



UNIVERSITY OF PADUA
Department of Information Engineering
Bachelor's Degree in Information Engineering

Final Report

Title: Evaluating Robustness in Instruction-Tuned VLMs for Robotic Planning

Supervisor: Prof. Pietro Falco

Candidate: Simone Adami
Student ID: 2105534

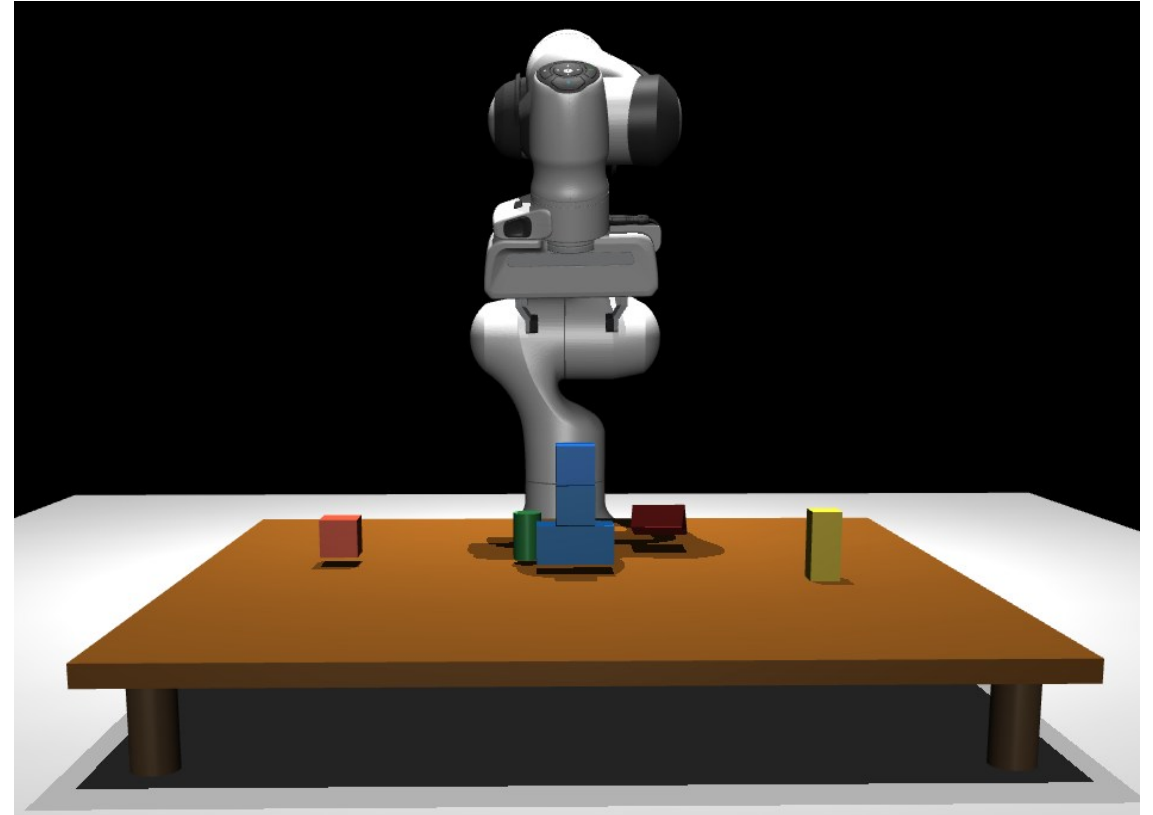
ACADEMIC YEAR 2025/2026



PURPOSE

The core problem this thesis addresses is the investigation regarding how well FineTuned Vision-Language Models (VLMs) perform in robotics when they encounter unpredictable situations outside of their training data. While existing research has successfully used VLMs for robotic planning, a gap remains.

Current models are mostly validated within the strict, safe boundaries of the synthetic data they were trained on. **How these models behave under Out-Of-Distribution conditions, such as scenarios, objects, or tasks they have never seen before?**





LLM

A **Large Language Model (LLM)** is an artificial intelligence system trained on massive amounts of text to understand, generate, and interact with human language. It fundamentally operates as an advanced guessing machine, repeatedly calculating the statistical probability of which word should logically come next in a sequence.

- **Large:** These models contain billions of internal variables called "parameters". They are trained on a significant portion of the public internet
- **Language:** They are optimized specifically to process natural human language and unstructured text.
- **Model:** They use a complex mathematical structure to recognize patterns in how words, sentences, and concepts fit together.

Most modern LLMs, including models like ChatGPT, Claude, and Gemini, are built on a specific neural network architecture called **Transformer**.

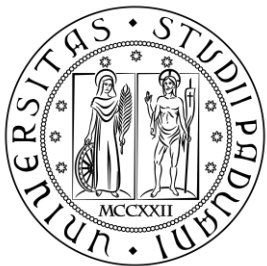


LLM

How an LLM Works:

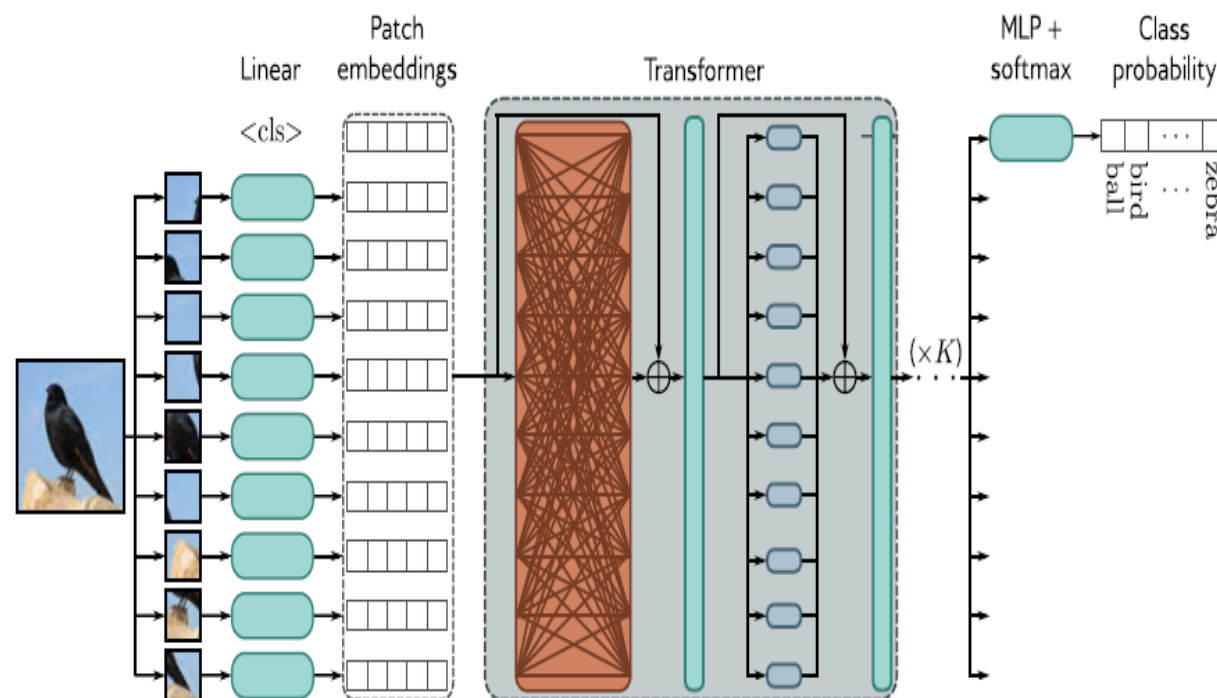
- **Tokenization:** It breaks text down into smaller parts called "tokens" and converts them into numbers.
- **Pattern Recognition (Training):** By processing trillions of these tokens, the neural network learns grammar, facts, and logical structure by mapping out how terms relate to one another in multidimensional spaces.
- **De-Tokenization:** "restore" output tokens as human language text
- **Fine-Tuning:** Adapts a general-purpose model to perform specific tasks instead of generating simply open-ended conversational text.

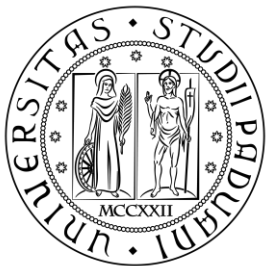
Because they capture deep context , LLMs can perform a wide range of tasks such as translating languages, writing code or summarizing documents.



VLM

- Vision-Language Models (VLMs) are a class of artificial intelligence that bridge the gap between seeing and reading.
- Combines computer vision (the ability to process images) with natural language processing (the ability to understand text), VLMs can analyze both visual and textual data simultaneously.
- This dual capability allows them to "see" an image and hold a conversation about it, enabling tasks like answering specific questions about a picture, and reasoning across multiple modalities.



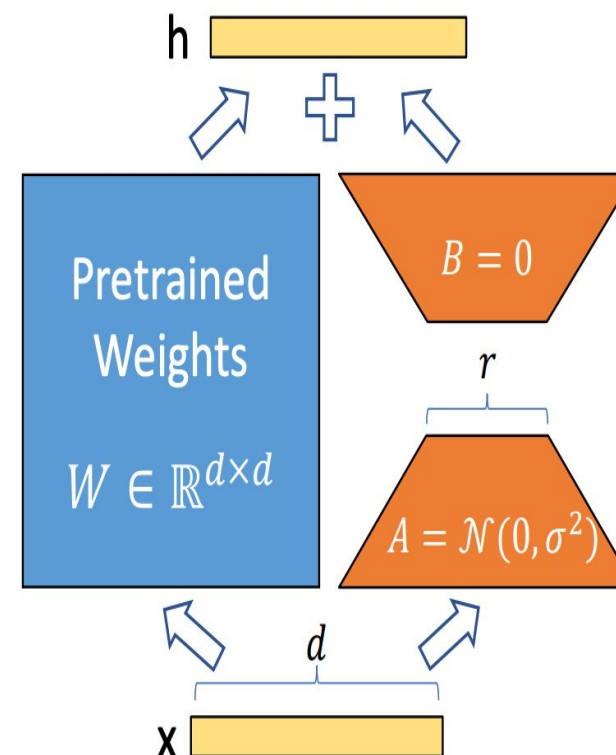


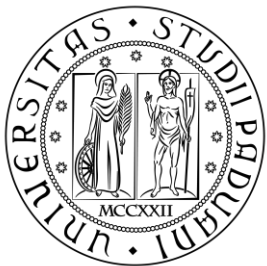
TUNING

Fine-tuning process is used to adapt a general purpose Vision-Language Model (VLM). After that it can, for example, generate logically constrained robotic policies instead of open-ended conversational text.

In this work:

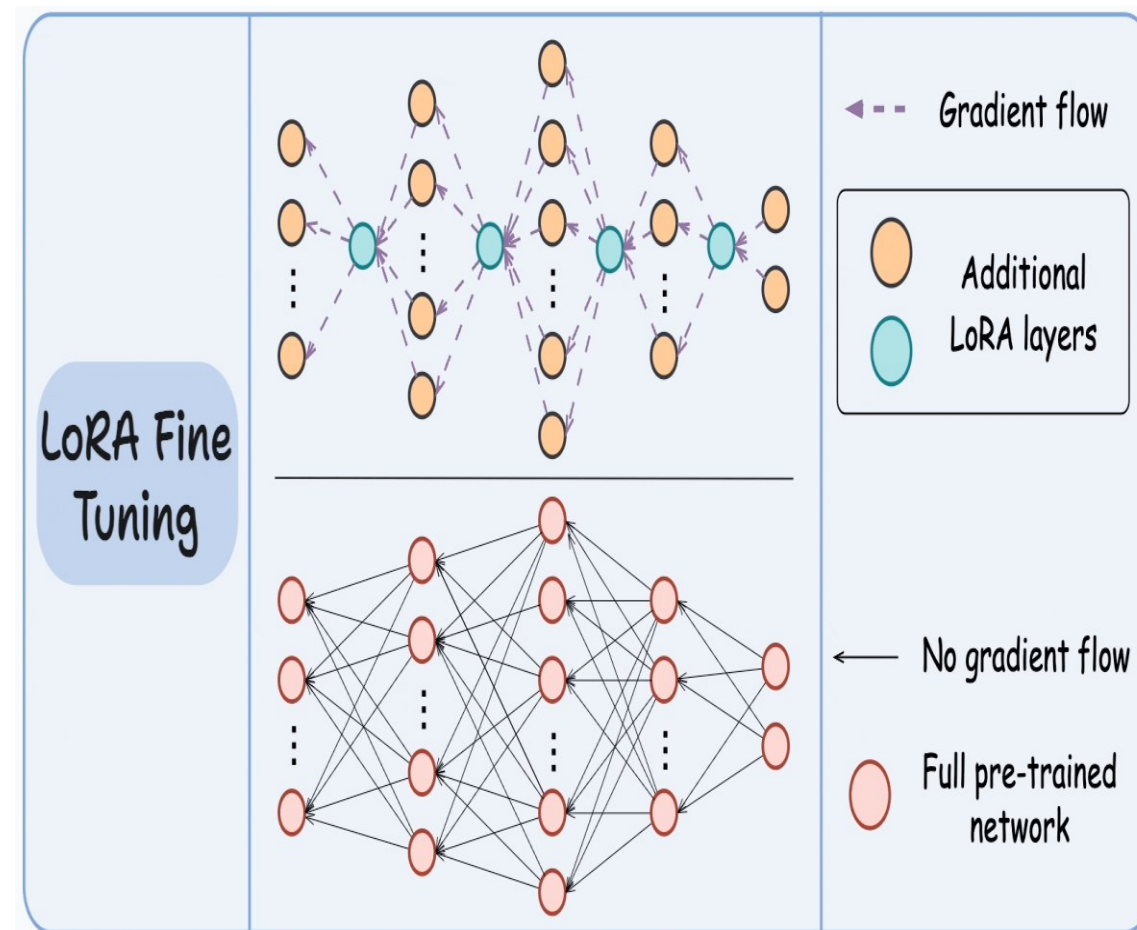
- The fine-tuning process is designed to optimize the mapping between a high-dimensional visual input (the image), a linguistic intent (the user instruction), system constraints (the available robotic actions and rules), and the target symbolic execution trace (the Behavior Tree).
- Pixtral-12B was used, an open-source model consisting of a 400M-parameter vision encoder and a 12B-parameter LLM. To manage the VRAM limitations, a strict context window constraint of 4,400 tokens has been enforced, any training sample that exceeded this limit was discarded to prevent memory errors.

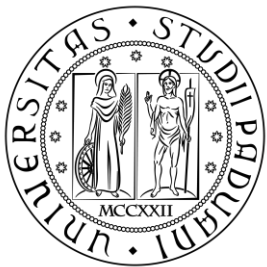




LoRA

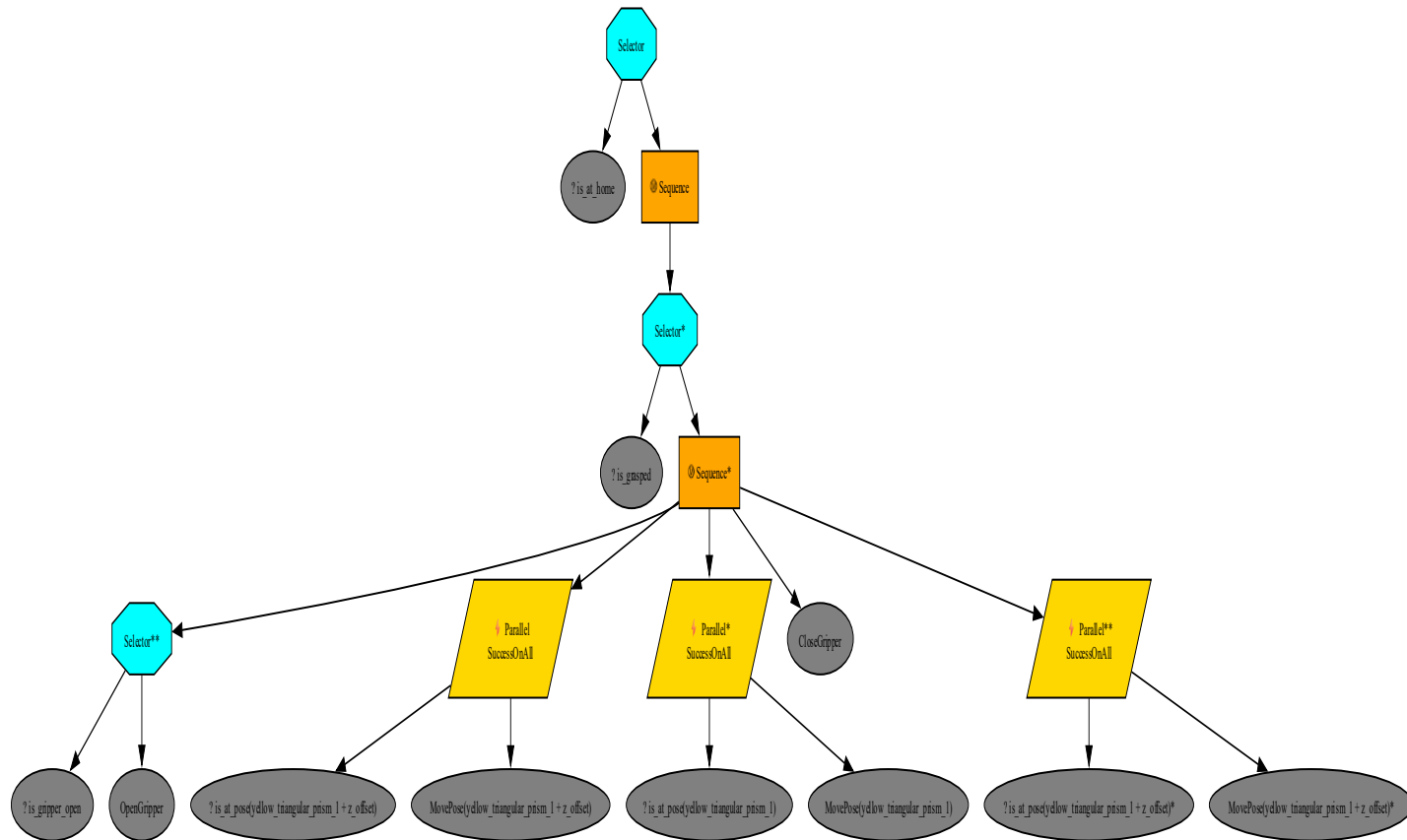
- To Fine-Tune the model LoRA was used
- LoRA is a parameter-efficient fine-tuning method that updates only a tiny fraction of a model's weights rather than the entire neural network.
- **How it works:** A LoRA freezes the original massive neural network so its core knowledge remains untouched. It then injects a tiny, secondary neural network (the "adapter") alongside it. During training, only this "tiny" adapter learns the new information.
- **The benefit:** Because only a fraction of the total parameters is trained, it is possible to train a LoRA on a single graphic card.
- The deployed model was configured with a LoRA rank of 16, a scaling factor of 32, and a dropout rate of 0.05

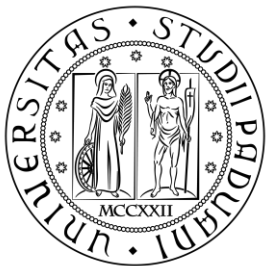




BEHAVIOR TREES

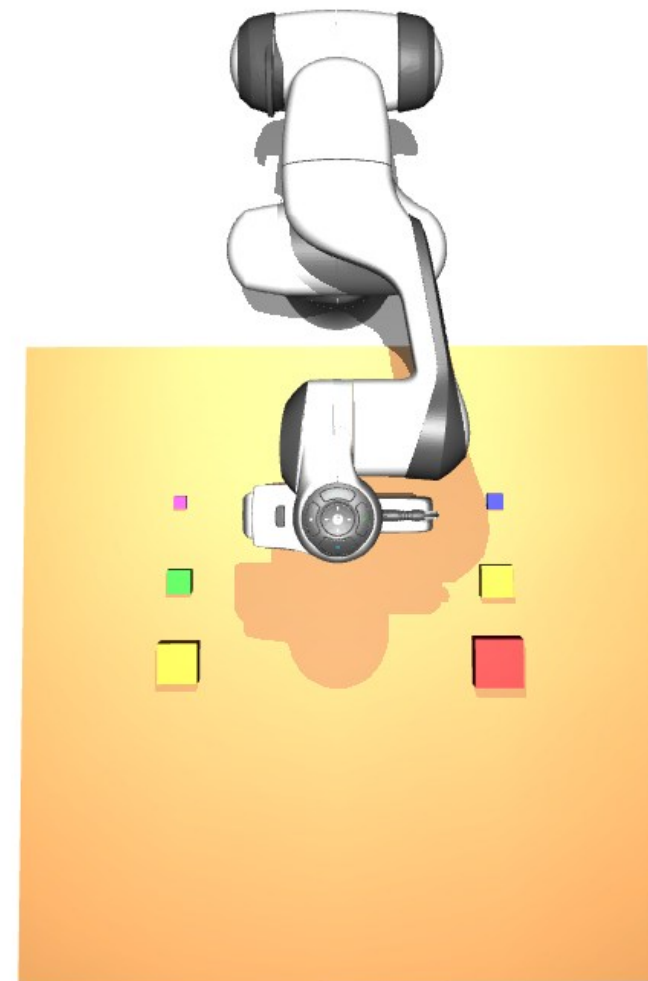
- Behavior Tree is a structured, symbolic policy representation used to govern how a robot makes decisions and completes tasks.
- By decomposing complex tasks into primitives, every step of the robot's decision-making process is fully traceable through the tree's structure. A Behavior Tree is built using a hierarchy of different "nodes" where Internal Nodes consist of Sequence, Selector, and Parallel node types. Leaf Nodes consist of Actions (e.g., MovePose, CloseGripper) and Conditions (e.g., ? is gripper_open).
- If a check fails, the tree's nature allows it to immediately trigger "recovery branches" to adapt to the failure.





MUJOCO

- **MuJoCo** stands for **M**ulti-**J**oint dynamics with **C**ontact.
- It is a physics engine that aims to facilitate research and development in areas that demand fast and accurate simulation of articulated structures interacting with their environment .
- The user defines models in the native MJCF scene description language – an XML file format designed to be as human readable and editable as possible.
- The library includes interactive visualization with a native GUI. MuJoCo can be used to implement model-based computations such as control synthesis, state estimation, system identification, mechanism design, data analysis through inverse dynamics, and parallel sampling for machine learning applications.





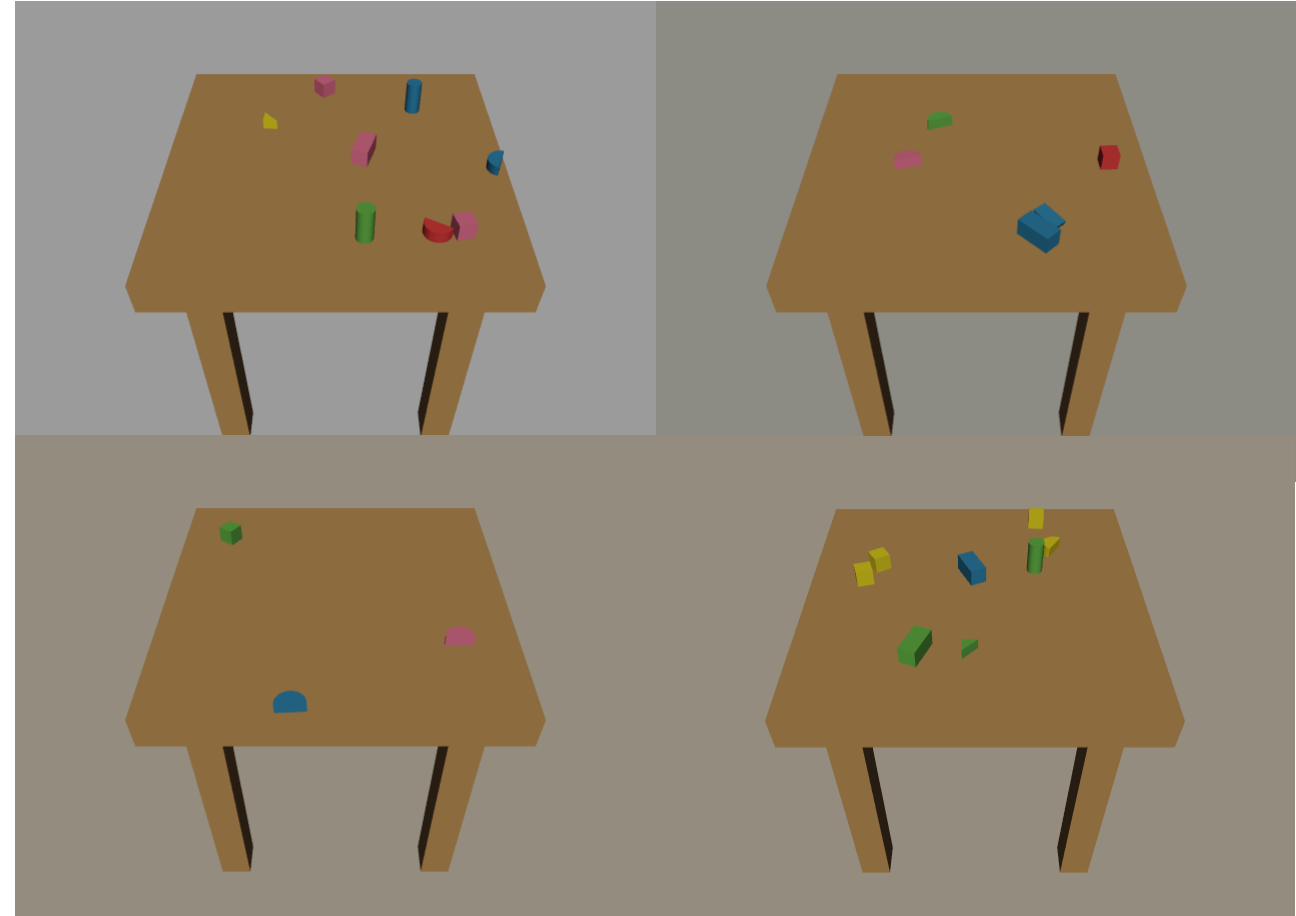
CLUSTER & GPU

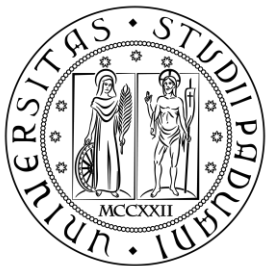
- Simulations were conducted on the DEI cluster, a computing platform equipped with different CPUs and GPUs managed by the Slurm scheduler.
- **Job Allocation:** When a job script is submitted (using command sbatch), a request is handled to the scheduler.
- **The Queue:** Slurm places the request into a queue and constantly monitors the "compute nodes", the servers actually housing the GPUs.
- **Offloaded Processing:** Once the required GPUs are free, Slurm temporarily assigns them exclusively to the job and moves the script to that node. The GPUs process the heavy mathematical calculations entirely on the university's hardware.
- **Output Delivery:** When the script finishes, Slurm automatically saves the output files, trained models, and error logs back to your personal directory on the login node.
- **GPU:** It utilizes thousands of cores to process data in parallel, unlike a CPU which executes complex tasks sequentially. This parallel architecture grants GPUs efficiency for heavy data workloads.



IN DISTRIBUTION

- The first part of this work aimed to investigate the fine-tuning over the Pixtral-12B VLM through a series of 10k samples, based on the recognition and movement of blocks, stacking them based on their colors and shapes.
- This was carried out with two files of JSON type. This file's structure made it possible for the model to recognize what was present inside the images and to have access to a list of processes to be executed to complete the tasks. All this information was not available only with the image or the user request alone.





IN DISTRIBUTION

- To understand better how the trained simulator behaved and having a better understand of the workspace we compare a non-fine-tuned model with one fine-tuned with LoRA.
- These simulations showed a difference between the two models output, the trained one is accurate both in building the tree and understanding the tasks, while the untrained one showed great accuracy despite having poor syntax when building the trees. Having a structured text after fine-tuning allows the code parser to process the behavior tree and execute it on the robot.

```
**Perception Phase**
- **MeasureGripperSites**
  - Ensures the gripper's position is known.
- **MeasureMassWithTorque**
  - (Optional) Measures the mass of the block to ensure it is a blue block.

**Approach Phase**
- **MovePose(pose_of_first_blue_block + z_offset)**
  - Moves to the approach pose above the first blue block.
- **MoveDownUntilContact**
  - Moves downward until contact is detected to ensure the block is grasped securely.
- **OpenGripper**
  - Opens the gripper to prepare for grasping.
- **MovePose(pose_of_target_stacking_location + z_offset)**
  - Moves to the approach pose above the target stacking location.

**Interaction Phase**
- **CloseGripper**
  - Closes the gripper to grasp the blue block.
- **MovePose(pose_of_target_stacking_location)**
  - Moves to the target stacking location.
- **MoveDownUntilContact**
  - Moves downward until contact to place the block securely.
- **OpenGripper**
  - Opens the gripper to release the block.

**Reset Phase**
- **MoveJoints("home")**

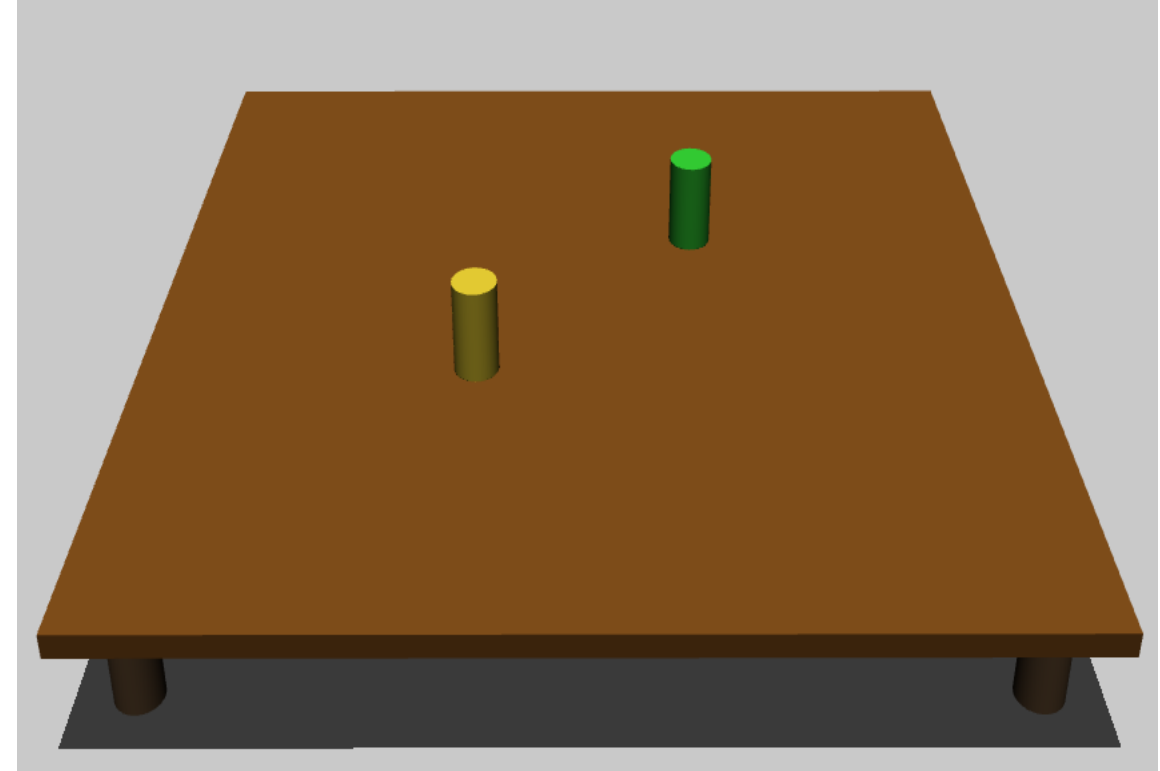
"children": [
  {
    "name": "is_at_home",
    "type": "Condition"
  },
  {
    "children": [
      {
        "children": [
          {
            "name": "is_gripper_open",
            "type": "Condition"
          },
          {
            "name": "OpenGripper",
            "type": "Action"
          }
        ],
        "type": "Selector"
      },
      {
        "children": [
          {
            "name": "is_at_pose",
            "type": "Condition",
            "args": [
              "blue_cube_5 + z_offset"
            ]
          }
        ]
      }
    ]
  }
]
```



OOD MAIN DIFFERENCES

Then, Out Of Distribution (OOD) performances of the tuned model were tested .

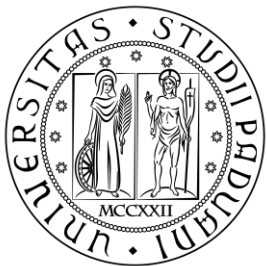
- Trained model lost completely the capability to write simple text (due to LORA adapter) so as a consequence it is no longer able to answer to questions.
- Untrained model is able to identify with better accuracy left and right and also to recognize blocks by their shape or length. With only two blocks both models are able to recognize the cardinal points while with more blocks the fine-tuned model has some difficulties.





OOD TASKS

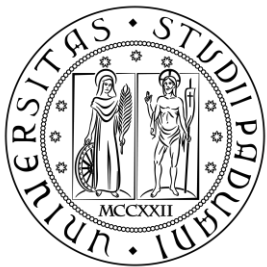
- **Test:** The system was prompted to execute a novel task: determining the effective weight of a block using a dedicated novel action. This evaluated finetuned model's ability to complete tasks and use actions it was not trained for.
- **Observation:** The probabilistic model exhibited a strong bias toward its training distribution, frequently defaulting to standard "pick up and put down" logic rather than prioritizing the measurement, completely omitting that action. It failed to integrate the novel measurement step into the robotic pipeline.
- **Test:** The system was asked to switch the position of two blocks .This evaluated the capacity for algorithmic planning, specifically the ability to create a buffer space to avoid collisions between blocks .
- **Observation:** The fine-tuned model showed architectural limitations. Bounded by its standard "pick up and put down" training , it failed to find the required buffer space , commanding the robot to smash the blocks into the same physical coordinates.



CHANGE OF INSTRUCTIONS

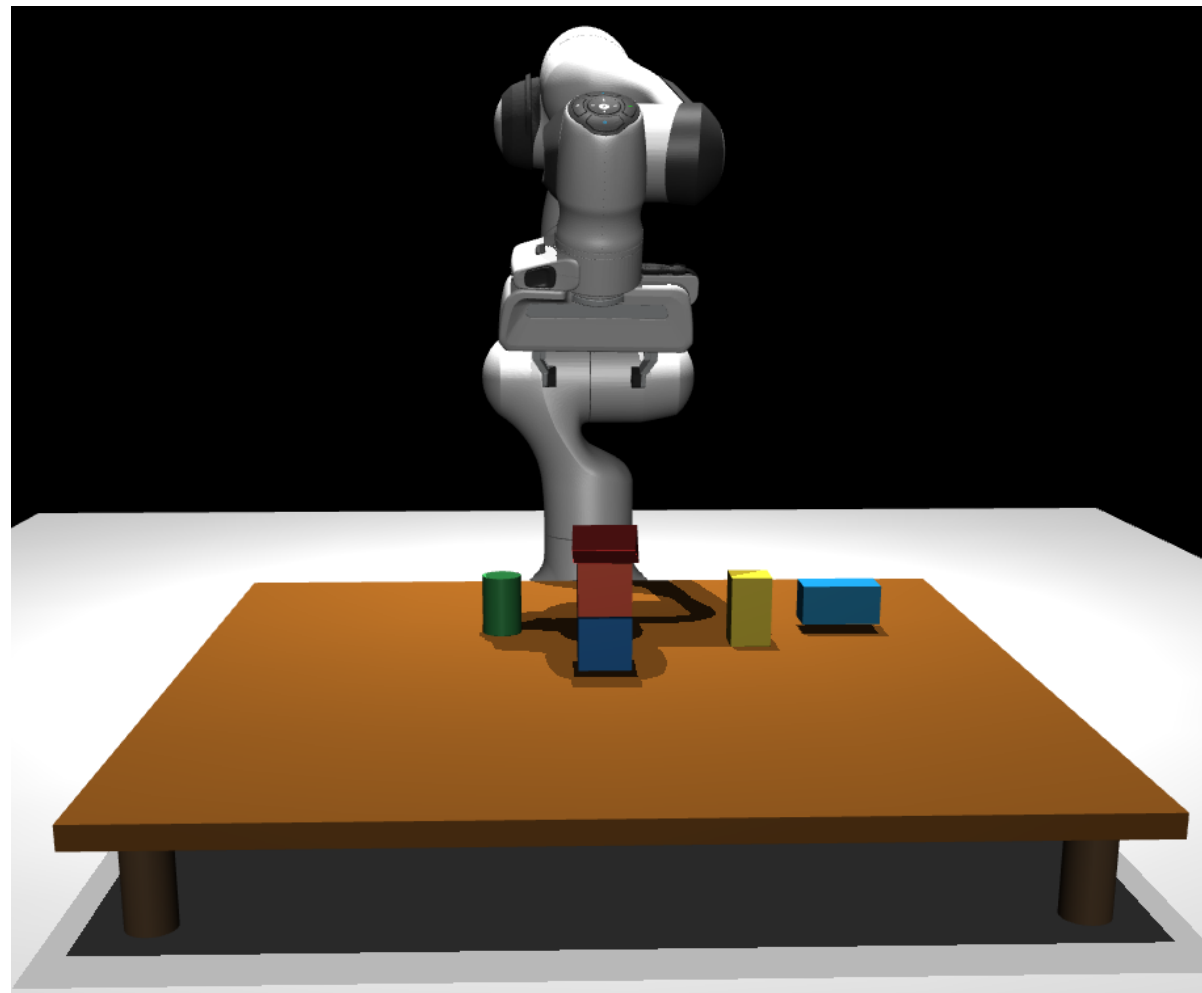
- **Test:** To prove the model learned structural logic rather than memorizing standard robotic libraries, all node instructions were replaced with Italian Cities names.
- **Observation:** The model achieved true logical abstraction. It didn't just memorize standard robotics vocabulary. It learned the syntax and semantic relationships required to solve the task using a completely custom, arbitrary vocabulary.
- **Legend:**
 - Empoli = MoveJoints
 - Catanzaro = is_contact_detected
 - Palermo = is_gripper_open
 - Roma = OpenGripper
 - Treviso = is_at_pose
 - Verona = MovePose
 - Milano = CloseGripper
 - Torino = MoveDownUntilContact.

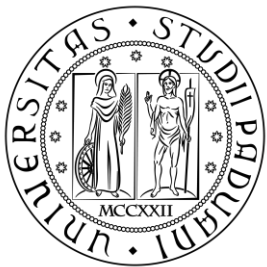
```
978 [VISUALIZED BT (LORA)]:
979 {-} Sequence
980   [o] Selector
981     --> Empoli
982     {-} Sequence
983       [o] Selector
984         --> ? Catanzaro
985         {-} Sequence
986           [o] Selector
987             --> ? Palermo
988             --> Roma
989             // Parallel
990               --> ? Treviso(green_cylinder_smooth_1 + z_offset)
991               --> Verona(green_cylinder_smooth_1 + z_offset)
992             // Parallel
993               --> ? Treviso(green_cylinder_smooth_1)
994               --> Verona(green_cylinder_smooth_1)
995             --> Milano
996             // Parallel
997               --> ? Treviso(green_cylinder_smooth_1 + z_offset)
998               --> Verona(green_cylinder_smooth_1 + z_offset)
999             // Parallel
1000               --> ? Treviso(yellow_cylinder_smooth_0 + z_offset)
1001               --> Verona(yellow_cylinder_smooth_0 + z_offset)
1002             // Parallel
1003               --> ? Catanzaro
1004               --> Torino
1005             --> Milano
1006             // Parallel
1007               --> ? Treviso(yellow_cylinder_smooth_0 + z_offset)
1008               --> Verona(yellow_cylinder_smooth_0 + z_offset)
1009             --> Empoli(home)
```



LONG-HORIZON PLANS

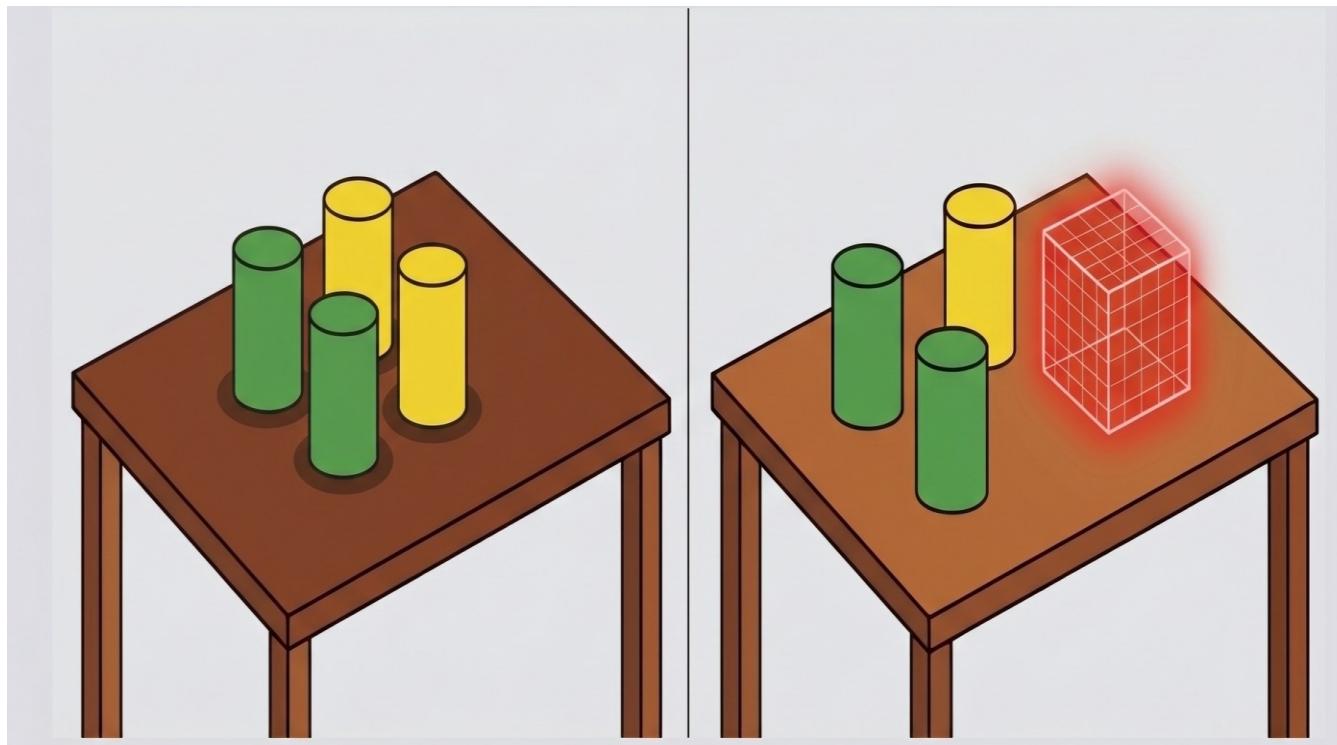
- **Test:** The system was instructed to "build a house" using multiple blocks. This evaluated whether the model could perform long-horizon, multi-stage semantic planning and sequence complex architectural structures without exceeding its hardware and context constraints.
- **Observation:** The fine-tuned model only achieved a 50% JSON validity rate. The massive sequence of actions required exceeded the model's token limits, resulting in truncated JSON outputs and cluster timeouts. Since any training sample exceeding a 4,400-token limit was systematically discarded during fine-tuning, the model never learned to orchestrate long-horizon manipulation sequences. Furthermore, the complexity of the task exceeded computational limits.

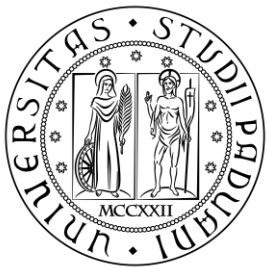




HALLUCINATIONS

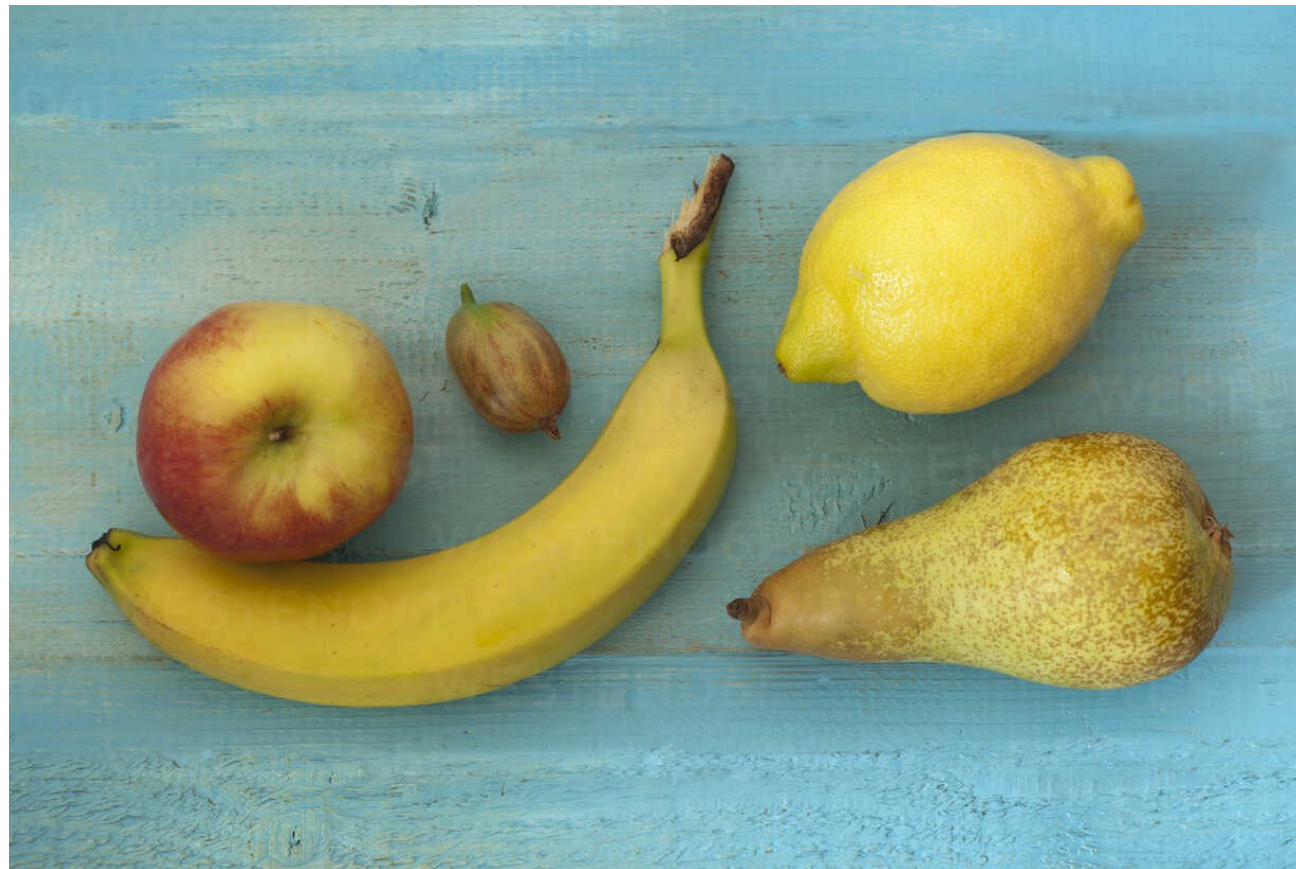
- **Test:** The system was instructed to "pick the red block" in a visual scene where no red block exists (only yellow and green cylinders).
- **Observation:** The LoRA model completely failed conceptually (despite getting 100% in tree generation), while the base model correctly identified the missing object and refused the command 40% of the time, the LoRA model blindly hallucinated a red block to force a successful tree generation. The finetuning process over-optimized the model on successful scenarios, turning it into a blind compiler that prioritizes perfect code syntax over visual reality since the finetuning dataset lacked negative examples or rejection criteria.





TEST ON DIFFERENT OBJECTS

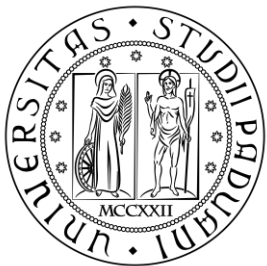
- **Test:** The system was instructed to "pick the yellow fruits" in a scene with different kind of fruits, requiring the model to dynamically identify unseen objects.
- **Observation:** The trained model demonstrated structural abstraction on a zero-shot task. While the base model reasoned correctly about the novel objects but with text, the LoRA model successfully mapped its learned Behavior Tree syntax onto entirely new untrained objects.





RESUME TABLE

	Mean Time (distribution)	Mean Time (out of distribution)	JSON rate (distribution)	JSON rate (out of distribution)	Logical correctness (distribution)	Logical correctness (out of distribution)
Untrained	58.74 s	46.29 s	0 %	0 %	66 %	70 %
Trained	203.40 s	83.24 s	100 %	76.4 %	100 %	25.9 %



CONCLUSION

Key Out-of-Distribution observations:

- The fine-tuned LoRA model demonstrated **structural robustness**, successfully applying strict Behavior Tree syntax to completely untrained objects.
- Because the fine-tuning process is over-optimized for successful task completion, the LoRA model **lost** the base model's **conversational ability**.
- The LoRA model forcefully **hallucinated phantom targets** to produce a syntactically perfect, yet physically impossible, execution tree.
- When prompted to execute a **novel task** the model exhibited a strong **bias toward its training distribution**.
- Long-horizon instructions pushed the model far beyond the strict **4,400-token limit**.

Future Improvements:

- Introduce "impossible" visual scenes into the training dataset so the model learns to output safe rejection commands instead of hallucinating objects.
- Train the model on a wider variety of robotic primitives (e.g., measuring, pushing, sorting) so it learns true causal integration rather than defaulting to memorized pick-and-place routines.
- Increase the 4,400-token training threshold to allow the model to learn and execute long-horizon, multi-stage actions without crashing.