POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Multimodal Behavior Tree Generation: A Lightweight Vision-Language Approach to Robotic Task Planning

Laurea Magistrale in Computer Engineering - Ingegneria Informatica

Author: Cristiano Battistini

Advisor: Prof. Matteo Matteucci

Co-advisor: Dr. Gianluca Bardaro, Riccardo Andrea Izzo

Academic year: 2025-2026

1. Introduction

Robots are taking on increasingly complex roles in domestic and service environments, demanding flexible task planning capable of handling dynamic and unpredictable surroundings. Among the representations proposed for robotic tasks, behavior trees (BTs) have gained traction as a formalism that combines modularity with reactive execution, widely adopted in the ROS2 ecosystem through the BehaviorTree.CPP library[1].

Recently, large language models (LLMs) have been applied to robotic planning, and a few works have shown that LLMs can generate behavior trees in XML format from textual task descriptions. Along this line, BTGenBot [2] demonstrated that compact fine-tuned models can produce valid behavior trees for robotic tasks. However, all existing approaches operate on text alone, and no existing dataset links visual observations and natural language instructions to executable behavior trees. The absence of visual grounding was explicitly noted as an open direction for future work [2]. To the best of our knowledge, no prior work has addressed it with a compact, fine-tuned vision-language model. This work bridges this gap by combining visual perception with lightweight model fine-tuning for behavior tree generation. The main contributions are:

(i) a multimodal dataset pairing visual observations, natural language instructions, and executable behavior trees, constructed from real-world robotic episodes in Open X-Embodiment [6] through a teacher-guided pipeline;

(ii) multiple open-source VLMs (500 M–4 B parameters), fine-tuned via Parameter-Efficient Fine-Tuning and deployable on consumer hardware;

(iii) an offline evaluation showing that a compact fine-tuned model can match the output fidelity of frontier-scale closed-source systems;

(iv) a comprehensive evaluation executing generated behavior trees inside an embodied simulator on realistic household activities, benchmarked against a frontier-scale model that serves as a performance ceiling.

[1] https://www.behaviortree.dev/

2. Related Work

This section reviews behavior trees and existing approaches to their automatic generation.

2.1. Behavior Trees

A behavior tree (BT) is a directed rooted tree whose internal nodes control execution order and whose leaves interact with the robot [1]. At a fixed frequency a *tick* signal propagates from the root downward; each node executes and returns SUCCESS, FAILURE, or RUNNING. The two main control-flow nodes are *Sequence*, which stops at the first child failure, and *Fallback*, which stops at the first child success. *Decorator* nodes such as `RetryUntilSuccessful` and `Timeout` wrap a single child and modify its execution semantics. Leaf nodes are either *Actions*, which command the robot, or *Conditions*, which query the environment. Every BT maps to a unique XML document recognized by BehaviorTree.CPP, the standard execution engine in ROS 2.

2.2. Automatic BT Generation

Existing approaches to automatic BT generation differ in how the model is obtained (fine-tuning on BT data versus prompting a general-purpose model) and in what the model receives as input (text alone or text together with visual observations of the scene).

Compact fine-tuned models. LLM-BrAIn [5] first showed that a 7B-parameter model fine-tuned on 8 500 synthetically generated pairs can produce BTs from text, although no syntactic or semantic evaluation was performed. BTGenBot [2] addressed this gap by training a 7B model on roughly 600 BTs collected from real open-source robotics projects and evaluating with a custom validator, simulation, a physical robot, and LLM-based review. BTGenBot-2 [3] pushed the approach to a 1B model trained on 5 204 augmented pairs, surpassing its predecessor in simulation success rate at a fraction of the cost. All three systems output BehaviorTree.CPP-compatible XML but accept only text: someone must first describe the scene in words.

Prompted large models. A parallel line of work prompts closed-source models such as GPT-3.5 and GPT-4 without fine-tuning (LLM-as-BT-Planner, LLM-BT, IntentBT, BETR-XP-LLM). These systems remain text-only and depend on external APIs, and in some cases the language model never touches the tree structure directly.

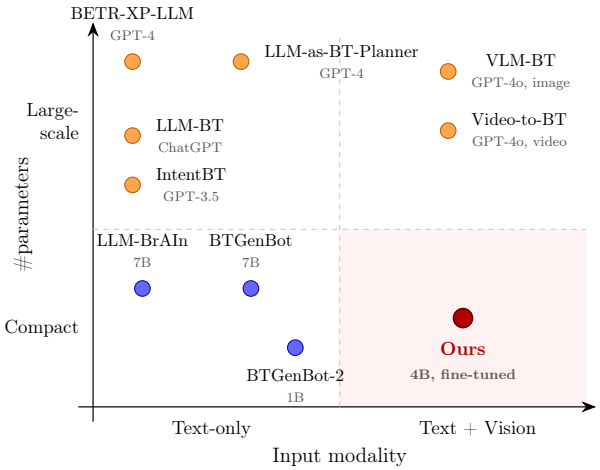

Figure 1: BT generation methods by input modality and model scale.

Adding vision. Vision-language models (VLMs) extend LLMs by feeding visual input through a visual encoder and a projection module into the language model backbone, enabling text generation conditioned on both an image and a textual prompt. Two recent systems apply VLMs to BT generation: VLM-BT prompts GPT-4o with an instruction and a map to produce a BT in JSON format; Video-to-BT extracts reactive BT skeletons from human demonstration videos via GPT-4o or comparable models. Both rely on very large models without fine-tuning on BT data, and neither releases a dataset linking visual observations to executable behavior trees.

Figure 1 summarizes the landscape along two axes: input modality and model scale. No prior work occupies the intersection of visual input and compact fine-tuned model. This work targets precisely that quadrant: a 4B-parameter VLM, fine-tuned on a purpose-built multimodal dataset, that receives an image and a textual instruction and produces BehaviorTree.CPP-compatible XML.

3. Method

Given a single RGB image of the robot workspace and a natural-language instruction, the goal is to generate an executable behavior tree in BehaviorTree.CPP XML format. A

model f_θ must learn the mapping

$$f_\theta : (\mathbf{I},\ t,\ \mathcal{A}) \longrightarrow (SA,\ BT), \qquad (1)$$

where $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$ is the RGB observation, t is the instruction string, $\mathcal{A} \subseteq \mathcal{P}$ is the subset of allowed primitives drawn from a fixed action library $\mathcal{P}$, SA is a structured state analysis of the scene, and BT is the behavior tree in XML.

3.1. Dataset Generation

No existing dataset pairs visual observations with behavior trees, so the training data must be constructed from scratch. Figure 2 provides an overview of the full pipeline.

Source data. We draw on Open X-Embodiment [6], a large-scale collection of over one million robotic episodes across 22 embodiments. From this collection we select 23 sub-datasets covering tabletop manipulation, assistive control, mobile manipulation, bimanual tasks, and kitchen activities. Only the visual observations and the language instructions are retained; all low-level action sequences are discarded.

Frame selection. Each episode is sub-sampled with a stride of 10, and nine maximally diverse frames are selected by a K-center greedy algorithm operating on MobileNetV2 embeddings. These frames are arranged into a 3×3 contact sheet that provides temporal context to the teacher; the student model receives only frame 0.

Teacher pipeline. For each episode, a two-stage LLM pipeline generates the behavior tree labels. First, a *Scene Analysis* stage sends the instruction and the contact sheet to GPT-5-mini (with vision), which returns a structured YAML block containing the target object, destination, expanded instruction, scene context, and expected action sequence. Then, an *Architect* stage receives this analysis together with the image and produces a linear behavior tree in XML, consisting of a single `Sequence` with `Action` leaves only, constrained to a fixed vocabulary of 22 primitives. A purely programmatic *Conformance Validator* parses the XML and checks every action node against the primitive library; on failure, the Architect stage is re-invoked until a conforming tree is produced. Splitting the generation into two LLM calls lets each prompt focus on a single objective; the Scene Analysis also serves the student model, which learns to reproduce it as a chain-of-thought before generating the plan.

Augmentation. Two strategies enrich the base trees before training. *Structural augmentation* injects control-flow constructs (`RetryUntilSuccessful`, `Fallback`, `Timeout`, `SubTree`, `Condition`) into the linear trees. A set of 17 prompt templates (5 targeting a single construct, 12 combining two or more) defines the patterns; for each base tree, the augmentation code selects applicable templates and *samples one* according to a target probability distribution, then sends it to GPT-5-mini, which produces both the modified instruction and the augmented tree. Each template modifies the XML and the textual instruction simultaneously, so the model learns to produce a construct only when explicitly requested. *Lexical augmentation* is then applied independently to both the base and the structurally augmented episodes: it replaces canonical primitive names with synonyms (e.g. `GRASP` → `GRAB`), introduces two-parameter variants for placement actions, and injects rare verb forms at low probability; 50 % of episodes are kept unchanged to preserve a stable learning signal.

Final dataset. The resulting dataset contains 2 433 episodes (2 205 training, 228 evaluation): 1 622 base episodes with linear trees and 811 structurally augmented episodes.

3.2. Fine-Tuning

We fine-tune three open-source VLMs that span an order of magnitude in parameter count: SmolVLM2-500M (500 M parameters), Qwen2.5-VL-3B (3.7 B), and Gemma 3 4B (4.3 B). All three follow the standard VLM architecture: a visual encoder, a projection module, and a decoder-only language backbone.

We adopt QLoRA, which quantises the frozen base weights to 4-bit NF4 precision and trains only low-rank adapters in BFloat16, reducing memory consumption by roughly four times compared to standard LoRA while matching its task performance. We use rank-16 adapters (α=16, no dropout), injected into every linear layer across all three model components (self-attention projections, feed-forward layers, visual encoder, and projection module). Each model is trained for 3 epochs on the 2 205 training samples with an effective batch size of 16, a learning rate of 2×10^{-4}, and a 50-step warmup. Each

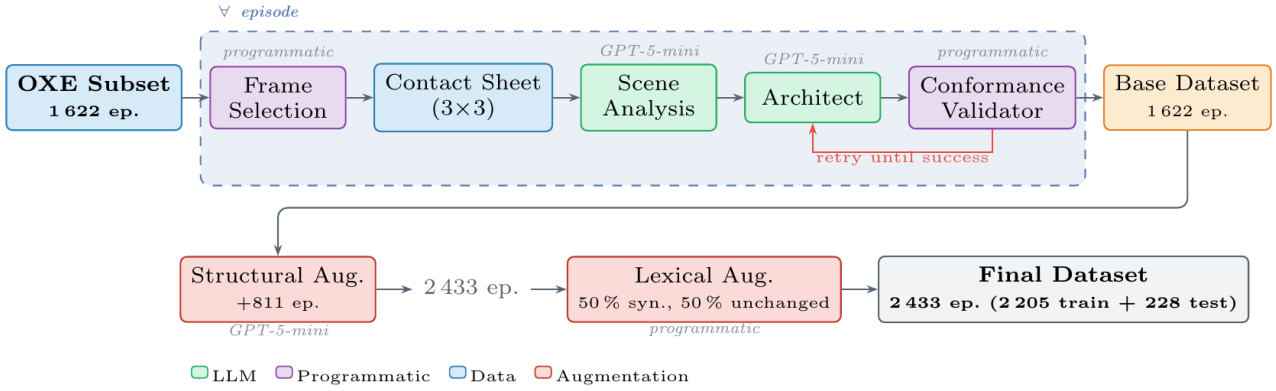

Figure 2: Dataset generation pipeline: teacher stages (top) and augmentation (bottom).

Input (User Turn)

Instruction: *"Place the teapot on the stove"*

Allowed actions: `NAVIGATE_TO`, `GRASP`, `PLACE_ON_TOP`

Output (Assistant Turn)

State Analysis (YAML)

```
scene_analysis:
  target: "teapot"
  destination: "stove"
  expanded_instruction: "Pick up the blue teapot from the
       countertop and place it on the stove"
  scene_context: "Teapot sits on countertop right of sink
      ; stove surface is clear."
  expected_sequence: "Approach the teapot, grasp it, move
       to the stove, set it down."
```

Behavior Tree (XML)

```
<root main_tree_to_execute="MainTree">
 <BehaviorTree ID="MainTree">
  <Sequence>
   <Action ID="NAVIGATE_TO" obj="teapot"/>
   <Action ID="GRASP" obj="teapot"/>
   <Action ID="NAVIGATE_TO" obj="stove"/>
   <Action ID="PLACE_ON_TOP" obj="stove"/>
  </Sequence>
 </BehaviorTree>
</root>
```

Figure 3: Example training sample from the student dataset.

training sample is a two-turn conversation (Figure 3): the user turn presents the initial frame of the episode together with the task instruction and the list of allowed primitives; the assistant turn contains the state analysis YAML followed by the behavior tree XML. All experiments run on a single GPU (NVIDIA L4 or Tesla T4 depending on the model).

3.3. Simulation Environment

Generated behavior trees are executed inside OmniGibson, the reference simulator of the BEHAVIOR-1K [4] benchmark. All episodes use *symbolic execution*: each primitive is realised as an instantaneous state change rather than a physics-based motor command, isolating plan correctness from low-level control variance. This design evaluates plan quality in isolation from controller reliability. A task is considered solved if and only if every goal predicate (BDDL, e.g. `ontop(book, table)`) defined by the BEHAVIOR-1K activity is satisfied; no partial credit is awarded.

4. Experimental Results

The evaluation combines an offline analysis on the held-out split with a simulation-based stage that executes generated trees inside OmniGibson on 15 BEHAVIOR-1K household tasks.

4.1. Offline Evaluation

Each fine-tuned adapter and its corresponding base model are evaluated on the 228-example held-out split (152 linear, 76 with decorators). Without fine-tuning, no base model produces a single valid output in the BT-CPP format: the schema has never been seen during pre-training. The only baseline that shows any XML ability is Qwen2.5-VL, which achieves high generic XML validity, yet even its output never conforms to the BehaviorTree.CPP specification. A LoRA adapter is sufficient to close this gap entirely: Gemma-3 and Qwen2.5-VL both reach 100 % validity, while SmolVLM2 retains a residual failure rate (Table 1). Fine-tuning also cuts inference

time: adapted models produce concise output, whereas base models generate long unstructured text (Gemma-3 baseline is roughly five times slower, with a standard deviation that exceeds the mean).

Table 1: Inference time and syntactic validity (228-example test split).

	SmolVLM2	Gemma-3	Qwen2.5-VL
Adapter (fine-tuned)			
Inf. time (s)	12.7±24.4	20.4±5.5	17.2±4.9
XML valid (%)	88.60	100.00	100.00
BT-CPP valid (%)	87.72	100.00	100.00
Baseline (no fine-tuning)			
Inf. time (s)	39.0±31.6	104.6±114.2	13.9±15.1
XML valid (%)	27.19	17.54	82.89
BT-CPP valid (%)	0.00	0.00	0.00

Structural compliance (whether the tree uses the same control-flow constructs as the reference) and action Jaccard (overlap between generated and ground-truth action sets) expose where the capacity threshold lies (Table 2): on linear examples all three adapters perform well; on decorator examples the two larger models reproduce control-flow constructs reliably, while SmolVLM2 succeeds on only a small fraction. Gemma-3 is the most disciplined on structure; Qwen2.5-VL is the most precise on action selection.

Table 2: Structural compliance and action Jaccard (fine-tuned only).

	SmolVLM2	Gemma-3	Qwen2.5-VL
Struct. match (%)	66.67	96.93	94.74
Linear	142/152	152/152	146/152
Decorator	10/76	69/76	70/76
Action Jaccard	0.886±0.197	0.971±0.129	0.984±0.079

BLEU and ROUGE, two standard metrics that measure token-level overlap with a reference text, are computed on a 30-example subset (10 linear, 20 with decorators); the decorator share is intentionally high because control-flow constructs appear only when the prompt explicitly requests them, so a high overall score signals that the model correctly interprets the instruction and produces the matching XML structure. The fine-tuned Gemma-3 4B matches or marginally exceeds three frontier-scale closed-source models evaluated without fine-tuning (Figure 4): a 4 B-parameter model trained on synthetic data reaches the lexical fidelity of systems orders of magnitude larger.

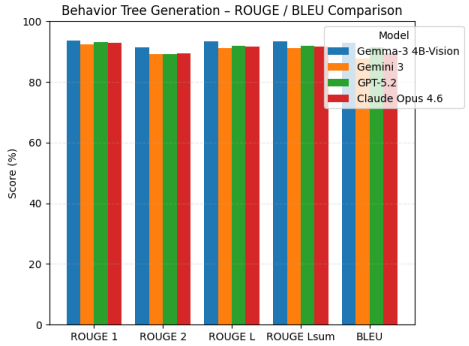

Figure 4: Offline ROUGE and BLEU: fine-tuned Gemma-3 4B vs. frontier models (30 held-out episodes).

4.2. Simulation-Based Evaluation

The 15 tasks span three difficulty levels: seven *Easy*, five *Medium* (open/close logic), and three *Hard* (nested containment or strict ordering). Each model is tested under Chain-of-Thought (CoT), which provides a numbered primitive workflow, and Zero-Shot (ZS), which states the goal with no decomposition.

Fine-tuning is essential: all three base models score 0/15 on every metric, making simulation impossible. After fine-tuning, Gemma-3 and Qwen both reach 100 % BT Valid, fully closing the syntactic gap with GPT-5 (Table 3). The remaining gap is in planning quality: Gemma-3 with CoT approaches but does not match the frontier model. GPT-5 saturates the benchmark in both conditions; this is intentional, as the tasks measure how close lightweight models can get to a known ceiling.

Table 3: Simulation results (15 BEHAVIOR-1K tasks, up to 3 attempts). Best open-source per metric in **bold**.

Model	Var.	Pr.	BT Valid	SR	PASS@3
GPT-5	—	CoT	100%	100%	100%
		ZS	100%	100%	100%
Gemma-3	FT	CoT	100%	**87%**	**93%**
		ZS	100%	73%	80%
	Base	CoT	0%	0%	0%
		ZS	0%	0%	0%
Qwen2.5-VL	FT	CoT	100%	67%	87%
		ZS	100%	47%	67%
	Base	CoT	0%	0%	0%
		ZS	0%	0%	0%
SmolVLM2	FT	CoT	7%	0%	0%
		ZS	27%	0%	7%
	Base	CoT	0%	0%	0%
		ZS	0%	0%	0%

There is a qualitative threshold between 500 M and 3 B parameters: SmolVLM2 is rarely able

to produce structured XML, regardless of how much task-specific training it receives. Once a model crosses this threshold, failures shift from syntactic (broken XML) to semantic (valid XML, wrong plan). The failure hierarchy is consistent: SmolVLM2 cannot produce well-formed XML; Qwen2.5-VL generates valid trees but semantically confused plans; Gemma-3 produces correct plans most of the time but stumbles on ordering violations; GPT-5 never fails. The CoT-to-ZS gap widens on Medium and Hard tasks, where CoT externalises multi-step reasoning the model cannot perform alone; for SmolVLM2 the additional tokens are counterproductive. The two recurring failure patterns are physical-precondition violations (e.g. placing inside a closed container) and object-name hallucination, both increasing in frequency as model size decreases.

5. Conclusions

This work proposed a method for generating behavior trees from visual observations and natural-language instructions using lightweight vision-language models. A multi-stage teacher pipeline constructs the first multimodal dataset from real robotic episodes; open-source VLMs are then fine-tuned via QLoRA on a single consumer GPU, demonstrating for the first time that a compact VLM can produce executable behavior trees from visual input.

After fine-tuning, the 4B model always produces syntactically valid behavior trees in zero-shot on the offline test set, whereas the same model without the adapter never does. Tested on fifteen household activities in simulation, plans generated with Chain-of-Thought prompting reach competitive performance against GPT-5 despite the large gap in model scale; on easy and medium tasks the compact model performs comparably, while the gap widens on hard tasks that require reasoning about implicit physical preconditions. A capacity threshold emerged: below approximately three billion parameters, fine-tuning cannot teach reliable structured output and the dominant failure mode shifts from semantic to syntactic.

The model generates only the plan; low-level actions remain hand-coded. The output is a behavior tree, so the plan stays interpretable and open to human-in-the-loop intervention; it conforms to the BehaviorTree.CPP specification, making the generated XML directly loadable in ROS 2. With four billion parameters the model fits on embedded accelerators without cloud connectivity, and the step to physical deployment becomes a matter of integration, not of re-engineering. All weights, code, and data are open source.

Replacing hand-coded primitives with learned policies, keeping the visual channel open for closed-loop replanning, and enriching the dataset with longer-horizon demonstrations would each address failure modes that the experiments made concrete.

References

[1] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[2] Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci. Btgenbot: Behavior tree generation for robotic tasks with lightweight llms. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9684–9690. IEEE, 2024.

[3] Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci. Btgenbot-2: Efficient behavior tree generation with small language models, 2026.

[4] Chengshu Li, Ruohan Zhang, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024.

[5] Artem Lykov and Dzmitry 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)*, pages 392–397. IEEE, 2024.

[6] Abby O'Neill et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.