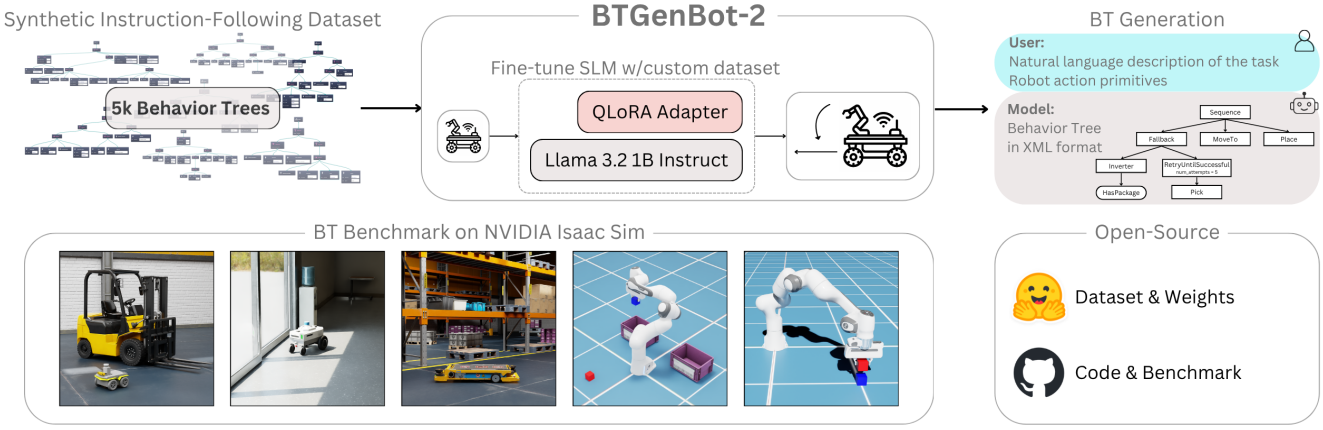

Fig. 1. We present **BTGenBot-2**, a 1B-parameter open-source small language model (SLM), trained on 5k natural language instructions and behavior tree pairs from a synthetic instruction-following dataset. BTGenBot-2 achieves state-of-the-art performance in generating directly executable behavior trees. We open-source our synthetic dataset, model weights, codebase, and the BT Benchmark to support dissemination and reproducibility.

and comparable assessment across functional and non-functional metrics. BTGenBot-2 is validated extensively with this benchmark in simulation and on real robots, establishing itself as a strong BT generator.

We publicly release our model, dataset, benchmark, and deployment code[2].

II. Related Work

A. Behavior Trees (BTs)

A BT is formally defined as a directed rooted tree where the internal nodes are called *control flow nodes* and the leaf nodes are called *execution nodes* [12]. The root node has no parent, while all other nodes form a parent-child relationship. During execution, a periodic tick signal originates at the root node and propagates through the tree from left to right. Nodes execute only when ticked and return one of the possible status signals: *Success*, *Failure*, or *Running*. The classical representation of a BT comprises four categories of control flow nodes (Sequence, Fallback, Parallel, and Decorator) and two categories of execution nodes (Action and Condition). Prior studies comparing BTs to other representations such as Finite State Machines (FSMs), PDDL, and Decision Trees highlight the advantages of BTs in terms of modularity, reactivity, and interpretability [19]. These benefits, in addition to the availability of libraries such as BehaviorTree.CPP[3] and pytrees[4], have further driven the utilization.

B. LLMs in robotics

Large Language Models (LLMs) represent a class of deep learning models typically built upon the Transformer architecture. Although they were originally tailored for text generation [20], LLMs have since demonstrated remarkable capabilities in instruction-following [21], reasoning [4] and in-context learning [20]. Nowadays, among the most widely recognized LLMs, there are GPT [1], Gemini [2] and Llama [3]. Their rapid adoption has also extended to the field of robotics, where LLMs have been leveraged for language-based task generation [10], [18], [22], end-to-end action generation with visual representation [23], [24], and human-robot interaction [25].

C. BTs generation using LLMs

Recent efforts in robotics increasingly leverage LLMs or foundation models to achieve high-level task planning. Early methods translate user commands into Python scripts, mapping language-based instructions to code primitives [10], [11], [22], or into classical planning languages such as PDDL [26] and LTL [9]. One emerging approach revolves around the direct generation of BTs from a natural-language description of the task [18], [27]–[31]. LLM-BrAIn [27] uses completely synthetic data and relies only on human evaluation, neglecting practical robot deployment. BTGenBot [18] leverages real-world tested BTs to fine-tune a lightweight model, offers a comprehensive evaluation framework, and outputs XML-based BTs compatible with the BehaviorTree.CPP library. LLM-as-BT-Planner [28] presents four in-context learning strategies, including a human-in-the-loop approach, to build BTs from user directives. OBTEA [29] encodes goals as well-formed first-order logic formulas and processes these via an LLM before expanding the plan into a BT. Similarly, LLM-BT [30] incorporates semantic maps showcasing robustness and adaptability to environmental changes, while BETR-XP-LLM [31] employs dynamic expansion to resolve errors in both planning and execution phases. Prior works [28]–[31] leverage closed-source GPT models or rely on external APIs, making them unsuitable for deployment on a real robot. Our method utilizes an open-source SLM directly deployable on a robot and can operate on consumer hardware. Furthermore, while [28], [29] heavily rely on prompting techniques such as in-context learning and few-shot demonstrations, our model performs zero-shot without requiring the user to provide any additional examples. In addition, [29], [31] focus on the

[2]https://airlab-polimi.github.io/BTGenBot-2/
[3]https://www.behaviortree.dev/
[4]https://py-trees.readthedocs.io/en/devel/

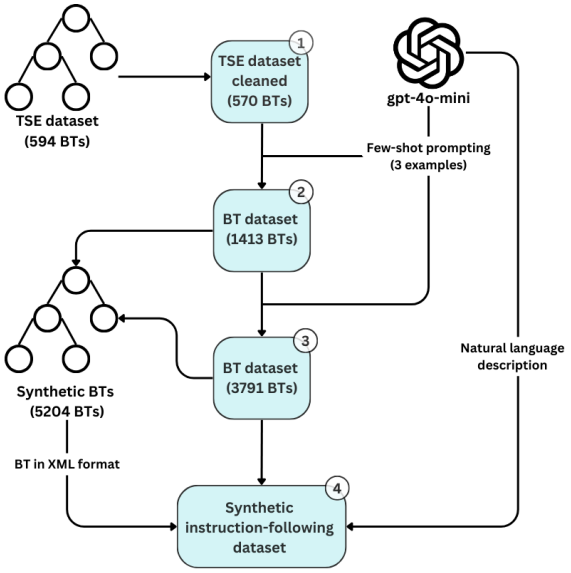

Fig. 2. **Dataset generation.** Starting with the TSE dataset, a new instruction-following dataset is created through four key steps: (1) cleanse the raw XML data, (2) for each original BT, use *gpt-4o-mini* to generate three variants, (3) repeat step 2 with the new dataset, (4) merge all resulting datasets while producing a natural-language description for each BT.

Instruction:
You are a helpful assistant that can assist with creating behavior trees. Your task:
- Convert the provided summary of a behavior into an XML-formatted behavior tree.
- Ensure the behavior tree matches the description in the summary.
- The behavior tree must be compatible with the BehaviorTree.CPP library.
- Only use the actions and parameters provided in the action list below the summary.

Output Requirements:
- Output only the XML representation of the behavior tree. Do not include explanations, comments, or any additional text.
- Ensure all actions and parameters strictly match the provided list.
- If possible limit the use of SubTrees.

Please generate the behavior tree based on the summary and action list provided.

Input:
The behavior tree facilitates a robot's navigation by implementing a fallback mechanism to achieve its tasks. It first attempts to compute and follow a path using the A* algorithm to reach a designated goal (goal1). If this sequence is unsuccessful, it tries to compute a path to a second goal (goal2) using the Dijkstra algorithm before following it. If both attempts to reach the goals fail, the robot will move to a predefined location called "Base." This structure ensures that the robot has multiple options to achieve successful navigation, providing robustness and flexibility in its operational environment.

Actions: [ComputePathToPose (parameters: goal, path, planner_id), FollowPath (parameters: path, controller_id), MoveTo (parameters: name, location)]

Output:

```
<root BTCPP_format="4">
 <BehaviorTree ID="MainTree">
  <Fallback>
   <Sequence>
    <ComputePathToPose goal="{goal1}" path="{path1}" planner_id="AStar"/>
    <FollowPath path="{path1}" controller_id="FollowPath1"/>
   </Sequence>
   <Sequence>
    <ComputePathToPose goal="{goal2}" path="{path2}" planner_id="Dijkstra"/>
    <FollowPath path="{path2}" controller_id="FollowPath2"/>
   </Sequence>
   <MoveTo name="return_to_base" location="Base"/>
  </Fallback>
 </BehaviorTree>
</root>
```

Fig. 3. **Dataset sample.** A representative example from the generated instruction-following dataset with its three components: the *instruction* that provides system contextual information, the *input* comprising a natural-language task description and its corresponding robot actions, and the *output* showcasing the generated XML-based behavior tree.

expansion of pre-existing BTs, while our approach directly generates executable BTs. Finally, aside from BTGenBot [18] and our framework, existing approaches to BT generation are closed-source and not ROS2–compatible. Consequently, we will consider BTGenBot as our main baseline.

III. Methodology

This section details our methodology for generating executable BTs using an SLM, as outlined in Figure 4. Building on BTGenBot, we surveyed the top-performing open-source SLMs available on *Open LLM Leaderboard* by Hugging Face and selected *Llama-3.2-1B-Instruct* [32], an instruction-tuned model with an 8k context length. Section III-A formulates the problem. Section III-B details our data curation pipeline, while Section III-C describes the QLoRA fine-tuning process. Finally, Section III-D outlines inference and runtime error recovery mechanisms, while Section III-E introduces our standardized framework for evaluating LLM-based BT generation.

A. Problem Formulation

We formulate the generation of BTs as a supervised learning task. Let $\mathcal{D} = \{(\mathcal{I}_i, \mathcal{A}_i, \mathcal{T}_i)\}_{i=1}^{N}$ be a dataset where $\mathcal{I}$ represents a natural language instruction, $\mathcal{A}$ is the set of robot action primitives, and $\mathcal{T}$ is the corresponding behavior tree in XML format. Our objective is to fine-tune a SLM parameterized by θ to maximize the probability of generating the correct behavior tree $\mathcal{T}$ given the inputs:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{(\mathcal{I},\mathcal{A},\mathcal{T})\in\mathcal{D}} \log P_\theta(\mathcal{T} \mid \mathcal{I}, \mathcal{A}) \qquad (1)$$

The generated output is implicitly constrained to be a valid XML schema compatible with BehaviorTree.CPP, containing only leaf nodes present in the action space $\mathcal{A}$.

B. Dataset

We created a new instruction-following dataset following the Alpaca schema [33], which includes three components for each sample: *instruction*, *input*, and *output*. The *instruction* component provides system contextual information for the model, including its role, the required output format, and any additional constraints. The *input* component is a natural-language description of a robotic task paired with the list of available robot actions. Finally, the *output* component is an XML-based BT that fits the provided description of the task. The foundation of our dataset is the TSE dataset [13], which includes approximately 600 behavior trees sourced from open-source robotics projects. These BTs have been validated on physical robots, ensuring high-quality and reliable real-world implementations. As illustrated in Figure 2, the generation of our dataset is articulated in four steps:

(1) Data Preparation. We start with 594 BTs from the TSE dataset [13]. We first verified syntactic correctness and compatibility with *BehaviorTree.CPP* using a Python XML parser. After these validation steps, we are left with 570 BTs.

(2) First BT Generation. From this curated set, we used *gpt-4o-mini* to produce three new variants of each BT, following the approach of [33], which generates instruction-response pairs from a curated human-written instruction-response pairs seed set. We chose *gpt-4o-mini* due to its fast and low-cost inference. Then, inspired by [34], we employed nucleus sampling (top-p = 0.99) to encourage creativity while using the same few-shot examples, but we constrained the generation so that the execution nodes remain unchanged. This ensured compatibility with the underlying robotic APIs and *Behav-*

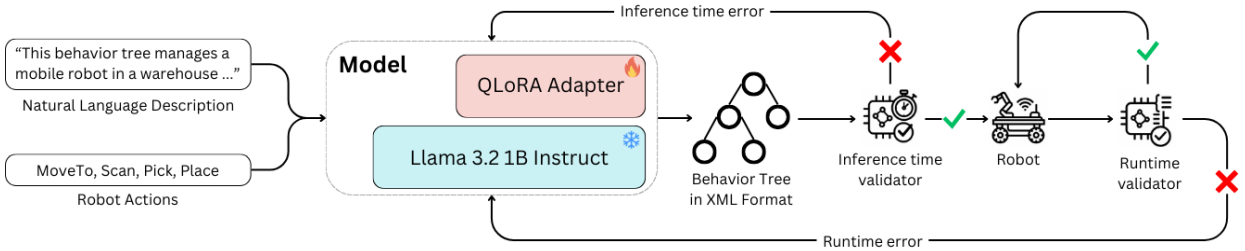

Fig. 4. **Overview of the model architecture.** The model takes as input a natural language task description of a robotic task along with the set of available robot action primitives, generating a ROS2-compatible BT in XML format. The model is adapted using a QLoRA adapter while keeping pre-trained parameters frozen. The generated BTs are initially validated at inference time, checking for syntax and action-space consistency before execution. Additionally, at runtime, an inline logger tracks stack traces and blackboard states, triggering subtree regeneration in case of errors.

iorTree.CPP. Through a final cleaning phase, this process produced 1413 synthetic BTs.

(3) Second BT Generation. We repeated the process from step (2), with the newly generated 1413 BTs as the input set, obtaining an additional 3791 synthetic BTs. This step further increased the overall diversity of the dataset.

(4) Instruction Generation. We merged the two synthetic BT datasets from steps (2-3), resulting in a total of 5204 BTs. For each of these BTs, *gpt-4o-mini* then generates: (i) a brief natural language description of the task (less than 200 words), and (ii) a list of its corresponding action nodes and parameters. This step completes the dataset by pairing each BT with a task description and the required robot actions.

To evaluate the quality of outputs generated by *gpt-4o-mini*, we conducted a visual assessment on a subset of results. This involved checking for semantic coherence between the natural language descriptions and their corresponding BTs, as well as verifying the syntactic correctness of the BTs using an XML parser. Figure 3 shows a representative example from the resulting dataset. Although generating additional samples might have further diversified the data, our preliminary experiments suggested decreasing returns and BT quality decline. The resulting dataset reflects real-world scale by preserving the original training distribution in terms of task types. The BTs average 11.23 nodes and 10.23 transitions, yet extend to up to 157 nodes and 156 transitions, covering both common BTs and larger ones used in complex industrial settings. Our dataset provides a significant advantage over existing BT collections. For instance, [18] offers a high-quality real-world dataset, but the number of training tokens is significantly below what would be needed for fine-tuning a 7-billion-parameter model. Conversely, the dataset introduced by [27] is purely synthetic. While the size of approximately 8500 samples is closer to the recommended token counts, it lacks the same level of real-world validation as the BTs sourced from the TSE dataset. Our dataset combines the strengths of both approaches: a core of curated real-world BTs, enhanced with controlled synthetic generation to expand size and variety.

C. Fine-Tuning

LLMs fine-tuning is a computationally intensive task, therefore, we employed a Parameter-Efficient Fine-Tuning (PEFT) approach, specifically QLoRA [17], a quantized extension of LoRA [16]. QLoRA retains LoRA's benefits, such as avoiding full model retraining, typically training on the order of one-thousandth as many parameters as the original model. For instance, in a 1-billion-parameter model such as *Llama-3.2-1B-Instruct*, LoRA may only require training approximately one million parameters. 15 tokens per parameter heuristic suggests about 15 million tokens for a 1-billion-parameter model [35], whereas our dataset amounts to about 3.5 million tokens, an order of magnitude below the target. Despite falling short of the previous heuristics, we decided to proceed based on emerging evidence that even relatively small but high-quality datasets can effectively fine-tune LLMs to achieve competitive performance [36]. For training, we used the dataset described in Section III-B, which was partitioned into training and test sets with a conventional 95%-5% split. Most of the QLoRA hyperparameters were left at default (e.g., *lora_r*, *lora_alpha*, and *lora_dropout*). Following [17], we expanded the original set of LoRA target modules beyond the attention layers ["k_proj", "q_proj", "v_proj", "o_proj"] to include the MLP layers ["gate_proj", "down_proj", "up_proj"]. Typical LLMs train for at most one or two epochs, but we found that using up to five epochs was beneficial for our comparatively smaller dataset, yielding training accuracies exceeding 95%. Regarding the learning rate, we kept the standard fixed value of 1e-4. We used a learning rate warmup with a ratio of 0.1 to improve training stability and reduce early overfitting. The final model was trained for approximately 30 hours on two NVIDIA RTX Quadro 6000 with a total of 48GB of VRAM, using a batch size of 16.

D. Error Handling and Recovery

A key motivation for adopting BTs as our representation is their inherent ability to handle failures and enable recovery strategies. In BTGenBot-2, we explicitly address two sources of errors that must be considered when automatically generating BTs: inference time errors that occur when generating BTs from natural language, and runtime errors that arise during execution on a robot.

1) Inference Time Validator: LLMs often produce outputs that may be syntactically malformed or semantically inconsistent with the robot's action space. To ensure deployability,

BTGenBot-2 incorporates a validation layer that constrains decoding to XML format and enforces a closed vocabulary of BT nodes and action primitives. Any malformed output is automatically rejected, leading to a new generation. This validation layer takes advantage of a Python parser, designed for BTs, using a YAML file that describes all the allowed primitives, thus avoiding the risk of hallucination when generating the BT. This layered validation ensures that, before execution, every BT produced by BTGenBot-2 is both syntactically correct and aligned with the robot primitives of the target robotic system, eliminating the need for human intervention.

2) Runtime Validator: Unlike flat action sequences, BTs support reactive execution through control flow nodes such as Fallback and Retry. Additionally, BTGenBot-2 focuses on local recovery mechanisms that can be encoded directly in the generated BT. This is achieved with a C++ in-process logger and implemented by every tree node. When any leaf or condition node returns "Failure", the node records the stack trace and a snapshot of the blackboard. If a local recovery (e.g., Fallback, Retry) succeeds, no regeneration is prompted. Otherwise, if the failure is expected to propagate, the runtime parser requests a subtree regeneration from the LLM. Upon validation with the inference time validator, the original tree is modified with the new sub-tree. This mechanism further enhances reactivity and robustness, enabling the system to correct even unexpected errors during execution.

E. Behavior Tree Benchmark

Evaluating the generation of BTs in robotics is inherently challenging as it requires evaluating both structural validity and execution. However, existing approaches often rely on ambiguous representations, unsupported by widely adopted frameworks such as ROS2 BehaviorTree.CPP and py_trees, and on narrowly defined task suites, limiting the possibility of reproducible comparison. Additionally, effective testing of articulated BTs requires a complex but deliberately diverse robotic setup to ensure comprehensive coverage of BT variants and task domains. With LLMs now able to synthesize BTs from natural language, the absence of a common and reproducible benchmark has become a critical bottleneck. We propose a standardized benchmark for evaluating BT generation, adopting *BehaviorTree.CPP* as the canonical syntax, given its compatibility with ROS2. The benchmark comprises 52 tasks across navigation and manipulation and is organized into three difficulty levels. Navigation includes 32 tasks, with 12 easy, 10 medium, and 10 hard. Manipulation includes 20 tasks, with 6 easy, 8 medium, and 6 hard. Overall, the benchmark contains 18 easy, 18 medium, and 16 hard tasks. Navigation tasks include point-to-point and waypoint-sequence problems at the easy level, priority and conditional navigation at the medium level, and mixtures of these at the hard level, with particular emphasis on the use of novel action primitives, fault recovery and more complex reasoning. Manipulation tasks focus on tabletop scenarios, including pick-and-place, stacking, sorting, and object reordering. To stress generalization, the hard tasks are intentionally out-of-distribution relative to the original BTs in our dataset, providing a substantially harder test of a model's ability to serve as a broadly generalizable BT generator. All tasks are implemented in *NVIDIA Isaac Sim*, a widely used simulation environment for robot learning.

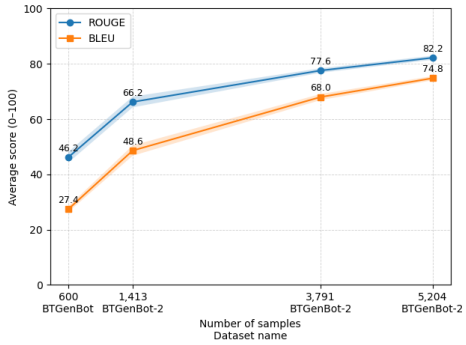

Fig. 5. **Data scaling evaluation.** Average ROUGE and BLEU scores (mean ± std) with increasing dataset size: ROUGE: 46.2 ± 1.94, 66.2 ± 1.94, 77.6 ± 0.80, 82.2 ± 0.75; BLEU: 27.4 ± 1.02, 48.6 ± 1.85, 68.0 ± 1.10, 74.8 ± 0.75, corresponding to 600, 1,413, 3,791, and 5,204 samples. Standard deviation evaluated across 5 runs with temperature=0.9

IV. Experiments and Results

The goal of our experimental evaluation is to assess the performance and effectiveness of our model as a powerful tool to generate robotic tasks through the use of BTs and natural language instructions. Specifically, we benchmark our model against prior works that employ LLMs for BT generation, and against current state-of-the-art generalist LLMs. In our evaluation, BTGenBot [18] serves as the primary baseline for LLM-based BT generation, while GPT-5 and Claude Opus 4.1 represent the leading general-purpose LLMs.

A. Data Curation Pipeline Effectiveness

In this section, we assess how dataset scale affects BT generation quality, thereby validating the curation pipeline introduced in Section III-B. We fine-tune *Llama-3.2-1B-Instruct* on four dataset configurations with the procedure described in Section III-C, varying the number of training epochs according to dataset size while keeping other hyperparameters fixed. To assess the model's convergence toward the target data distribution, we employed text-based metrics such as ROUGE [37] and BLEU [38]. The ROUGE score measures the similarity between the generated text and the reference text using overlapping n-grams and word sequences that appear in both texts. The BLEU score measures the overall quality of text with respect to the original. All models were evaluated on a test split comprising 5% of each dataset configuration. As shown in Figure 5, both metrics increase monotonically with additional training samples. The largest gain occurs when scaling from the prior BTGenBot corpus to a curated subset of 1,413 BTs. Further scaling up yields smaller but consistent improvements, leading to convergence. These results provide evidence that our data curation pipeline, with a curated corpus of BTs, practically improves BT generation.

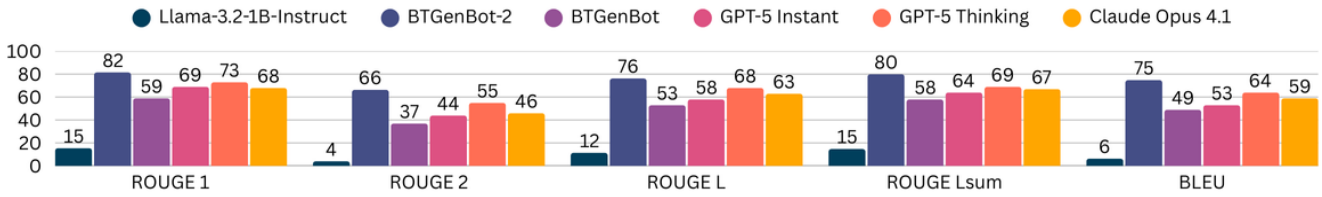

Fig. 6. **ROUGE and BLEU Scores.** Evaluation on a test set of 250 BTs from the custom instruction-following dataset demonstrates that fine-tuning substantially improves performance compared to the pre-trained baselines, with BTGenBot-2 achieving the highest scores. GPT-5 Thinking performs competitively, with GPT-5 Instant and Claude Opus 4.1 falling slightly behind. The original BTGenBot is outscored by its successor, BTGenBot-2.

B. Preliminary Evaluation Phase

In this preliminary evaluation phase, we benchmarked six models for BT generation quality on a test set of 250 BTs spanning navigation and manipulation tasks. The *Llama-3.2-1B-Instruct* model was used as a lower-bound reference, while its fine-tuned variant, *BTGenBot-2*, represents our proposed approach. As baselines, we considered *BTGenBot*, as well as leading proprietary LLMs such as OpenAI *GPT-5* (Instant and Thinking modes) and Anthropic *Claude Opus 4.1*. Model quality was initially assessed using ROUGE 1/2/L/Lsum and BLEU. While these scores cannot strictly evaluate the semantic validity of a BT, they serve as an essential preliminary validation for assessing the model's ability to acquire the correct XML syntax and action usage. ROUGE 1/2 captures unigram and bigram overlap (lexical recall and word-order sensitivity), while ROUGE L/Lsum evaluates sequence-level structural similarity through the longest common subsequence. As shown in Figure 6, *BTGenBot-2* achieves the best scores across all metrics, outperforming even the strongest proprietary baseline (*GPT-5 Thinking*). In contrast, the base pre-trained version of the model exhibited the worst performance, highlighting the importance of domain-specific fine-tuning. Intermediate performance was observed for *BTGenBot* and *GPT-5 Instant*, with the latter slightly outperforming the original *BTGenBot*.

C. Validation Phase

In this validation phase, we evaluate the same models as in Section IV-B, excluding the base version *Llama-3.2-1B-Instruct* due to its consistently low performance in Section IV-B. Our model, BTGenBot-2, is designed to work zero-shot, promoting a plug-and-play system that is intuitive and accessible to end users. This eliminates the need for prompt engineering or the inclusion of examples, while preserving high-quality BT generation. However, for a fair comparison, we also evaluated the model in a one-shot setting, allowing direct comparison with proprietary models that were not fine-tuned on our dataset. For BTGenBot-2, we consider an additional version, namely BTGenBot-2-ER (Error Recovery), which employs the two validators outlined in Section III-D. To evaluate the generalizability of our ER module, we conducted an ablation study by applying it also to Claude Opus 4.1. The goal of this phase is to assess the end-to-end pipeline as a generalist BT generator. Evaluation is performed on the benchmark proposed in Section III-E, with results reported separately for each level. For the evaluation, we use a mixture of functional and non-functional metrics [39]. The functional metrics assess whether a BT executes its intended functions:

- **Success Rate (SR)**: a BT is successful if it is executable and achieves the goal state.
- **Pass@k (P@k)**: measures the likelihood that at least one of the top-k generated candidates is correct [40], where correctness is determined by the definition of SR.
- **Inference Time**: time required by the 4-bit quantized model to generate a complete BT, measured on an NVIDIA GTX 1080 GPU (8 GB VRAM).

Complementary, non-functional metrics measure aspects of the BT that are not directly related to its functionality and apply regardless of its performance:

- **Action Coherency**: verifies that the BT uses only the predefined robotic actions from the task prompt, computed with the Python parser in Section III-D1.
- **XML Syntax**: assess the correctness of the XML schema and the compatibility with the BehaviorTree.CPP library, using the system of Section III-D1.
- **Semantic Correctness**: evaluated by three human experts using a binary scale to measure whether the structure and logic of the BT align with the natural language task description. Final decisions were made by majority vote.

Table I compares the models on easy/medium/hard tasks in both zero-shot and one-shot settings, using both functional and non-functional metrics. All models were prompted uniformly, receiving only the task description and the list of available robot primitives. In the zero-shot setting, BTGenBot-2 achieves an average success rate of 84.61%, outscoring the strongest baseline GPT-5 Thinking by almost 14%, and surpassing by a larger margin GPT-5 Instant and Claude Opus 4.1, as well as the previous BTGenBot by more than 30%. As expected, Pass@3 improves absolute accuracy without altering the relative ranking of models. In terms of efficiency, BTGenBot-2 generates a BT in 11 seconds on average, comparable only to GPT-5 Instant, and up to 3× faster than other proprietary reasoning models and 16× faster than the original BTGenBot. Regarding non-functional metrics, BTGenBot-2 demonstrates strong reliability: it is the only model that consistently uses only the provided robot actions, while the original BTGenBot consistently fails due to the original fine-tuning dataset not including the list of actions in the input prompt. XML syntax correctness is nearly perfect in BTGenBot-2, outperforming other models that occasionally produce invalid

TABLE I
EVALUATION OF MODELS ON FUNCTIONAL AND NON-FUNCTIONAL METRICS

	Functional										Non-Functional		
Model	**SR (E)**	**P@3 (E)**	**Time (E)**	**SR (M)**	**P@3 (M)**	**Time (M)**	**SR (H)**	**P@3 (H)**	**Time (H)**	**Avg SR**	**Action Coherency**	**XML Syntax**	**Semantic Correctness**
Zero-shot													
BTGenBot-2-ER	**18 / 18**	**18 / 18**	5.8s	**17 / 18**	**17 / 18**	8.2s	**12 / 16**	**13 / 16**	11.7s	**90.38%**	**100%**	**100%**	**94.23%**
BTGenBot-2	17 / 18	**18 / 18**	**5.1s**	15 / 18	16 / 18	**7.5s**	**12 / 16**	12 / 16	**11.0s**	84.61%	**100%**	96.15%	**94.23%**
BTGenBot	13 / 18	15 / 18	85s	10 / 18	12 / 18	98s	7 / 16	8 / 16	178s	57.69%	63.46%	78.84%	46.15%
GPT-5 Instant	15 / 18	16 / 18	12.8s	14 / 18	15 / 18	15.0s	5 / 16	7 / 16	15.2s	65.38%	88.46%	82.69%	80.76%
GPT-5 Thinking	17 / 18	**18 / 18**	34.2s	15 / 18	15 / 18	36.8s	5 / 16	6 / 16	39.7s	71.15%	92.30%	86.53%	86.53%
Claude Opus 4.1-ER	**18 / 18**	**18 / 18**	23.9s	15 / 18	15 / 18	28.1s	7 / 16	9 / 16	30.4s	76.92%	92.30%	90.38%	88.46%
Claude Opus 4.1	17 / 18	**18 / 18**	23.1s	13 / 18	14 / 18	27.4s	4 / 16	6 / 16	29.8s	65.38%	90.38%	82.69%	80.76%
One-shot													
BTGenBot-2-ER	**18 / 18**	**18 / 18**	**8.4s**	**18 / 18**	**18 / 18**	9.6s	**15 / 16**	**15 / 16**	13.1s	**98.07%**	**100%**	**100%**	**98.07%**
BTGenBot-2	**18 / 18**	**18 / 18**	**8.4s**	16 / 18	17 / 18	**8.9s**	14 / 16	**15 / 16**	**12.6s**	92.38%	**100%**	98.07%	94.23%
BTGenBot	16 / 18	17 / 18	101s	14 / 18	14 / 18	114s	7 / 16	8 / 16	197s	71.15%	73.07%	84.61%	51.92%
GPT-5 Instant	**18 / 18**	**18 / 18**	15.3s	16 / 18	16 / 18	16.2s	7 / 16	8 / 16	18.8s	78.84%	96.15%	86.53%	84.61%
GPT-5 Thinking	**18 / 18**	**18 / 18**	35.8s	17 / 18	17 / 18	36.8s	9 / 16	10 / 16	42.2s	84.61%	**100%**	92.30%	88.46%
Claude Opus 4.1-ER	**18 / 18**	**18 / 18**	25.5s	**18 / 18**	**18 / 18**	29.8s	10 / 16	12 / 16	34.3s	88.46%	**100%**	96.15%	94.23%
Claude Opus 4.1	**18 / 18**	**18 / 18**	25.0s	**18 / 18**	**18 / 18**	29.4s	8 / 16	10 / 16	33.6s	84.61%	96.15%	90.38%	88.46%

outputs. BTGenBot-2 demonstrates good capabilities in generating BTs that are syntactically correct and fully compatible with BehaviorTree.CPP. Concerning semantic correctness, our model captures task intent most accurately, followed by GPT-5 Thinking, while GPT-5 Instant, Claude Opus 4.1, and BTGenBot lag behind. With one-shot prompting, each model receives a single example of BT generation. All models benefit from the additional example, with proprietary models averaging around 80% success rate. BTGenBot-2 achieves the best performance at 92.38%, while also remaining the fastest model despite longer average inference times due to the more complex one-shot prompt. For non-functional metrics, only BTGenBot-2 and GPT-5 Thinking achieve perfect action coherency, though GPT-5 Thinking lags with the other metrics. Considering the variant with error recovery, BTGenBot-2-ER, the integration of the two validators further improves performance in both zero-shot and one-shot settings, achieving an average success rate of 90.38% and 98.07%, respectively. Notably, in nearly all cases, a single iteration is sufficient to successfully correct the BT. This improvement comes at the negligible cost of less than one additional second of inference time on average, due to the regeneration of incorrect BTs. XML syntax reaches 100%, ensuring completely valid and executable BTs. Results in both zero-shot and one-shot scenarios demonstrate that the error recovery module is beneficial even when applied to generalist LLMs such as Claude Opus 4.1. Notably, the ER module increases the model's success rate, particularly in hard tasks, while significantly improving all the non-functional metrics, achieving perfect action coherency and high syntax correctness. Overall, the results show that BTGenBot-2-ER delivers the strongest combination of reliability, structural fidelity, and efficiency across tasks and prompting settings. It achieves substantial gains over the previous BTGenBot and consistently outperforms proprietary models, establishing itself as a robust and effective tool for generating complex BTs from natural language instructions.

D. Real Robot Validation

We assessed the practical applicability of our pipeline with real-world trials on physical robots. For navigation, we used an AgileX Scout, a mobile wheeled robot with skid-steering kinematics, while for manipulation we employed an SO-ARM 101, a 6-DoF manipulator. Within this setup, we tested the benchmark's easy tasks and achieved a success rate of $17/18$ with Pass@3, improving to $18/18$ with error recovery. These experiments demonstrate that BTGenBot-2 can be effectively deployed in real-world scenarios. Since the benchmark includes complex tasks that extend beyond the constraints of our laboratory setup, the results in Table I provide a more comprehensive evaluation of the model's capabilities.

V. CONCLUSION

We presented BTGenBot-2, a 1B-parameter open-source SLM that translates natural language task descriptions into executable BTs. Through a carefully curated synthetic dataset, compute-efficient fine-tuning, and two novel validators, BTGenBot-2 delivers strong performance while remaining lightweight enough for deployment on resource-constrained robots. We also released the first standardized benchmark for LLM-based BT generation, spanning 52 navigation and manipulation tasks at varying difficulty levels, providing a reproducible basis for evaluation. Experiments in NVIDIA Isaac Sim and qualitative tests on a real robot show that BTGenBot-2 consistently outperforms prior systems, including its predecessor and large foundation models such as GPT-5 and Claude Opus 4.1. These results highlight the advantages of SLM for robotic task planning, bridging the gap between a natural language instruction of the task and a practical representation that is executable on a real robot.

ACKNOWLEDGEMENT

This paper is supported by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence).

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] J. Huang and K. C.-C. Chang, "Towards reasoning in large language models: A survey," *arXiv preprint arXiv:2212.10403*, 2022.

[5] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 31 967–31 987, 2023.

[6] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 2998–3009.

[7] S. Gupta, K. Yao, L. Niederhauser, and A. Billard, "Action contextualization: Adaptive task planning and action tuning using large language models," *IEEE Robotics and Automation Letters*, 2024.

[8] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 79 081–79 094, 2023.

[9] J. Pan, G. Chou, and D. Berenson, "Data-efficient learning of natural language to linear temporal logic translators for robot task specification," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 554–11 561.

[10] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[11] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.

[12] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[13] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior trees and state machines in robotics applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 9, pp. 4243–4267, 2023.

[14] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2718–2725.

[15] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love *et al.*, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[16] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[17] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023.

[18] R. A. Izzo, G. Bardaro, and M. Matteucci, "Btgenbot: Behavior tree generation for robotic tasks with lightweight llms," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 9684–9690.

[19] O. Biggar, M. Zamani, and I. Shames, "An expressiveness hierarchy of behavior trees and related architectures," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5397–5404, 2021.

[20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[21] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu *et al.*, "Instruction tuning for large language models: A survey," *arXiv preprint arXiv:2308.10792*, 2023.

[22] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on robot learning*. PMLR, 2023, pp. 287–318.

[23] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*. PMLR, 2023, pp. 2165–2183.

[24] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi *et al.*, "Openvla: An open-source vision-language-action model," *arXiv preprint arXiv:2406.09246*, 2024.

[25] C. Y. Kim, C. P. Lee, and B. Mutlu, "Understanding large-language model (llm)-powered human-robot interaction," in *Proceedings of the 2024 ACM/IEEE international conference on human-robot interaction*, 2024, pp. 371–380.

[26] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, "Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning," *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2081–2088, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:261245497

[27] A. Lykov and D. Tsetserukou, "Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model," in *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. IEEE, 2024, pp. 392–397.

[28] J. Ao, F. Wu, Y. Wu, A. Swiki, and S. Haddadin, "Llm-as-bt-planner: Leveraging llms for behavior tree generation in robot task planning," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025, pp. 1233–1239.

[29] X. Chen, Y. Cai, Y. Mao, M. Li, W. Yang, W. Xu, and J. Wang, "Integrating intent understanding and optimal behavior planning for behavior tree generation from human instructions," in *International Joint Conference on Artificial Intelligence*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:269758016

[30] H. Zhou, Y. Lin, L. Yan, J. Zhu, and H. Min, "Llm-bt: Performing robotic adaptive tasks based on large language models and behavior trees," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 16 655–16 661.

[31] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, "Automatic behavior tree expansion with llms for robotic manipulation," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025, pp. 1225–1232.

[32] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[33] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. 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*, vol. 3, no. 6, p. 7, 2023.

[34] O. Honovich, T. Scialom, O. Levy, and T. Schick, "Unnatural instructions: Tuning language models with (almost) no human labor," *arXiv preprint arXiv:2212.09689*, 2022.

[35] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[36] H. Bansal, A. Hosseini, R. Agarwal, V. Q. Tran, and M. Kazemi, "Smaller, weaker, yet better: Training llm reasoners via compute-optimal sampling," in *The Thirteenth International Conference on Learning Representations*, 2024.

[37] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013/

[38] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[39] S. Gugliermo, D. C. Domínguez, M. Iannotta, T. Stoyanov, and E. Schaffernicht, "Evaluating behavior trees," *Robotics and Autonomous Systems*, vol. 178, p. 104714, 2024.

[40] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.