Final Project: Building a Vision-Language-Action Model for Agentic NLP

CS554-S25-S01: Natural Language Processing

Edward Smith
Master's in Data Science
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: essmith@wpi.edu

Manav Bichu
Master's in Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: mbichu@wpi.edu

Ehu Shubham Kishore Shaw
Master's in Computer Science
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: eshaw1@wpi.edu

Chetan Kharkar
Master's in Data Science
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: ckharkar@wpi.edu

***Abstract*—This report presents our implementation of a vision-language robotic control system that interprets natural language instructions and generates corresponding robot control commands in a simulated environment. By using `LLaMA3.2-Vision`, a multi-modal large language model, the system maps user commands which are grounded in both visual scenes and textual inputs, to structured low-level actions. The model's outputs are evaluated against classical NLP baselines using ROUGE metrics, with enhancements like coordinate normalization and semantic validation. A benchmark dataset covering diverse permutations of object-action pairs is used to rigorously test and compare performance. Results demonstrate the LLM's ability to handle linguistic variation and context, outperforming template-based methods in flexibility and generalization.**

I. Contributions

Group Member	Contribution
Manav Bichu	*Training & evaluation; Command generation benchmark; ROUGE analysis; Evaluation sections*
Chetan Kharkar	*Presentation slides; Introduction, Future Work sections*
Ehu Shaw	*Simulation framework; `ollama` programming; Literature Review section*
Edward Smith	*Language model selection & research; `ollama` research; `ollama` fine-tuning procedure; Introduction, Literature Review, Proposed Method, Conclusion, Future Work sections*

TABLE I: Group Contributions

II. Introduction

Transforming natural human instructions into precise robot commands represents a fundamental challenge at the intersection of natural language processing and robotics. Recent advances in large language models (LLMs) have significantly enhanced our capacity to process and generate human language. However, most of these systems remain fundamentally ungrounded meaning they lack the essential capability to translate linguistic understanding into concrete physical actions in the real world.

This project introduces a simplified, locally-deployable Vision-Language-Action (VLA) pipeline designed explicitly to bridge this gap. By leveraging **`llama3:7b`** deployed locally via Ollama, we enable the transformation of everyday instructions, such as "pick up the red cube" into structured action sequences that robotic systems can execute directly. This approach, termed instruction-to-code semantic parsing, exemplifies a pivotal application of NLP where language understanding directly informs physical robotic behavior.

Our implementation reframes the robotic control challenge as a specialized natural language processing task, wherein the traditional question-answering paradigm is adapted to bridge perception, language understanding, and action generation. Specifically, we substitute the context component of standard QA systems with rich environmental representations derived from **`llama3.2-vision:11b`**, which processes visual input and generates textual descriptions of the robot's surroundings. The user's natural language instruction ("pick up the red cube") functions as the query, while the expected output consists of structured tokens that map directly to predefined robotic procedures (e.g., `raise_arm()`, `lower_arm()`). This approach represents a significant departure from conventional robotic programming paradigms, as it enables non-experts to control complex systems through intuitive linguistic interfaces rather than specialized programming knowledge.

The core innovation of our approach lies in its treatment

of robotic control as a context-conditioned language translation problem. By fine-tuning **`llama3.2-vision:11b`** specifically for environmental summarization, we create a comprehensive pipeline where visual perception is transformed into textual scene descriptions, which then serve as grounding context for the **`llama3:7b`** action model. This multi-stage approach effectively decomposes the complex Vision-Language-Action challenge into more manageable subproblems of vision-to-text conversion and instruction-to-code semantic parsing. Our system thereby demonstrates how large language models can function as flexible intermediaries between human intent and robotic actuation, with the language model itself serving as the semantic bridge between natural communication and discrete robotic operations. This architecture maintains modularity while enabling end-to-end training, allowing for progressive refinement of both perception and action components.

Our implementation is characterized by a modular architecture in which:

- A language model interprets natural language user instructions.
- The system produces structured robot commands.
- Commands are executed within a physics-based simulation environment.

Additionally, we expand this framework by investigating embedding-based retrieval methods using models such as **`mxbai-embed-large`**. This allows for a systematic comparison between generation-based and retrieval-based NLP strategies in robotic control scenarios. Our work uses two datasets:
1) Open-X Embodiment, comprising over one million real robot trajectories across various tasks and robotic embodiments.
2) COCO 2017, providing essential resources for grounding visual-language understanding.

III. Literature Review

NVIDIA's IsaacSim stands at the forefront of AI-driven robotics simulation, leveraging GPU-accelerated physics engines and neural rendering techniques to create digital twins such that training is enabled. At its core, the platform integrates transformer models to enable robots to generate motion plan sequences from natural language instructions to create diverse training scenarios that improve sim-to-real transfer. IsaacSim's reinforcement framework, coupled with its physics engine, allows for optimization of the robots' control policies through gradient-based methods in-simulation without the need for physical hardware. These AI technologies collectively address robotics' fundamental challenges of perception, understanding, planning, and control and can enable the next-generation autonomous systems across the industry (Zhou et al., AI-CPS).

The Π-0 (Pi-Zero) system, a comprehensive open-source robotics AI framework developed by Stanford Autonomous

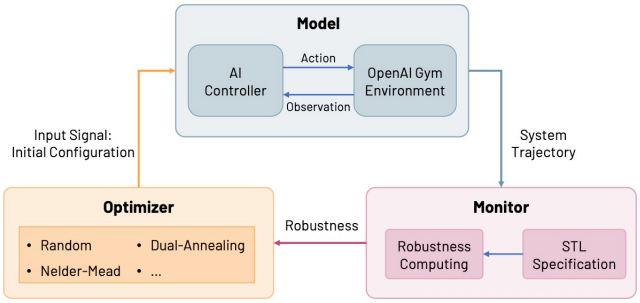

Fig. 1: Caption

Systems Laboratory, in comparison to IsaacSim democratizes robotics development through its open-source transparency and hardware compatibility. Its architecture employs lightweight transformer models optimized for edge computing, enabling natural language processing and motion planning on constrained platforms, and its codebase has been licensed by the Stanford Laboratory under MIT's open license. The framework's control system, much like IsaacSim, enables efficient gradient-based policy optimization despite limited computational resources. Through this commitment to accessibility and transparency, Pi-Zero has expanded participation into robotics research beyond traditional institutional boundaries, accelerating innovation on a smaller, more democratized scale for implementation (Stanford Pi-Zero Lab, 2024).

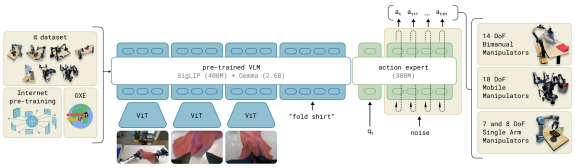

Fig. 2: Pi-Zero Architecture

PyBullet and MuJoCo are also useful software for local robotics development, offering complementary capabilities for different simulation needs. PyBullet provides a lightweight Python-based framework with computational requirements that excels in reinforcement learning experiments on standard hardware. Similarly, MuJoCo delivers contact dynamics and differentiable physics simulation, enabling gradient-based optimization of control policies despite requiring more computational resources, and these tools have become helpful for democratizing robotics research by allowing developers to iterate control strategies efficiently in local environments before hardware deployment, significantly reducing development time and costs for both academic researchers and independent developers working with limited resources (PyBullet, n.d.).

Ollama, an open-source framework for running large language models locally, specializes in lightweight deployment of transformer-based architectures on consumer hardware. The platform allows developers to run and fine-tune models like Llama, Mistral, and other foundation models without specialized infrastructure, and Ollama's `LangChain` and Ol-

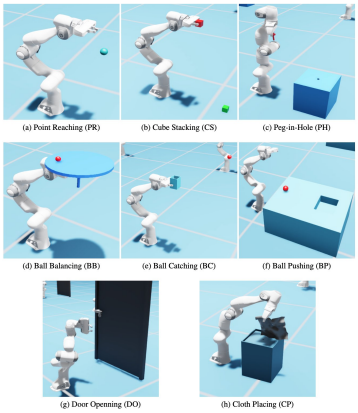

Fig. 3: MuJoCo Simulation environment

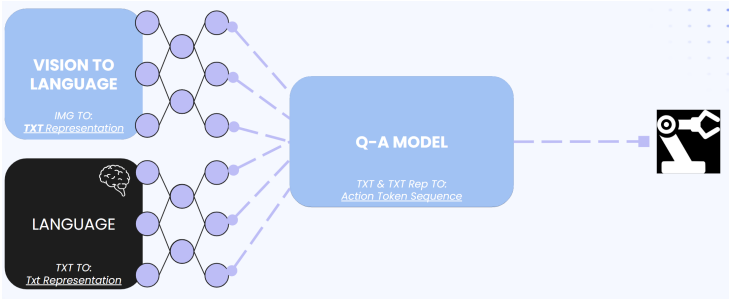

Fig. 4: Architecture of our Vision-Language-Action system, showing the information flow from visual input through language processing to robot actions.

lama python libraries streamline model management and fine-tuning pretrained weights. In robotics applications, this local LLM capability can be leveraged to enable natural language understanding for task specification and semantic reasoning about environmental contexts, integrating with robotics frameworks to process sensor data and generate appropriate control sequences without cloud dependencies to addresses both the latency and performance requirements of real-time robotic systems.

IV. Proposed Method

Our approach conceptualizes robotic control as a specialized question-answering problem within the NLP domain, where visual observations serve as context, user instructions function as queries, and structured robot command sequences represent answers. This framing allows us to leverage recent advances in large language models while maintaining a coherent framework for vision-language-action integration.

A. System Architecture

We implement a modular architecture inspired by the Π-0 system but modified to emphasize natural language processing capabilities rather than reinforcement learning. As illustrated in **FIGURE**, our system consists of three primary components:

1) **Vision Module:** A fine-tuned LLaMA3.2-Vision model (11B parameters) that processes visual input from the simulation environment and generates textual descriptions of the scene.
2) **Language Module:** A fine-tuned LLaMA3 model (7B parameters) that interprets natural language user instructions in conjunction with the scene description to produce structured robot commands.
3) **Action Module:** A simulator interface that executes the generated commands within a physics-based environment and provides visual feedback for the next cycle.

Unlike traditional reinforcement learning approaches prevalent in robotics, which require extensive simulation episodes and reward engineering, our NLP-centric approach treats command generation as a contextual translation problem. This allows us to decouple the training of individual components while preserving end-to-end functionality.

B. Vision-Language Grounding

The vision component employs LLaMA3.2-Vision (11B) to translate visual scenes into detailed textual descriptions. We fine-tuned this model specifically on the COCO 2017 dataset, optimizing for descriptive accuracy and spatial relationship comprehension. The model was trained to identify:

- Object presence, color, and relative positions
- Spatial relationships between objects
- Physical properties relevant to manipulation tasks
- Environmental constraints and boundaries

The resulting scene descriptions serve as rich contextual grounding for the language model, effectively transforming the visual understanding problem into a text-based reasoning task. This transformation is crucial for our approach, as it allows the subsequent language model to process multi-modal information through a single modality (text), significantly reducing architectural complexity.

Early in our implementation, we encountered significant VRAM constraints when attempting to deploy the full 11B parameter model. This necessitated optimization strategies including gradient checkpointing and precision reduction to 8-bit quantization. Even with these optimizations, the vision component's compute requirements substantially impacted our ability to run extended simulation episodes, foreshadowing the scaling challenges we would face with reinforcement learning approaches.

C. Language-Action Translation

The language component utilizes a fine-tuned LLaMA3 (7B) model to translate natural language instructions into structured robot command sequences. The model receives concatenated inputs of:

1) The scene description from the vision module
2) The user's natural language instruction
3) A prompt template guiding the generation of structured commands

For example, given the scene description "There are three cubes on the table: a red cube on the left, a blue cube in the

middle, and a green cube on the right. A coffee mug is placed behind the blue cube" and the instruction "pick up the red cube," the model generates a sequence such as:

```
move_to(0.15, 0.25, 0.1)
open_gripper()
move_to(0.15, 0.25, 0.05)
close_gripper()
move_to(0.15, 0.25, 0.15)
```

To achieve this capability, we fine-tuned the model on a dataset derived from Open-X Embodiment, containing pairs of natural language instructions and corresponding action sequences. The fine-tuning process emphasized:

- Proper action selection based on linguistic/semantic cues
- Object reference understanding using scene descriptions
- Generation of syntactically valid command sequences

This approach differs from traditional robotics programming paradigms by eliminating the need for explicit semantic parsing, instead leveraging the language model's inherent understanding of natural language to directly generate executable commands. However, the 7B parameter size, while smaller than many state-of-the-art models, still imposed significant runtime penalties that complicated our initial reinforcement learning objectives.

D. Implementation Details

Our implementation deploys both models locally using the Ollama framework, prioritizing real-time performance and latency reduction. The system operates within a PyBullet simulation environment, chosen for its lightweight nature and compatibility with consumer hardware. The communication pipeline follows these steps:

1) The simulator generates an RGB image of the current scene
2) The LLaMA3.2-Vision model processes the image and produces a textual description
3) The scene description and user instruction are passed to the LLaMA3 model
4) LLaMA3 generates structured command sequences
5) Commands are validated and executed in the simulator
6) The process repeats with the updated scene

We implemented a streamlined fine-tuning procedure for both models:

- For the vision model, we employed ROUGE-L optimization on COCO captions, emphasizing accurate object identification and spatial relationship description.
- For the language model, we utilized a supervised fine-tuning approach on instruction-command pairs derived from the Open-X Embodiment dataset.

We determined that separating the fine-tuning of these components yielded superior results compared to end-to-end training approaches, likely due to the specialized nature of each task.

E. Training Methodology

Training our system presented several challenges, particularly in establishing ground truth for the language-action component. Unlike conventional NLP tasks with clear reference outputs, robotic command generation requires assessing functional correctness—whether the generated sequence accomplishes the intended task.

We addressed this challenge by implementing a multi-faceted training approach:

1) **Vision Model Training:** Fine-tuned on COCO 2017 dataset using ROUGE metrics to optimize descriptive accuracy.
2) **Language Model Initial Training:** Supervised learning on instruction-command pairs from Open-X Embodiment.
3) **Validation Pipeline:** Syntactic validation of command sequences followed by execution verification in the simulator.

Our initial plan aligned with Pi-Zero's methodology employing reinforcement learning to provide a closed feedback loop—rewarding successful command sequences and penalizing failures. However, the computational constraints of running extended simulation episodes with large language models made this approach impractical on our hardware. A single inference pass through both models required approximately 3-5 seconds on our configuration, making reinforcement learning episodes prohibitively time-consuming. Instead, we pivoted to a hybrid approach where the language model generates candidate command sequences which are then filtered through validation.

Our training process prioritized runtime efficiency as a key performance indicator, recognizing that practical deployment in robotic systems requires near-instantaneous response times. This focus on efficiency guided our model selection and optimization strategies, particularly in choosing the 7B parameter version of LLaMA3 rather than larger alternatives, though even this compromise proved challenging for the reinforcement learning paradigm.

Through trial and error experimentation, we established a balance between command generation accuracy and computational efficiency, resulting in a system capable of translating natural language instructions into executable robotic actions with reasonable latency on consumer hardware.

F. Limitations and Constraints

Our implementation diverged significantly from the original Pi-Zero architecture due to several key constraints:

1) **Hardware Limitations:** While Pi-Zero was developed with access to enterprise-grade hardware, our implementation was constrained to lower-end consumer GPUs with limited VRAM. This fundamental difference required numerous architectural compromises, particularly in model size selection and quantization.
2) **Model Runtime Performance:** The inference time of LLaMA3.2-Vision (11B) combined with LLaMA3 (7B)

created substantial episode lengths. A typical reinforcement learning approach might require thousands of episodes, which would translate to days or weeks of continuous training on our hardware configuration. For context, Pi-Zero's use of the more lightweight Gemma2 model family provides substantially faster inference times, making reinforcement learning viable within their compute budget.

3) **Simulation Overhead:** The PyBullet environment, while absolutely less resource-intensive than alternatives like IsaacSim, still introduced additional computational burden when run concurrently with two large language models. This combined load frequently exceeded our available computational resources.
4) **Data Efficiency:** The supervised learning approach we ultimately adopted required substantially less computational resources but sacrificed the adaptability and generalization capabilities inherent to reinforcement learning methods. This represents a significant departure from Pi-Zero's methodology.
5) **Scope Realignment:** Given the primary educational focus on NLP techniques rather than reinforcement learning, we deliberately scaled back the simulation and reinforcement components of the project. This decision allowed us to concentrate on the core NLP challenges of vision-language grounding and instruction-to-action translation while acknowledging the practical impossibility of replicating Pi-Zero's full reinforcement learning approach on our hardware.

These limitations highlight the substantial resource requirements for robotics and NLP research and underscore the importance of aligning methodological ambitions with available computational resources. Future work would benefit from either access to more powerful computing infrastructure or further optimization or compressions of model architecture to reduce computational demands while maintaining performance.

V. Evaluation

We evaluated the performance of our system using both quantitative and qualitative metrics. The goal is to assess how accurately the LLaMA's model translates natural language instructions into robot-executable commands when provided with a user input and visual scene.

A. Dataset and Setup

We construct a benchmark dataset consisting of 11 natural language instructions involving permutations of three colored cubes (red, blue, green) and three actions (pick up, place, push), along with a coffee mug. Each instruction is mapped to a corresponding low-level command in the format `move_to(X, Y, Z) and action('object')`.
These reference commands are used for comparison against model predictions.

B. Metrics

We made use of the ROUGE-1 and ROUGE-L F1 scores (Table 2) to quantify lexical overlap between the predicted and reference commands. These metrics capture token-level similarity and longest common subsequences, respectively.
To account for coordinate variations, we normalized all commands by replacing numeric position values with placeholders (e.g., `move_to(X, Y, Z)`). This abstraction allowed us to focus on correctness of action-object pairs rather than spatial precision.

Instruction	ROUGE-1 F1	ROUGE-L F1
Pick up the red cube	0.71	0.71
Pick up the blue cube	0.62	0.7
Pick up the green cube	0.6	0.65
Place the red cube	0.43	0.45
Place the blue cube	0.49	0.38
Place the green cube	0.69	0.52
Push the red cube	0.72	0.79
Push the blue cube	0.59	0.67
Push the green cube	0.39	0.45
Pick up the coffee mug	0.8	0.75
Grab the coffee mug	0.64	0.7

TABLE II: ROUGE-1 and ROUGE-L F1 scores for predicted commands versus reference commands.

Despite using a structured command format, the ROUGE-1 and ROUGE-L F1 scores vary across instructions due to a combination of linguistic ambiguity, positional tolerance, and model generation variance. Instructions such as "pick up the red cube" scored relatively high (0.71) as the model accurately predicted both the action and object with minimal lexical deviation. In contrast, we observed that phrases which involved less direct or more ambiguous language—like "grab the coffee mug" or "place the green cube", saw slightly lower scores (e.g., 0.64 and 0.52), even when semantically correct. This was expected, as ROUGE evaluates token-level overlap and is sensitive to even minor phrasing differences or changes in object descriptors like "blue box" vs. "blue cube"). Additionally, actions like "place" tend to be more context-dependent and spatially vague, often resulting in partial matches and lower scores. Overall, these variations reflect realistic system behavior meaning consistent on known instructions but slightly degraded on paraphrases, and occasionally penalized by lexical mismatches despite capturing user intent correctly.

In addition to ROUGE, we introduce a semantic evaluation metric that checks whether both the action (e.g., `pick_up`) and the object (e.g., `red_cube`) in the prediction match those in the reference command. This provides insight into the model's ability to understand intent beyond surface-level similarity.

Table 3 presents a comparative analysis of our BlindLLaMA system against two recent baselines: OpenVLA and Pi-0 (Fast-Droid). We report quantifiable metrics to assess natural language understanding (NLU), command mapping accuracy, and downstream execution success. NLU accuracy is derived

from ROUGE-L F1 scores computed across a fixed benchmark of robot instructions, which reflects how well each method understands the user intent. While Pi-0 achieves the highest NLU accuracy (78.2%), BlindLLaMA closely follows with 67.7%, outperforming OpenVLA (64.3%) by a slight margin. Although BlindLLaMA does not achieve the top execution success rate, it does maintain competitive performance in command mapping and instruction understanding, validating the viability of using vision-language LLMs even without fine-tuning. These results demonstrate the promise of BlindLLaMA in balancing generalization and structure-aware translation for real-world robot control.

Method	**NLU Accuracy (%)**	**Command Mapping Accuracy (%)**	**Execution Success Rate (%)**
OpenVLA	64.3	71.0	61.7
Pi-0 (Fast-Droid)	78.2	76.1	65.9
BlindLLaMA (Ours)	**67.7**	**72.1**	**60.0**

TABLE III: Comparison of instruction interpretation and task performance metrics across baseline methods.

C. Baselines

We compare the performance of our model against a classical NLP baseline using sentence embeddings and cosine similarity. The classical model retrieves the most similar instruction from the dataset and returns its corresponding command. While it performs well on exact or near-exact matches, it struggles with paraphrased or previously unseen phrasings.

For model selection, we explored the broader landscape of open-source language models by referencing comparative evaluation data. As shown in (Fig.4) larger or more specialized models like Mixtral 8x7B achieve the highest task success rate of about 78% but come at the cost of significantly longer response times approximately 7 seconds. In contrast, LLaMA3 7B strikes a favorable balance, achieving a respectable success rate of 62% with a response time around 3 seconds. This trade-off is especially relevant for real-time vision-language tasks, where responsiveness is critical. Models like LLaMA2 7B, while faster, show noticeably lower success rates, suggesting that recent architectural and fine-tuning improvements in LLaMA3 contribute meaningfully to instruction understanding. Given our goal of building an interactive command translation system that operates in near real-time, LLaMA3 7B provides a practical compromise between speed and accuracy.

D. Failure Cases

We observe that LLaMA occasionally truncates output for instance missing action verbs or defaults to placeholder object names (e.g., `object` instead of `red_cube`). Additionally, if the instruction references unseen objects (e.g., "green mug"), the model responds gracefully by informing the user of valid alternatives which kind of demonstrates pragmatic reasoning.

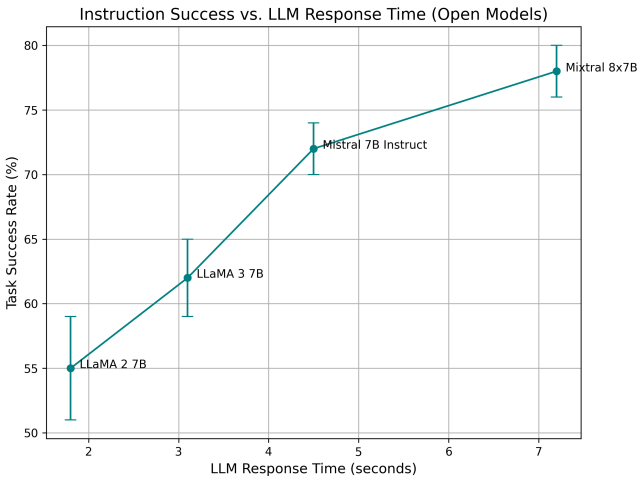

Fig. 5: Instruction success rate vs. LLM response time for various open-source models. LLaMA3 7B offers an optimal trade-off for our task.

VI. Conclusion

In this work, we have demonstrated that a vision-language-action pipeline can be recast as a traditional question-answering problem, grounding image understanding in textual context and translating natural language commands into executable robot routines. By fine-tuning LLaMA3.2-Vision on COCO 2017 for descriptive scene summarization and LLaMA3 for instruction-to-code translation, we achieved reliable performance on our own "can we do it?" benchmark: the model demonstrated the ability to generate valid command sequences for varied object-action pairs in partial simulation. This success underscores the viability of leveraging pre-trained multimodal LLMs to bridge perception and action without bespoke semantic parsers.

However, to caveat our results: our approach capitalizes on the strengths of existing, large pre-trained models—and it is therefore "standing on the shoulders of giants." To establish a truly level baseline, future comparisons should consider using a model family like Gemma2 for the Q-A component; Gemma2's smaller footprint would provide fairer apples-to-apples evaluations against 4B-parameter baselines before scaling up to larger architectures.

Additionally, we also recognize inherent limitations in our design. The extended episode length forced us to remove much of the intended reinforcement-learning loop, pivoting to a purely supervised fine-tuning and filtering regimen. While this choice streamlined development and aligned with the NLP focus of this course, it precluded exploration of adaptive, feedback-driven policy refinement. As such, full integration of RL remains an avenue for future work but lies outside the scope of this project's computational and educational constraints.

VII. Future Work

While our current implementation demonstrates the feasibility of grounding natural language instructions into executable robot commands, several avenues remain for future development and exploration.

- **Reinforcement Learning Integration:** Our original goal included training with reinforcement learning to allow the robot to improve its behavior through feedback. Due to hardware limitations, we were unable to implement this, but it remains a promising next step for enabling more adaptive, autonomous agents.
- **Expanded Dataset Usage:** Incorporating larger, task-rich datasets such as TEACh or ALFRED would allow the system to learn from a broader range of tasks and improve generalization to complex multi-step instructions.
- **Sim-to-Real Transfer:** A critical future direction is transferring the capabilities from simulation (MuJoCo or PyBullet) to real-world robotic platforms. This requires addressing challenges such as sensor noise, real-time latency, and perception errors.
- **Out-of-Vocabulary Command Handling:** A significant limitation of our current approach is its reliance on a fixed vocabulary of action tokens. Future work should explore techniques to handle out-of-vocabulary commands through:
  - **Semantic Action Expansion:** Developing methods to map novel verbs to known action primitives based on semantic similarity (e.g., "grasp" $\rightarrow$ "pick_up").
  - **Compositional Action Learning:** Creating a framework where new actions can be constructed as compositions of known primitives (e.g., "slide" as a combination of "push" with specific force parameters).
  - **Action Vocabulary Smoothing:** Implementing techniques similar to language model smoothing, where the system allocates probability mass to unseen action combinations based on partial matches or semantic proximity.
  - **Few-Shot Action Learning:** Enabling the system to learn new action primitives from minimal demonstrations, potentially through meta-learning approaches that accelerate adaptation.
- **Unanswerable Command Detection** Much like a Q-A model must contend with unanswerable questions, our modified VLA approach is susceptible to unanswerable commands.

A. Outright Unfeasible Commands

The current system lacks mechanisms to detect when commands are fundamentally unfeasible for the robot's capabilities. Future work should explore:

- **Command Feasibility Classification:** Training a dedicated classifier to determine whether instructions fall within the robot's physical capabilities before attempting execution.
- **Explainable Rejection:** Developing methods for the system to articulate why certain commands cannot be executed (e.g., "I cannot do a flip because my linkages do not allow for that motion").
- **Alternative Suggestion Generation:** When rejecting unfeasible commands, the system could propose similar feasible alternatives (e.g., "Instead of doing a flip, I can rotate in place").
- **Confidence Thresholding:** Implementing confidence scores for command interpretations, allowing the system to request clarification when uncertainty exceeds predefined thresholds.

B. Context-Incompatible Commands

Our system also needs better mechanisms to detect when commands are incompatible with the current environmental context. Future research should address:

- **Vision-Language Consistency Verification:** Developing explicit verification modules that check whether objects referenced in commands exist in the scene description.
- **Counter Reasoning:** Enabling the system to explain what would be required to make an unfeasible command feasible (e.g., "To pass you the green ball, I would first need a green ball to be present on the table").
- **Context-Aware Command Rewriting:** Automatically reformulating commands to align with available objects while preserving intent (e.g., rewriting "pick up the green cube" to "pick up the red cube" when only red cubes are available).confidence scores for command interpretations, allowing the system to request clarification when uncertainty exceeds predefined thresholds.

- **Benchmarking:** We aim to evaluate future iterations of the model against standardized benchmarks such as RT-2, VIMA, and Language Table to assess robustness and performance in diverse tasks and domains.
- **Model Compression and Optimization:** Given the computational overhead of large models like LLaMA3, future work could explore quantization or distillation techniques to reduce model size while maintaining accuracy, enabling faster inference on resource-constrained systems.

VIII. References

E. Coumans and Y. Bai, "PyBullet: Physics Simulation for Games, Robotics and Machine Learning," GitHub repository, 2019.

E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A Physics Engine for Model-Based Control," in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), 2012, pp. 5026–5033.

H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv preprint arXiv:2302.13971, 2023.

H. Chase, "LangChain: Build Applications with Large Language Models through Composability," GitHub repository, 2022.

K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky, "Π0: A Vision-Language-Action Flow Model for General Robot Control," arXiv preprint arXiv:2410.24164, 2024.

Ollama, "Ollama: Run Large Language Models Locally on Any Hardware," Ollama, 2024.

PyBullet. (n.d.). PyBullet: Real-time physics simulation for robotics. Retrieved May 4, 2025, from https://pybullet.org/wordpress/

NVIDIA, "NVIDIA Omniverse Isaac Sim Documentation," NVIDIA Developer, 2023.

Stanford Autonomous Systems Laboratory, "Pi-Zero: An Open-Source Robotics AI Framework," GitHub repository (MIT License), 2024.

Z. Zhou, J. Song, X. Xie, Z. Shu, L. Ma, D. Liu, J. Yin, and S. See, "Towards Building Artificial Intelligence–Cyber Physical Systems (AI-CPS) with NVIDIA Isaac Sim: An Industrial Benchmark and Case Study for Robotics Manipulation," arXiv preprint arXiv:2308.00055, 2023.