The One RING : a Robotic Indoor Navigation Generalist

Supplementary Material

Appendices for *The One RING : a Robotic Indoor Navigation Generalist*

The following items are provided in the Appendix:

- *Details about the real-world evaluation robot platforms and human evaluation setup (App. 6),*
- *Data collection for randomized embodiments using expert planners in simulation (App. 7),*
- *Full experimental setup (App. 8),*
- *Model architecture details (App. 9),*
- *A visualization of the random embodiments in our training set, along with the 5 nearest neighbors to each of the real robots (Stretch-RE1, LoCoBot, Unitree A1) (App. 10), and*
- *Limitations (conditioning on explicit embodiment parameters(App. 11).*

In our supplementary materials, we also include a supplementary website (see the index.html file) that contains

- *Real-world qualitative videos of evaluating* RING *zero-shot on four different robot platforms, including Stretch RE-1 with our camera setup, Stretch RE-1 with factory camera configuration, LoCoBot, and Unitree GO1,*
- *Qualitative videos for human evaluation, using* RING *as navigation assistant,*
- *Videos showing our dataset of trajectories collected from random embodiments in simulation.*

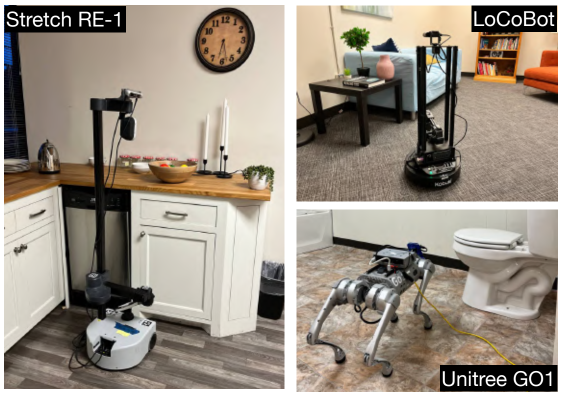


Figure 8. **Robot platforms**. We use 3 different platforms, including Stretch RE-1, LoCoBot, and Unitree GO1 for our real-world evaluations.

6. Real Robot Platforms and Human Evaluation Setup.

6.1. Stretch RE-1, LoCoBot, Unitree GO1

We use Stretch RE-1, LoCoBot, and Unitree GO1 as our robot platforms for real-world evaluations, shown in Fig. 8. For Stretch RE-1, we evaluate two different sensor configurations: the factory configuration and the configuration suggested by SPOC [17]. We summarize the main differences in these platform in Tab. 7. For robot movements, we either implement a Kalman filter or wrap around provided robot APIs to realize low-level controllers for a discrete action space {`MoveBase(±20cm)`, `RotateBase(±6°, ±30°)`, `Done`} across all platforms. It is important to note that during the training stage, we do not use any embodiment configurations from these robots to generate imitation learning data or to initialize RL fine-tuning embodiments.

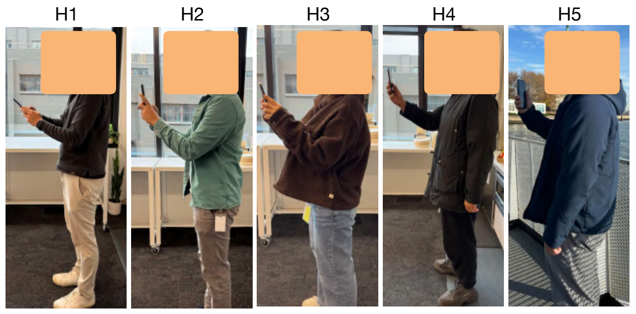


Figure 9. **Human Participants for Navigation Assitance**. Our five human participants have different height and different camera-holding poses, resulting in different sensory observations (details about human participants in Table 8).

6.2. Human Evaluations

Human participants. We asked five human participants to use RING as a navigation assistant and evaluated its performance across a range of different human embodiments. These variations stem from differences in camera-holding poses, participant heights, step sizes, and rotation angles. A summary of these variations across different participants can be found in Tab.8 and Fig.9. Due to these variations, each participant contributes a unique set of evaluation embodiments and sensor configurations.

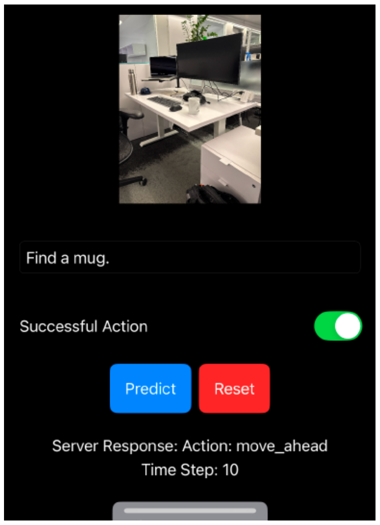
Human evaluation details. We developed a simple iOS app, as shown in Fig. 10, that enables human participants to input the target object as text (e.g., Find a mug), capture an image using the iPhone's back camera, and send

	Stretch RE-1	Stretch RE-1 (Factory)	LoCoBot	Unitree GO1
Body dimension (cm)	33 × 34 × 141	33 × 34 × 141	35×35×89	64.5×28×40
Camera model	2× D455	D435	D435	D435
Camera vertical FoV (degrees)	59°	69°	42°	42°
Camera height (cm)	140	130	87	28
Camera pitch (degrees)	27°	30°	0°	0°

Table 7. **Details about evaluation robot platforms.** Our four robot platforms have varying dimensions and camera configurations, resulting in diverse evaluation embodiments.

	H1	H2	H3	H4	H5
Height	6′3″	5′10″	5′5″	6′1″	5′11″
Step size	0.25m	0.35m	0.4m	0.3m	0.3m
Rotation Degrees	30°	45°	45°	35°	30°

Table 8. **Details about human evaluators.** Our five human participants have varying heights, step size, and rotation degrees, resulting in different evaluation embodiments.


Find a mug.

Server Response: Action: move_ahead
Time Step: 10

Figure 10. **iOS app for human evaluation**. We developed a simple iOS app that enables human participants to text goal, capture an image using iPhone's back camera, send both to a remote server, and receive the predicted action from our RING policy.

both the text prompt and the captured image to a remote server. Upon receiving the prompt and image, the remote server processes them using our RING policy to predict action probabilities and samples an action. The predicted action is then sent back to the iPhone and displayed within the app. The prompted action space used is identical to that of our real-world robot.

The human participant navigates by following the suggested action at their own pace and rotation degree, as detailed in Tab. 8. After completing each action, they tap the `Predict` button again, prompting the app to capture a new

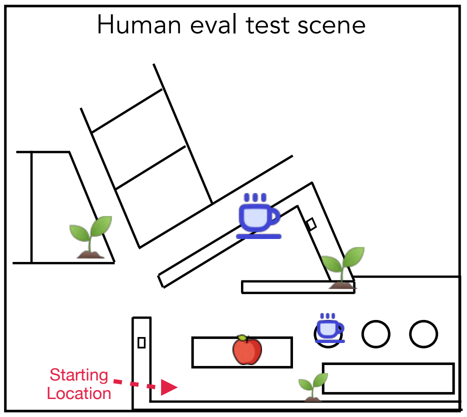


Figure 11. **Real-world test scene for human evaluation**. Three different target objects include (🍎 `Apple`, 🌱 `Houseplant`, ☕ `Mug`). The red arrow in the bottom left corner is the starting location.

image and send it to the server along with the text prompt. This process repeats until the `Done` action is returned or the 100th step is reached. An episode is considered successful if, before reaching 100 steps, the target object is visible in the latest image and within a 1 meter distance when RING calls `Done`.

The layout of the test scene is shown in Fig. 11, illustrating two locations for finding a `Mug`, three for a `Houseplant`, and one for an `Apple`. The participant always begins at the bottom-left corner of the scene.

7. Data Generation with Expert Planners

Expert planners introduced by [17] are not efficient and robust for random embodiments. As a result, we made major improvements to the planners to allow for better trajectories.

The major factor in this improvement is to consider *safety* of the policy (defined as the avoidance of approaching any obstacles along the way.) We use A* [20, 22] to generate safe navigation trajectories for training as follows: **1)** Extract reachable locations in a scene on a finely spaced grid, ensuring that the agent's collider does not intersect with any object's collider. Thus, different embodiments yield different reachable locations according to their collider. **2)** Compute a clipped Euclidean distance to the nearest obstacle. Then, for each location, set the cost of visiting it as the inverse of the third power of the distance. **3)** Construct a grid-like graph where each reachable location is a node connected to its immediate neighbors. For each connection, assign a cost equal to the maximum cost of visiting either of the two connected nodes. **4)** Extract a minimum-cost path connecting the reachable positions in the graph nearest to the source and to the target via A*. **5)** Extract waypoints by skipping over points in the A* path as long as skipping them doesn't increase the total path cost from the latest waypoint. **6)** The expert linearly interpolates between waypoints up to the precision reachable by the action space to generate each trajectory.

8. Additional Benchmark/Experiment Details

Action Space. Following on prior work with AI2-THOR, we discretize the action space for all agents in our training: {`MoveAhead`, `MoveBack`, `RotateRight`, `RotateLeft`, `RotateRightSmall`, `RotateLeftSmall`, `Done`}. Here, `MoveAhead` advances the robot by 0.2 meters, `MoveBack` moves the robot backward by 0.2 meters, `RotateRight`/`RotateRightSmall` rotates it clockwise by 30° / 6° around the yaw axis, and `RotateLeft`/`RotateLeftSmall` rotates it counterclockwise by 30° / 6° around the yaw axis, and `Done` indicates the agent has located the target, ending the episode. We evaluate RING zero-shot on all robots (Stretch-RE, LoCoBot, Unitree Go1) with the same action space using their low-level controllers. When finetuning for embodiment-specialized policies, we finetune for a slightly different action space for LoCoBot: {`MoveAhead`, `MoveBack`, `RotateRight`, `RotateLeft`, `LookUp`, `LookDown`, `Done`}. `LookUp` tilts the camera up by 30° around the roll axis and `LookDown` tilts the camera down by 30° around the roll axis. All baselines are trained and evaluated with the same action space for fair comparison.

Success Criteria. We follow the definition of Object Goal Navigation from [2], where an agent must explore its environment to locate and navigate to a specified object within a maximum of n steps. To indicate it has found the target, the agent must execute the `Done` action. Success is determined by the environment based on whether the agent is within a distance d of the target and if the target is visible in its view. If the agent exceeds n steps without executing the `Done` action, the episode is considered a failure. For simulation benchmarks, we follow CHORES-S [17] with $n = 600$ and $d = 2$. For real-world evaluations, we use $n = 300$ and $d = 1$.

Success weighted by collision (SC). Collision is one of the main challenges for a unified policy operating across diverse embodiments in visual navigation tasks. Previous works measure the collision rate ($\frac{\#collisions}{\#steps}$) to understand how often a policy collides with objects in a scene. However, this does not reflect the effectiveness of the policy at the task level. For example, in a successful episode, a single collision and multiple collisions should have different impacts on the performance measurement. As a results, inspired from Success Weighted by Episode Length (SEL), we propose Success Weighted by Collision (SC),

$$SC = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{1}{1 + c_i}, \tag{2}$$

where S_i is a binary indicator of success for episode i, c_i is the number of collisions in episode i, and N is the number of evaluation episodes. In this metric, the policy is penalized most heavily for a single collision, with the penalization decreasing for each additional collision, as the penalty diminishes inversely with the number of collisions. Intuitively, > 0 collisions are much worse than 0, as a real robot may suffer damage from one bad collision, but the difference between 10 and 11 collisions is a more marginal difference.

Hyparameters. We list the hyperparameters used in training and the architecture in Table 9.

Imitation Learning	
Batch Size	224
Context Length	100
Learning Rate	0.0002
RL Finetuning	
Total Rollouts	64
Learning Rate	0.0002
Mini Batch per Update	1
Update Repeats	4
Max Gradient Norm	0.5
Discount Value Factor γ	0.99
GAE λ	0.95
PPO Surrogate Objective Clipping	0.1
Value Loss Weight	0.5
Entropy Loss Weight	0.0
Steps for PPO Update	128
Model Architecture	
Transformer State Encoder Layers	3
Transformer State Encoder Hidden Dims	512
Transformer State Encoder Heads	8
Causal Transformer Deocder Layers	3
Causal Transformer Deocder Hidden Dims	512
Causal Transformer Deocder Heads	8

Table 9. **Hyperparameters for training and model architecture.**

8.1. Real-World Benchmarks

All robots are evaluated in a multi-room apartment shown in Fig. 4. Based on the embodiment, the benchmark has different starting locations and objects. Among our target object categories, `Apple` can be found in the Living room and Kitchen, `Bed` can only be found in the Bedroom, `Sofa` and `Television` can only be found in the Living room, `Vase` can be found in the Livingroom, Corridor, Office, and Kitchen, `Chair` can be found in the Office and Kitchen, `HousePlant` can be found in the Living room, Office, and Kitchen.

- **LoCoBot**: Following Phone2Proc [10], use the same five target object categories, including `Apple`, `Bed`, `Sofa`, `Television`, and `Vase`, and the three starting poses shown in 4.
- **Stretch RE-1**: We follow SPOC [16] to use the same six target object categories, including `Apple`, `Bed`, `Chair`, `HousePlant`, `Sofa`, and `Vase`, and the three starting poses, shown in Fig. 4. We consider 2 different camera configurations for Stretch: 1) off-the-shelf camera equipped on the Stretch RE-1 (D435 with a vertical field of view of 69° and resolution of 720×1280), 2) following [17], we use 2 Intel RealSense 455 fixed cameras, with a vertical field of view of 59° and resolution of 1280×720. The cameras are mounted facing forward but pointing downward, with the horizon at an angle of 27°.
- **Unitree Go1**: We create a new evaluation set for Unitree Go1 with 3 starting poses (Fig. 4) and 4 objects (`toilet, sofa, TV, trashcan`) positioned to accommodate the robot's lower height, ensuring that the objects can be visible from its lower viewpoint.

9. Model Architecture Details

We will now detail RING's architecture (see Fig. 12), which is inspired by previous works POLIFORMER [62] and FLARE [24].

Visual encoder. We use the Vision Transformer from the pretrained SIGLIP-VIT-B/16 as our visual encoder. Since the RGB images vary in dimensions across different embodiments, we include an additional preprocessing step before feeding them into the encoder. Specifically, we pad each RGB image to a square and then resize it to 256×256. In addition, we mask the image from the 2^{nd} camera with zeros for the embodiments with only one camera. The visual backbone takes the RGB observation $i \in \mathbb{R}^{256\times256\times3}$ as input and produces a patch-wise representation $r \in \mathbb{R}^{\frac{256}{16}\times\frac{256}{16}\times h}$, where $h = 768$ is the hidden dimension of the visual representation. We reshape the visual representation into a $\ell \times h$ matrix, $\ell = 256 \cdot 256/16 \cdot 16$, and project the representation to produce $v \in \mathbb{R}^{\ell\times d}$, where $d = 512$ is the input dimension to the transformer state encoder. Note that since we have two RGB images from two cameras, we produce two visual representations $v^{1,2}$ at the end of this module. The vision encoder remains frozen through training.

		Nearest Neighbors				
	Stretch RE-1	N1	N2	N3	N4	N5
Camera Position (x) (meters	0	-0.06	0.11	0	-0.08	0.03
Camera Position (y) (meters	1.44	1.13	0.67	0.24	0.72	0.32
Camera Position (z) (meters	0.07	0.03	0.06	0.07	0.07	-0.03
Camera Pitch (degrees)	27	29	33	34	32	33
Camera Yaw (degrees)	0	0	0	0	0	0
Vertical FoV (degrees)	59	57	56	54	59	54
RGB Resolution (H)	224	224	224	224	224	224
RGB Resolution (Y)	396	394	394	396	396	398
Rotation Center (x) (meters)	0	0	0.09	-0.17	0	0.02
Rotation Center (z) (meters)	0.11	0.02	0.02	-0.08	0.04	-0.12
Collider Size (x) (meters)	0.34	0.23	0.28	0.49	0.33	0.24
Collider Size (y) (meters)	1.41	1.41	0.9	0.84	1.23	0.43
Collider Size (z) (meters)	0.33	0.27	0.41	0.29	0.44	0.38
distance	-	0.38	0.7	0.79	0.8	0.92

Table 10. **Five Nearest Neighbor Embodiments for Stretch RE-1 in Training Data.**

Goal encoder. We follow the Text Transformer from the pretrained SIGLIP-VIT-B/16 to encode the given natural language instruction into goal embedding $t \in \mathbb{R}^{64\times h}$, where $h = 768$ is the hidden dimension and this Text Transformer returns 64 tokens after padding. Before passing the goal embedding to the transformer state encoder, we always project the embedding to the desired dimension $d = 512$, resulting in $g \in \mathbb{R}^{64\times512}$.

Transformer State Encoder. This module summarizes the state at each timestep as a vector $s \in \mathbb{R}^d$. The input to this encoder includes two visual representations $v^{1,2}$, the goal feature g, and an embedding f of a `STATE` token. These features are concatenated and fed to the non-causal transformer encoder. The output corresponding to the `STATE` token is the state feature vector $s \in \mathbb{R}^d$ which summarizes the state at each timestep. This feature vector is a goal-conditioned visual state representation.

Causal transformer decoder. We use a causal transformer decoder to perform explicit memory modeling over time. This can enable both long-horizon (*e.g.*, exhaustive exploration with backtracking) and short-horizon (*e.g.*, navigating around an object) planning. Concretely, the causal transformer decoder constructs its state belief b^t using the sequence of state features $\mathbf{s} = \{s^j|_{j=0}^{j=t}\}$ within the same trajectories. To avoid recomputing the attention on the previous state features, we follow PoliFormer [62] to use KV-Cache to store the past **K**ey and **V**alue into two cache matrices in each attention layer. Therefore, we only perform feedforward computation for the most current state feature s^t.

Linear actor-critic head. With the latest state belief b^t, we simply use a linear actor-critic head to project it to predict action logits over the action space. For RL-finetuning, the linear actor-critic head also predicts a value estimate about the current state.

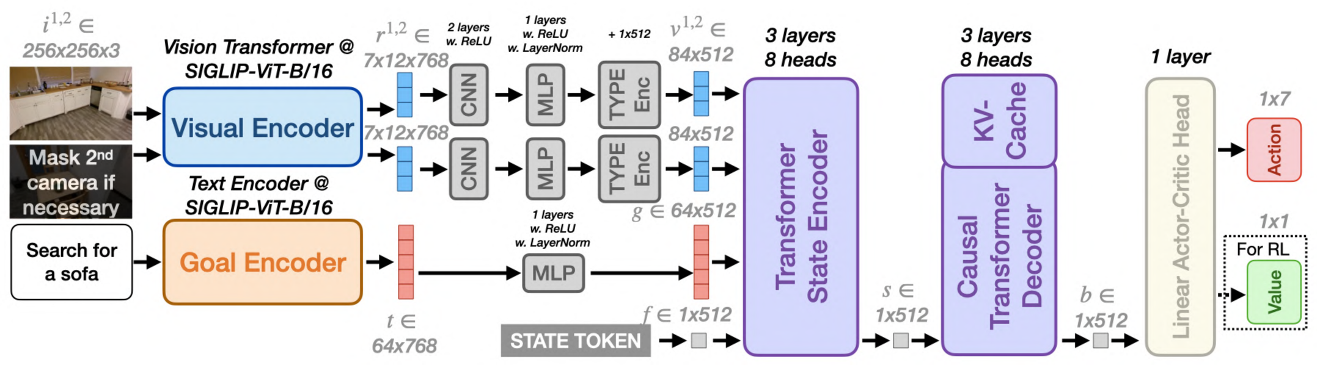


Figure 12. **RING architecture**. The notations in gray correspond to hidden feature vectors and the black text on top of each module indicates the hyperparameters for that module. RING accepts visual observations and a language instruction as inputs and predicts an action to execute. During RL finetuning, RING also predicts a value estimate. We mask the image from the 2^{nd} camera with all 0 for the embodiments with only one camera, such as LoCoBot and Unitree. More specifically, we use the Vision Transformer and the Text Encoder from SIGLIP-ViT-B/16 as our visual encoder and goal encoder. After encoding, we compress and project the visual representation r and text embedding t to v and g, respectively, with the desired dimension d. Next, the Transformer State Encoder encodes v, g, along with `state` token embedding f into a state feature vector s. The Causal Transformer Decoder further processes s, along with previous experiences stored in the KV-Cache, to produce the state belief b. Finally, the Linear Actor-Critic Head predicts action logits (and, during RL finetuning, a value estimate) from b.

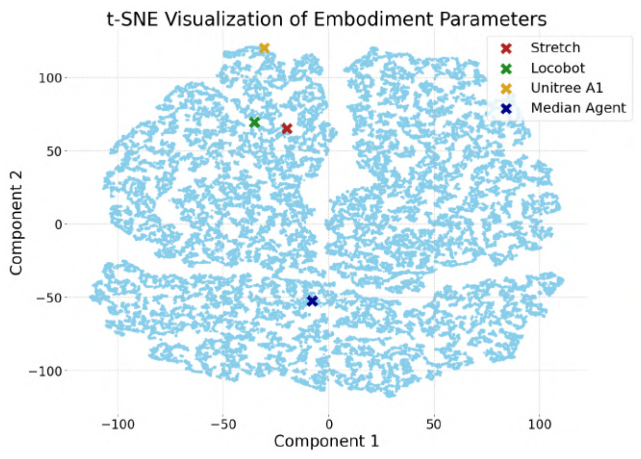


Figure 13. t-SNE visualization of the embodiment parameters $\mathbf{c}_e \in \mathbb{R}^{19}$ for 50k random agents. The three specific robots are also shown for visualization (they are not included in our training set).

10. Nearest Neighbor Embodiments to Real Robots in our Training Data

Fig. 13 presents a t-SNE visualization of the embodiment parameters $\mathbf{c}_e \in \mathbb{R}^{19}$ for 50k samples from the random embodiments in our training set (examples showin in Fig. 14). We also show the corresponding parameters for Stretch, LoCoBot, and Unitree A1 for visualization purposes. Our random embodiments range widely over the space of possible embodiments, with many closely approximating each of the three real robots. Tables 10, 11, and 12 list the five nearest neighbors to each robot in the compressed t-SNE space and their corresponding embodiment parameters. Although the nearest neighbors do not exactly match each robot's em-

		Nearest Neighbors				
	Locobot	N1	N2	N3	N4	N5
Camera Position (x) (meters)	0	-0.09	0.12	0.03	-0.06	-0.1
Camera Position (y) (meters)	0.87	1.01	0.81	0.39	0.85	0.42
Camera Position (z) (meters)	0	-0.1	-0.05	-0.1	-0.02	0.09
Camera Pitch (degrees)	0	0	0	-1	0	1
Camera Yaw (degrees)	0	0	0	0	0	0
Vertical FoV (degrees)	42	45	44	42	45	45
RGB Resolution (H)	224	224	224	224	224	224
RGB Resolution (Y)	396	396	394	394	392	392
Rotation Center (x) (meters)	0	0.04	0.1	0.1	-0.02	-0.15
Rotation Center (z) (meters)	0	-0.13	0	0.13	0.02	-0.12
Collider Size (x) (meters)	0.35	0.27	0.36	0.37	0.27	0.42
Collider Size (y) (meters)	0.89	1.28	1.23	0.86	1.46	0.59
Collider Size (z) (meters)scale z	0.4	0.43	0.23	0.36	0.36	0.45
distance	-	0.18	0.22	0.34	0.41	0.42

Table 11. **Five Nearest Neighbor Embodiments for LoCoBot in Training Data.**

		Nearest Neighbors				
	Unitree A1	N1	N2	N3	N4	N5
Camera Position (x) (meters)	0.01	0.08	0.03	-0.01	-0.04	0.1
Camera Position (y) (meters)	0.3	0.56	0.37	0.85	0.55	0.82
Camera Position (z) (meters)	0.27	-0.11	0.06	0	0.12	0.02
Camera Pitch (degrees)	0	-3	-2	-4	-5	-5
Camera Yaw (degrees)	0	0	0	0	0	0
Vertical FoV (degrees)	42	49	49	51	50	51
RGB Resolution (H)	270	224	224	224	224	224
RGB Resolution (Y)	480	448	446	448	446	446
Rotation Center (x) (meters)	0	-0.07	0.05	-0.07	-0.09	-0.14
Rotation Center (z) (meters)	0.04	-0.02	0	-0.12	0.12	0.11
Collider Size (x) (meters)	0.3	0.46	0.27	0.35	0.27	0.49
Collider Size (y) (meters)	0.34	1.24	0.45	1.47	0.67	1.39
Collider Size (z) (meters)	0.64	0.34	0.37	0.36	0.33	0.39
distance	-	0.76	0.78	1.04	1.1	1.12

Table 12. **Five Nearest Neighbor Embodiments for Unitree A1 in Training Data.**

Figure 14. **Random embodiments in the AI2-THOR simulator**. Right column shows the egocentric view from the main camera and the left column shows a third-person view of the agent –white boxes indicate the robot colliders for visualization purposes only.

bodiment, they are sufficiently similar across different parameters. This extensive coverage of the embodiment space and proximity to real-world embodiments ensure consistent zero-shot generalization to all three robots.

It is worth noting that some parameters of the Unitree A1 fall outside the distribution of the training data. For example, we sample the collider size within the range $[0.2m, 0.5m]$ for both the x and z axes, whereas the Unitree A1 has a length of $0.64m$. This demonstrates that RING has the potential for out-of-distribution generalization.

11. Limitations

Although RING has the advantage of being deployable on a wide range of embodiments without any privileged information about its current body, when available it may be beneficial to have a policy explicitly conditioned on the current embodiment specification. This might lead to improved performance and more desirable behaviors, such as increased efficiency and collision avoidance.

We train RING-EMB-COND by explicitly providing the embodiment information to the policy. The embodiment parameters are represented as a configuration vector $\mathbf{c}_e \in \mathbb{R}^{19}$, with each dimension corresponding to a specific embodiment parameter listed in Table 1. This information is passed as an additional token to the Transformer State Encoder. We use a simple MLP to project $\mathbf{c}_e$ to the desired feature dimension $e \in \mathbb{R}^{1\times512}$ before passing it to the encoder.

Table 13 evaluates the 2 versions of the policy on our custom benchmark consisting of 2,000 random embodiments across 2,000 scenes, comparing metrics such as *Success Rate*, *Success Weighted by Collision (SC)*, *Collision Rate (CR)*, and *Safe Episode (percentage of episodes without any collisions)*.

The results do not show a clear benefit to conditioning the policy on embodiment information. This could be due to several reasons. It is possible that most relevant information about environment hazards and agent motion can be already inferred from visual observations. It is also possible that a significant fraction portion of collisions (both with an without embodiment specification provided) occur with objects that never enter the agent's visual field, in which case extra information about its own embodiment would not help. Alternatively, a more effective method for conditioning the policy on the parameters may exist. Future work should explore this with additional examination of agent-environment collision and and designing improved policy architectures to better integrate embodiment parameters, ultimately training a more efficient and robust policy that explicitly incorporates embodiment information.

Model	Ablations	Success ↑	SEL ↑	SC ↑	CR ↓	Safe Episode ↑
	Body Config					
RING	✗	67.62	56.24	42.53	**7.77**	**46.90**
RING-EMB-COND	✓	**69.44**	**57.42**	**44.69**	8.0	46.54

Table 13. **Conditioning RING on embodiment parameters**. We explicitly provide the embodiment parameters to the policy (RING-EMB-COND) and compare with RING without any information about the embodiment. Both policies are evaluated on a custom benchmark consisting of 2000 random embodiments in 2000 scenes.