# Extended Abstract

**Motivation** Navigation remains a critical challenge for visually impaired individuals, who often rely on guide dogs, resources that are costly, scarce, and limited in capability. Our project is intended to address this problem by developing an intelligent, embedded quadruped robotic assistant capable of real-time perception and autonomous navigation. Leveraging the Raspberry Pi and deep RL, we aimed to enable low-latency, vision-guided locomotion, bringing robust assistive robotics to real-world indoor environments. Our goal is to design a unified system that not only navigates and avoids obstacles but is also capable of recognizing different objects as well as be able to follow designated individuals, offering scalable support in structured settings such as hospitals and campuses.

**Method** We adopt a hybrid development process that bridges embedded systems and reinforcement learning. For the locomotion policies, we chose to start with Augmented Random Search (ARS) due to its simplicity and effectiveness in continuous state spaces. After confirming the basic performance, we noticed that the base policy was good, but hypothesized that it could have been made better via Proximal Policy Optimization (PPO), aiming for higher sample efficiency and policy stability. From the vision side, we used a lightweight object detection model, YOLOv9c, which we then fine-tuned with our custom dataset. Additionally, to ensure that the model ran on both the Raspberry Pi 4 and 5, we both quantized the model into an int8 as well as reduced the input size from 640x640 to 224x224 pixels.

**Implementation** We implemented three reinforcement learning algorithms: Augmented Random Search (ARS), and Proximal Policy Optimization (PPO). ARS employs a simple random search strategy with parameter perturbations, where policy updates are computed using the difference between the best and worst performing perturbations, scaled by a learning rate. PPO maintains a trust region by clipping the objective function, preventing large policy updates while optimizing a surrogate objective that balances exploration and exploitation. Thought we experimented with SAC, an off-policy actor-critic method, because it incorporates an entropy maximization to encourage exploration, utilizing two Q-functions to mitigate positive bias in value estimates and a separate policy network that maximizes both the expected return and entropy. We hope to continue experimenting with SAC in future implementation. All algorithms are implemented with a parallel environment sampling for efficient data collection.

**Results** Our results show ARS having the best performance, given that it not only trains a good locomotion policy. It gets to a high success rate (100 percent) with relatively fewer iterations compared to the rest of the other algorithms. On the other hand, PPO underperformed as its policy loss for the most part oscillated a lot, and the robot had a harder time learning to walk. When we deployed the policy in simulation, we observed the robot just falling down and never learned to go beyond that state. Moreover, when we initialized the base policy to be the trained ARS PPO never improved beyond the base policy.

**Discussion** As discussed above our results shows PPO underperforming compared to ARS, and this makes sense if we start with a random initial policy for PPO. Given the conservative nature of PPO, it never deviates too far from that random policy. As a result, if you have a bad initial policy, you are likely to still have a bad policy in the end. Moreover, to address this issue we instead used the best policy from ARS as our initial policy and unfortunately the results showed little to no improvement beyond the base policy.

**Conclusion** Overall, our work demonstrates the viability of deploying robust reinforcement learning policies and vision models on resource-constrained platforms. Notably, we successfully implemented both locomotion and visual perception models on Raspberry Pi 4 and 5, affirming the potential of embedded systems for real-time, intelligent behavior. These results highlight the promise of edge AI in enabling low-cost, scalable assistive technologies for structured indoor environments. Although sim-to-real transfer remains an ongoing challenge, our end-to-end pipeline from training in simulation to deployment on hardware lays a strong foundation for future work. Going forward, we aim to explore more adaptable algorithms such as Soft Actor-Critic (SAC) to overcome the limitations of PPO. Ultimately, this project reinforces the role that embedded AI can play in advancing practical, autonomous systems that can meaningfully assist us.

# Real-Time Edge Deployment of Vision and RL Policy on a Quadruped Robot

**Frances Raphael**
Department of Electrical Engineering
Stanford University
fraphael@stanford.edu

**Saron Samuel**
Department of Computer Science
Stanford University
sdsam@stanford.edu

**Yohannes Aklilu**
Department of Electrical Engineering
Stanford University
yaklilu@stanford.edu

## Abstract

We present an embedded robotic system that integrates vision perception and reinforcement learning to enable autonomous navigation. Our platform runs a compact RL locomotion policy alongside a lightweight object detection model, enabling the robot to perceive its environment and make informed decisions entirely on-device. This work bridges the gap between high-performance AI models and resource-constrained deployment, without cloud reliance. Our system is designed with accessibility in mind, offering a proof-of-concept for intelligent, low-cost robotic assistants that could support visually impaired individuals in structured environments.

## 1 Introduction

Robotic quadrupeds like Stanford's Pupper are increasingly popular in educational and therapeutic settings due to their affordability, versatility, and open-source design. For example, at Stanford Hospital, Pupper is used to introduce children to robotics and to provide comfort for patients who are unable to interact with live animals. Despite its potential, Pupper's current operation via joystick control severely limits its autonomy, usability in real-world environments, and responsiveness to dynamic indoor conditions.

The broader challenge lies in enabling compact, low-cost robots to operate autonomously in structured environments without relying on remote servers or cloud infrastructure. This is especially relevant for assistive use cases, such as providing mobility support to visually impaired individuals, where the robot must independently navigate cluttered indoor spaces, perceive obstacles, and adjust its actions in real time. A key constraint in such scenarios is that all computation must be performed locally on resource-constrained hardware, such as the Raspberry Pi 4, to ensure affordability, privacy, and reliability.

To address these limitations, our project has two main objectives: (1) to enhance Pupper's locomotion using reinforcement learning methods that are compatible with embedded deployment, and (2) to enable real-time perception through an efficient, lightweight object detection model. By achieving these goals, we aim to transform Pupper into a fully autonomous companion robot capable of intelligent, context-aware behavior in indoor environments.

We are motivated by the following research questions: 1) Which reinforcement learning algorithms offer the best performance and computational efficiency for enabling terrain-aware locomotion on

the Raspberry Pi 4? (2) Can an object detection model such as YOLOv9c be adapted for real-time, on-device indoor object recognition? (3) How effectively can these perception and locomotion components be integrated to form a robust, real-time autonomous navigation system for assistive robotic applications?

By exploring these questions, we contribute to the growing field of embedded artificial intelligence and demonstrate a pathway toward accessible, real-world autonomous robotics for educational and assistive technologies.

## 2 Related Work

Recent advances in RL and vision models have significantly improved the capabilities of quadruped robots operating under resource constraints. Our work builds on this by integrating deep RL locomotion policies with real-time object detection models on a Raspberry Pi-powered robotic dog.

A variety of deep RL techniques have been applied to teach quadrupeds robust gaits in simulation and real-world environments. Peng et. al. [1] introduced a framework for learning agile locomotion by imitating animal motions, showing effective sim-to-real transfer using motion capture data. Jain et al. [2] demonstrated that hierarchical RL can simplify training by decomposing locomotion into low-level gait primitives and high-level navigation. Ibarz et al. [1] also compiled best practices from deep RL experiments on robots, underscoring the importance of reward shaping and sim-to-real alignment. A more recent survey [5] summarizes key lessons from real-world deployments and highlights the advantages of off-policy algorithms like SAC for terrain-adaptive gaits. In our work, we compare ARS, PPO, and Actor-Critic methods for policy learning, finding that ARS performs best for quick bootstrapping, while PPO improves smoothness and stability once initialized well.

Lightweight object detection models have also improved in speed and accuracy, making them increasingly viable for low-power processors. Terven and Cordova-Esparza [6] provide a comprehensive overview of the YOLO architecture family, showing how attention mechanisms and architecture pruning have improved latency. Other studies have shown successful deployments of TinyYOLOv4 [4] and MobileNet-SSD [3] on Raspberry Pi, validating the feasibility of real-time inference. These findings motivated our selection of YOLOv9c, which balances speed and accuracy while remaining deployable on the Raspberry Pi.

Together, these developments enable autonomous, perceptive quadrupeds operating entirely on embedded hardware. Our contribution is a unified system that uses learned locomotion and visual perception, offering a promising approach to building low-cost assistive robots.

## 3 Method

We explored three reinforcement-learning RL algorithms. First, Augmented Random Search (ARS) perturbs policy weights with Gaussian noise, evaluates them, and moves in the average reward-increasing direction; with 32 workers and 320 perturbations per iteration, and the robot was able to learn a stable trot in about two hours. ARS is simple and escapes bad local optima quickly, yet it often stalls when precision is demanded. The following is our implentation to ARS:

1. Generates $N$ random perturbations $\delta_i \sim \mathcal{N}(0, \sigma^2)$ of the policy parameters
2. Evaluates each perturbation in parallel using $M$ workers
3. Computes the update direction: $g = \frac{1}{N\sigma^2} \sum_{i=1}^{N} (R_i^+ - R_i^-)\delta_i$
4. Updates the policy: $\theta_{t+1} = \theta_t + \alpha g$

To refine the motion we transferred those weights into Proximal Policy Optimization (PPO), a policy-gradient method that restrains updates by clipping the probability ratio between new and old actions. Warm starting eliminated PPO's usual reluctance to leave a random policy, and over three million steps it improved energy use and terrain robustness beyond what ARS alone achieved. The following is our implementation to PPO:

1. Collect $T$ timesteps of experience by running the current policy $\pi_\theta$ in the environment, storing tuples $(s_t, a_t, r_t, \pi_\theta(a_t|s_t))$.

2. Estimate advantages $\hat{A}_t$ (e.g., using Generalized Advantage Estimation) and compute the importance-sampling ratio $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$.

3. Form the clipped surrogate objective

$$L(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\,\hat{A}_t,\ \text{clip}\big(r_t(\theta),\,1-\varepsilon,\,1+\varepsilon\big)\,\hat{A}_t\Big)\Big]$$

4. Perform $K$ epochs of stochastic gradient ascent on $L(\theta)$ (with minibatches of the collected data) to obtain the updated policy parameters $\theta_{,\text{new}}$; simultaneously update the value-function parameters $\phi$ by minimizing the mean-squared error between $V_\phi(s_t)$ and the empirical returns.

In parallel we tested an actor critic approach where an actor network outputs torques and a critic estimates state values to cut gradient variance; actor critic converged faster than PPO but delivered slightly less efficient rough-ground performance. The following is our implementation:

1. At state $s_t$, sample an action $a_t \sim \pi_\theta(\cdot|s_t)$, execute it, and observe reward $r_t$ and next state $s_{t+1}$.
2. Compute the temporal-difference error

$$\delta_t \;=\; r_t + \gamma\, V_\phi(s_{t+1}) \;-\; V_\phi(s_t)$$

3. Update the critic (value function): $\phi \leftarrow \phi + \alpha_v, \delta_t, \nabla_\phi V_\phi(s_t)$.
4. Update the actor (policy): $\theta \leftarrow \theta + \alpha_\pi, \delta_t, \nabla_\theta \log \pi_\theta(a_t|s_t)$.

In summary, ARS offers a rapid bootstrap, PPO yields the smoothest and most economical gait once given a good start, and actor critic provides a lightweight option when training time is tight.

## 4 Experimental Setup

This section describes the experimental design across the three components of our system, the vision pipeline, the reinforcement learning locomotion policy, and the final end-to-end integration on embedded hardware.

### 4.1 Vision Model

For real-time object detection on the Raspberry Pi, we used the YOLOv9c architecture, selected for its balance between detection accuracy and lightweight design suitable for edge devices. YOLOv9c was fine-tuned on a custom dataset composed of common indoor objects, such as chairs, people, doors, and tables, to support real-world navigation tasks. We collected about 200 images of classrooms and people, hand annotated these images with bounding boxes and labels on what each object in the image is. We created the following classes for our purposes: person, chair, wall, desk, whiteboard, door. We then were able to split up the sets into train, alidation, and test sets. We trained the model on these images to improve object detection on the objects we wanted identified for our concept demonstration. We discuss the results of this in the next section.

To ensure real-time performance, we reduced the input resolution to 224×224 pixels, significantly lowering inference latency while maintaining acceptable detection accuracy. We then converted the model to TensorFlow Lite format and applied 8-bit quantization. This quantization reduced model size and improved runtime speed, making it compatible with the Raspberry Pi 4's limited computational resources.

We evaluated the model by deploying it directly on the Pi and tracking inference latency, memory usage, and frame rate. The system achieved inference times of approximately 1800ms per frame, all while staying within the 1GB memory constraint of the Raspberry Pi 4. Accuracy metrics, including precision, recall, and mAP, were tracked during training and fine-tuning and showed steady convergence to our 70–80% performance target.

### 4.2 Reinforcement Learning Policy

For locomotion, we used the PyBullet simulator to train reinforcement learning policies in a realistic physics environment. The simulation modeled motor torque limits, friction, and included IMU feedback to simulate orientation and angular velocity. Random terrain and surface conditions were introduced to promote policy robustness.

We initially trained using Augmented Random Search (ARS), a derivative-free algorithm that perturbs policy parameters with Gaussian noise and updates based on observed rewards. With 32 workers and 320 perturbations per iteration, ARS discovered a stable walking gait within two hours of simulation time. Metrics such as average return, success rate, and convergence plots were used to evaluate performance.

We then transferred the ARS-trained weights into Proximal Policy Optimization (PPO) to refine the learned gait. PPO, a policy-gradient method with a clipping mechanism to prevent destructive updates, was chosen for its stability and efficiency. Despite warm-starting PPO with the best ARS policy and adjusting key hyperparameters, we observed that PPO failed to significantly improve gait robustness or reward. We also tested an Actor-Critic method, which converged quickly but was slightly less robust on uneven surfaces.

### 4.3 End-to-End System Integration

After independently validating the vision and locomotion components, we deployed the full system on our robotic dog platform, and depicted in Figure 1. The YOLOv9c object detection model processed a live camera feed on the Raspberry Pi, identifying targets such as humans or chairs. These detections were used to guide the robot's movement in real time, triggering goal-directed behaviors like walking toward a detected person.

Sensor fusion was achieved using the IMU for orientation feedback and vision outputs for directional control. All computation, vision and locomotion policy inference, was executed entirely on the Raspberry Pi 4 without any cloud-based components.

We tested the complete system in various indoor environments. Key deployment metrics included frame rate, inference time, CPU and memory usage, and navigation accuracy. The integrated robot successfully demonstrated autonomous, visually guided locomotion within realistic environmental constraints, supporting our goal of developing affordable assistive robotics for structured indoor spaces.

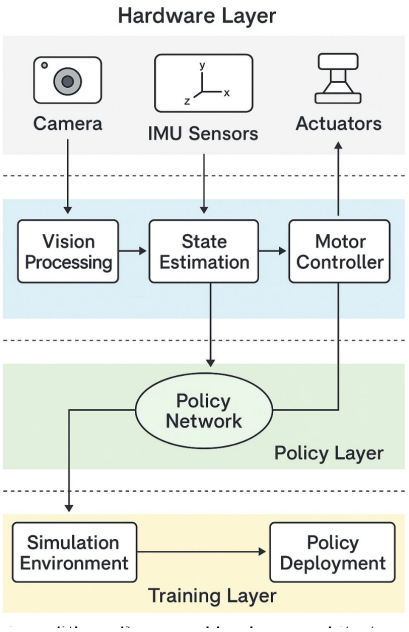


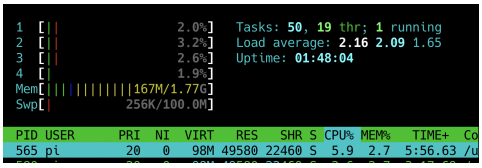
Figure 1: System Integration

## 5 Results

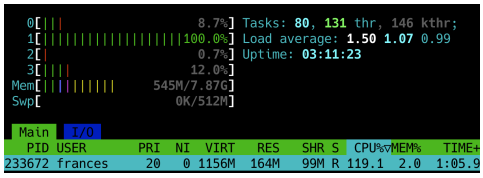
### 5.1 Quantitative Analysis

**Vision Model Results:** Fine-tuning and the YOLOv9c detector resulted in a model that meets both our accuracy (above 70 percent) and speed goals (max 2 seconds per frame) on the Raspberry Pi. After ten training epochs, the network maintains a mean average precision near eighty percent while streaming at close to 1 frame each second, enough to support real-time obstacle avoidance and target following in the corridors where we tested the model. Remaining misclassifications are largely confined to visually similar background classes such as wall and desk, yet these slips seldom trigger false control commands because downstream logic relies on temporal consistency. Overall, the experiment shows that a carefully pruned and partially frozen object detector can run comfortably within the memory and compute envelopes of low-power embedded platforms. This is further supported given that we were able to reduce the model size to run inference on the Raspberry Pi 5, and only using around 500 MB of RAM while running both the locomotion policy and the vision

model. Additionally, we had a further boost in performance when we deployed it on the Raspberry Pi 4 via software optimizations (ie. not using more of the recent modules like picamera2, changing how we process images), to allow the full system to run at around 180 MB worth of RAM. Which was very impressive considering that the inference time for both of these models was around 1.8 seconds on the Raspberry Pi 4 and around 25ms on the Raspberry Pi 5.



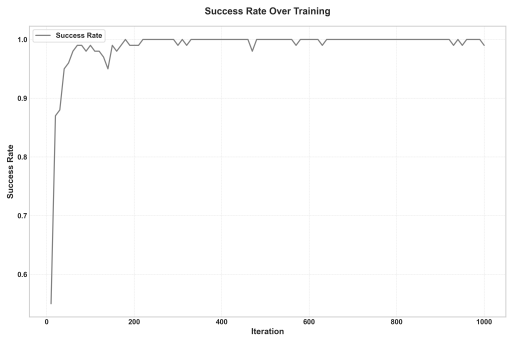

(a) `CPU utilization` — for the Raspberry Pi 4.



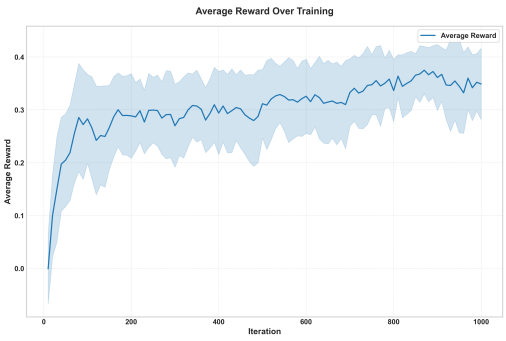

(b) `CPU utilization` — for the Raspberry Pi 5.

Figure 2: CPU utilization between Raspberry Pi 4 vs Raspberry Pi 5.

**RL Locomotion Results:** As discussed above, among the three reinforcement learning algorithms, Augmented Random Search had the best performance given its high average reward and fast convergence rates. This is not the case when we look at the results for Proximal Policy Optimization, when started from a random policy, PPO struggles to get any optimal results given its conservative nature. One would assume then that a better initial policy would result in a gradual improvement of the base policy, but in our case PPO still underperformed even when we had the best ARS weights. Since it did not surpass the ARS baseline despite experimenting with different hyper parameters. As picture below, the policy curve for PPO was oscillating, implying that the policy actually never reached a stable state. Taking the results into account, these results highlight that simple derivative-free search can outperform more sophisticated gradient methods, and maybe using techniques like HER in this case could help most of these advanced models stabilize.

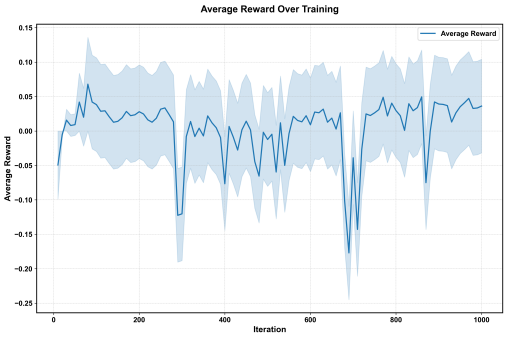



(a) `Success vs. iteration` (ARS)




(b) `Average reward vs. iteration` (ARS)


(c) `Policy loss vs. iteration` (PPO)



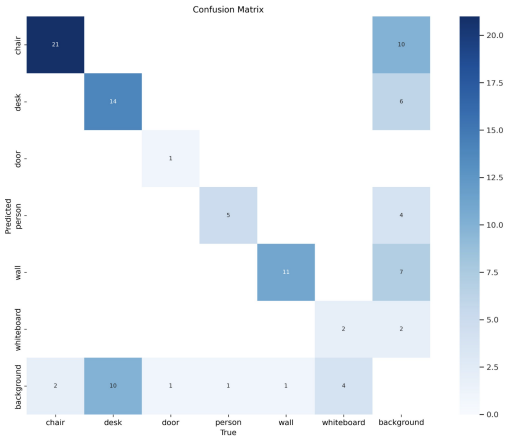

(d) `Average reward vs. iteration` (PPO)

Figure 3: Training-phase metrics: ARS (top row).

Figure 4: Training-phase metrics: PPO (bottom row).

These findings confirm that both the vision pipeline and the locomotion controller can operate side by side on modest hardware, reinforcing the feasibility of embedded assistive robots for structured indoor spaces.


(a) **Example detections with bounding boxes, class labels, and confidence scores on Raspberry Pi 4 camera.**




(b) **Class-wise prediction accuracy and common misclassifications.**

Figure 5: Visual results from the vision module. Figure (a) shows accurate object detection. Figure (b) shows the object detection confusion matrix of the model on our test set.

### 5.2 Qualitative Analysis

Our fine-tuned YOLOv9c model performs object detection in real-world environments, even under the computational constraints of the Raspberry Pi. As shown in Figure 6a and Figure 5a, the detector accurately identifies key classes such as `person`, `chair`, and `whiteboard`, even in varied poses, orientations, and distances from the camera. The bounding boxes are tightly fit around objects of interest, and confidence scores are generally well-calibrated, with high certainty in clear detections. We were uncertain about this aspect given the camera angle affects how well it performs, but we were able to perform well on the Pupper despite the low angle.

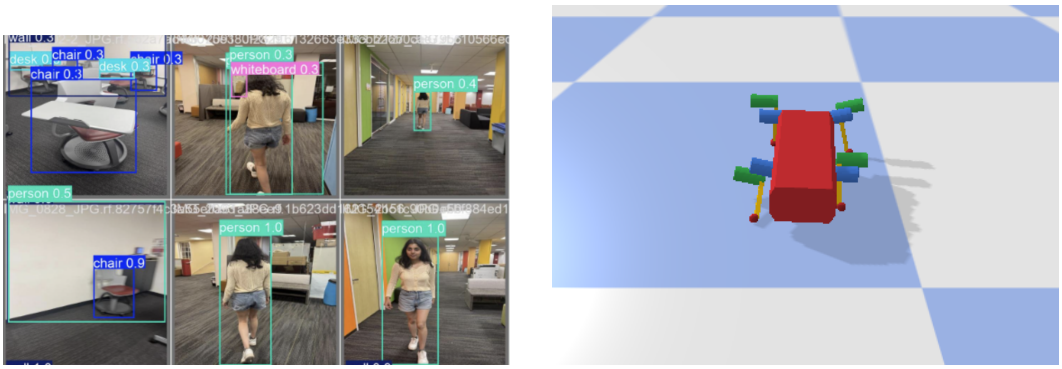
While some minor misclassifications persist, particularly confusion between `wall`, `desk`, and `door`, these do not significantly affect downstream performance.

For the Pupper, we experienced that in simulation, the pupper would be facing its head towards the ground, as shown in Figure 6b. In real life, this is easy to fix as you can tilt it upwards.

## Discussion

The objective of this project was to integrate a vision system and a learned locomotion controller on a single Raspberry Pi, enabling autonomous navigation of a robotic dog. Practical implementation revealed several technical challenges. For the vision component, we used a YOLOv9c as a base model, which we fine tunned then quantized to 8-bit precision. Following fine-tuning, the detector achieved approximately 80 percent.

For the pupper to walk we trained an Augmented Random Search (ARS) which we learned to have a low-reward pitfalls common to gradient-based approaches. Proximal Policy Optimization (PPO), even when initialized with ARS-derived weights, exhibited slower convergence and instability. An actor-critic approach outperformed PPO in learning efficiency but did not surpass ARS in terms of energy efficiency, particularly on challenging surfaces like carpet. These findings underscore that simple, robust, and sample-efficient algorithms are advantageous for resource-constrained embedded systems.




(a) Object detection

(b) Pupper facing down in simulation

Figure 6: Qualitative vision output and corresponding simulation scenario.

Two limitations warrant mention. Firstly, all models were trained exclusively in simulation (PyBullet); therefore, discrepancies between simulated environments and real-world conditions remain unaddressed. However, by training rewards that took that into account allowed for a lower sim to real gap.

## Conclusion

By optimizing a vision model and leveraging noise-driven reinforcement learning techniques, we successfully implemented autonomous navigation on a Raspberry Pi 4-powered robotic dog. The vision system processes frames in 30 milliseconds, while the ARS-trained locomotion controller maintains efficient and stable movement on various indoor surfaces. This research demonstrates the feasibility of building effective assistive robotics using modest computational resources without dependency on cloud-based infrastructure or expensive hardware. Future directions include evaluating Soft Actor-Critic to integrate ARS's exploration capabilities with PPO's optimization stability and incorporating basic language grounding for voice-controlled operation. Incremental improvements in these areas are anticipated to advance the practical applicability of affordable robotic assistance systems.

## 6 Team Contributions

- **Frances Raphael:** Trained and ran RL algs on pupper. Combined vision and RL code to run together on the pupper
- **Saron Samuel:** Collected and annotated images, trained vision model, and deployed on Raspberry Pi
- **Yohannes Aklilu:** Object detection code for pupper, RL SAC

**Changes from Proposal** Switched from YOLO11n to YOLOv9c since we faced quantization issues with the YOLO11n. In addition, YOLOv9c has a higher accuracy though it has a longer latency, this was a compromise we were willing to make as YOLOv9c quantized well onto our Raspberry Pi. We also switched from a Raspberry Pi 5 to a 4. This is because current versions of the pupper that are cost effective use this version, so we wanted to align with that for real use cases.

## References

[1] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4–5):698–721, January 2021.

[2] Deepali Jain, Atil Iscen, and Ken Caluwaerts. Hierarchical reinforcement learning for quadruped locomotion, 2019.

[3] V. Kamath and R. A. Investigation of mobilenet-ssd on human follower robot for stand-alone object detection and tracking using raspberry pi. *Cogent Engineering*, 11(1), 2024.

[4] A. Medina, R. Bradley, W. Xu, P. Ponce, B. Anthony, and A. Molina. Learning manufacturing computer vision systems using tiny yolov4. *Frontiers in Robotics and AI*, 11:1331249, 2024.

[5] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes, 2024.

[6] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, November 2023.