Supplementary Appendices for *PoliFormer: Scaling On-Policy RL with Transformers Results in Masterful Navigators*

These appendices contain additional information about our:

- Zero-shot real-world applications (App. A),
- Training procedure (App. B),
- Environment, benchmarks, and quantitative real-world experiments (App. C),
- Simulation evaluations (App. D), and
- Limitations (App. E).

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

- Six real-world qualitative videos where POLIFORMER performs the everyday tasks of Section 4.4 (recall also Figure 3), and
- Four qualitative videos in simulation showing our POLIFORMER's behavior in the four benchmark environments (CHORES, PROCTHOR, AI2-iTHOR, and ARCHITECTHOR).

A Details about Zero-shot Real-world Downstream Applications using an Open-Vocab Object Detector and VLM

By specifying POLIFORMER's goal purely using b-boxes, we produce POLIFORMER-BOXNAV. POLIFORMER-BOXNAV is extremely effective at exploring its environment and, once it observes a bounding box, takes a direct and efficient path towards it. We now describe how we utilize this behavior to apply POLIFORMER-BOXNAV zero-shot to a variety of downstream applications by leveraging an open vocabulary object detector (Detic [18]) and a VLM (GPT-4o [89]).

Open Vocabulary ObjectNav. To perform open vocabulary object navigation (*i.e.*, where one must navigate to any given object type), we simply prompt the Detic object detector with the novel object type, for example, `Bicycle`. As POLIFORMER-BOXNAV relies on the b-box as its goal specification, it finds a bicycle in the scene smoothly.

Multi-target ObjectNav. To enable multi-target object goal navigation, we make a few simple modifications to the inputs and output of the Detic detector. On the input side, we query with multiple prompts simultaneously (one for each object type); for instance, `HousePlant`, `Toilet`, and `Sofa`, as shown in Fig. 3 (bottom-left). We then, on the output side, only return the b-box with the highest confidence score. Since the returned b-box also contains the predicted object type, we know what the target object the agent finds is when issuing a `Done` action. Therefore, we remove the found target from the list of target types, and reset the POLIFORMER's KV-cache. If the agent issues a `Done` action without a detected b-box, we terminate episode and consider it a failure. As a result, the agent is required to find all the targets from the list of target types to succeed in an episode.

Human Following. We change the Detic prompt to `Person`. Once a b-box is detected, POLIFORMER drives the agent to approach it. Our experiment participant continues to walk away, so the agent keeps approaching them to minimize the distance.

Object Tracking. In this example, we control a remote control car that moves in the enviornment, and prompt the agent to find the car. Similar to **Human Following**, we change the prompt to `Toy Truck` in this example. As a result, the agent keeps trying to move closer to the detected b-box of the RC car, while avoiding collisions with objects in the dynamic scene.

Room Navigation. In this example, shown in Fig. 3 (middle-left), we provide no detections to the agent. As the agent sees no detections, it continuously explores the scene. As the agent explores, we

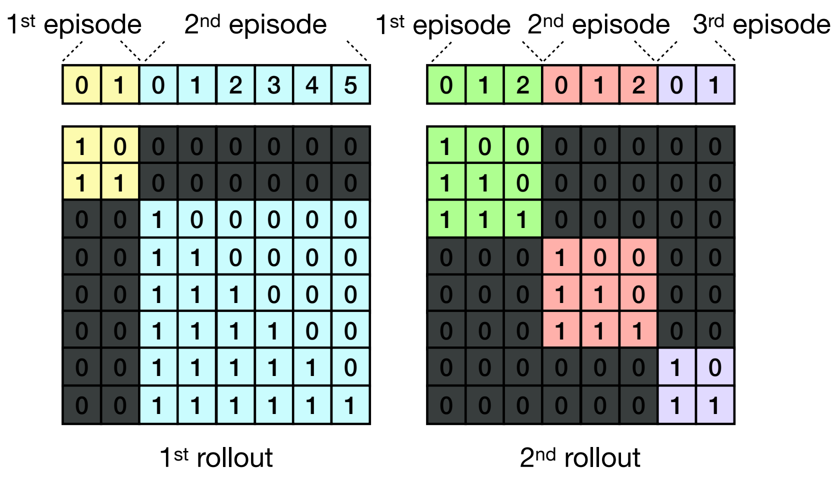

Figure 4: Attention Masks for training with block lower triangular structure.

query GPT4-o every 5 timesteps with the prompt `Am I in a Kitchen? Please return Yes or No.` with the most recent visual observation. Once GPT-4o returns `Yes`, the agent issues a `Done` action to end the episode.

Instance Description Navigation. In this example, shown in Fig. 3 (upper-left), the agent is prompted to find a specific book titled "Humans". Detic can generate open-vocabulary bounding boxes using instance-level descriptions but we found that doing this alone leads to high false-positive rates. To reduce these errors, we use GPT4-o to filter positive detections from Detic. In particular, a sample filtering prompt is "Is there a book titled "Humans" in this image? Please return Yes or No.". We find this combination works well in practice. The agent, not GPT-4o, remains responsible for deciding when it has successfully completed its task, and in the Fig. 3 example sees many books in its search but perseveres and eventually finds the correct one.

B Additional Training Details

Reward Shaping. For reward shaping, we follow EmbCodebook [85] and PROCTHOR [11] and use the implementation in AllenAct [90]: $\mathcal{R}_{penalty}+\mathcal{R}_{success}+\mathcal{R}_{distance}$, where $\mathcal{R}_{penalty} = -0.01$ encourages an efficient navigation, $\mathcal{R}_{success} = 10$ when the agent successfully completes the task ($= 0$ otherwise), and $\mathcal{R}_{distance}$ is the change of L2 distances from target between two consecutive steps. Note that we only provide a nonzero $\mathcal{R}_{distance}$ if the new distance is less than previously seen in the episode. We do not enforce a negative reward for increasing distance. This formulation encourages exploration.

Episodic Attention Mask. During training, to ensure that the causal transformer decoder cannot access observations or states across different episodes, we construct the episodic attention mask to only allow the past experiences within the same episode to be attended. In Fig. 4, we show a couple of possible rollouts collected during training. With the episodic attention mask, observations and states in an episode can only attend to previous ones within the same episode, in contrast with a naive causal mask where they could also potentially attend to observations and states in previous episodes.

Hyperparameters for Training. Tab. 3 lists the hyperparameters used in our training and model architecture design. Please find more details such as scene texture randomization, visual observation augmentations, and goal specification randomization when using text instruction in our codebase.

Training and Model Details	
Parameter	**Value**
Allowed Steps	600 (Stretch RE-1), 500 (LoCoBot)
Total Rollouts	192 (Stretch RE-1), 384 (LoCoBot)
Learing Rate	0.002
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	0.01
Training Stages	3
Steps for PPO Update Stage 1	32
Steps for PPO Update Stage 2	64
Steps for PPO Update Stage 3	128
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 3: Hyperparameters for training and model architecture.

C Additional Details about Environment, Benchmarks, and Real-World Experiments

Action Space. Following prior work using AI2-THOR, we discretize the action space for both LoCoBot and Stretch RE-1. For LoCoBot, we discretize the action space into 6 actions, including {`MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, `LookDown`, `Done`}, where `MoveAhead` moves the agent forward by 0.2 meters, `RotateRight` rotates the agent clockwise by 30° around the yaw-axis, `RotateLeft` rotates the agent counter-clockwise by 30° around the yaw-axis, `LookUp` rotates agent's camera clockwise by 30° around the roll-axis, `LookDown` rotates agent's camera counter-clockwise by 30° around the roll-axis, and `Done` indicates that the agent found the target and ends an episode. We follow previous works [11, 17, 85] to use the same action space for LoCoBot for a fair comparison. For Stretch RE-1, we remove the `LookUp` and `LookDown` camera actions, and add `MoveBack`, `RotateRightSmall`, and `RotateLeftSmall` to the action space, where `MoveBack` moves the agent backward by 0.2 meters, `RotateRightSmall` rotates the agent clockwise by 6° around the yaw-axis, and `RotateLeftSmall` rotates the agent counter-clockwise by 6° around the yaw-axis. Again, this action space is identical to the one used in prior work [6] for fair comparison.

Success Criteria. We follow the definition of Object Goal Navigation defined in [3], where an agent must explore its environment to locate and navigate to an object of interest within an allowed number of steps n. The agent has to issue the `Done` action to indicate it found the target. The environment will then judge if the agent is within a distance d from the target and if the target can be seen in the agent's view. An episode is also classified as failed if the agent runs more than n steps without issuing any `Done` action. Across different benchmarks, n and d vary depending on the scenes size and complexity and agent's capabilites. We follow ProcTHOR [11] to use $n = 500$ and $d = 1$ meter for LoCoBot, and follow CHORES-$\mathbb{S}$ [6] to use $n = 600$ and $d = 2$ meters for Stretch RE-1.

SPL and SEL. Success Weighted by Path Length (SPL) and Success Weighted by Episode Legnth (SEL) are two popular evaluation metrics to evaluate how efficient an agent is to find the target. SPL is defined as $\frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{max(l_i,p_i)}$, where N is the total number of episodes, S_i is a binary indicator

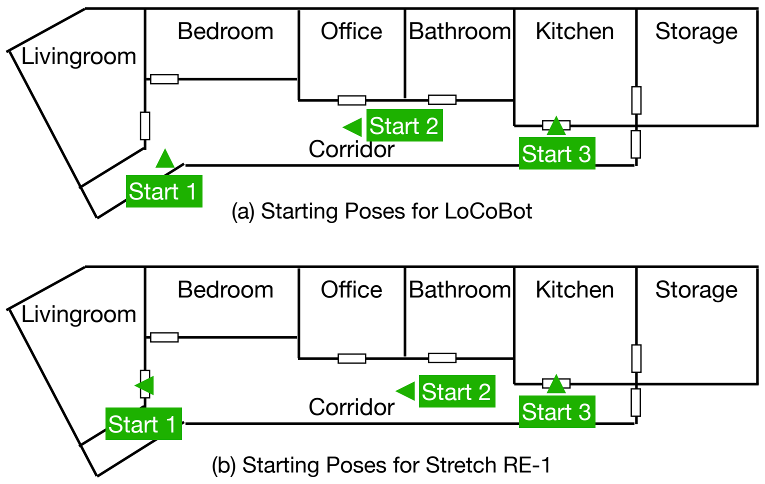

Figure 5: Starting Poses of (a) LoCoBot and (b) Stretch RE-1 used in the real world experiments. The arrow direction indicates where the agent faces with.

of success for episode i, l_i is the shortest travel distance to the target, and p_i is the actual travel distance. SEL is defined similarly: $\frac{1}{N}\sum_{i=1}^{N} S_i \frac{w_i}{max(w_i, e_i)}$, where w_i is the shortest number of steps to find the target, and e_i is the actual number of steps used by the agent. By definition, SPL focuses on how far the agent has travelled, while SEL focuses on how many steps the agent has used (which also penalizes excessive in-place rotation). SPL can be derived by computing the geodesic distance between the agent's starting location and the target's location, while SEL needs a planner with priviledged environment information to calculate number of steps of expert trajectories. Therefore, we follow ProcTHOR [11] to report SPL to evaluate the LoCoBot agent, since those benchmarks do not provide planner, while we follow CHORES-$\mathbb{S}$ [6] to report SEL, since expert trajectories are available.

Real-world Experiment Setup. For the experiments using LoCoBot, we follow Phone2Proc [17] to use the same five target object categories, including `Apple`, `Bed`, `Sofa`, `Television`, and `Vase`, and the three starting poses, shown in Fig. 5 (a). Among those target 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, and `Vase` can be found in the Livingroom, Corridor, Office, and Kitchen. For the experiments using Stretch RE-1, we follow SPOC [6] to use the same six target object categories, including `Apple`, `Bed`, `Chair`, `HousePlant`, `Sofa`, and `Vase`, and the three starting poses, shown in Fig. 5 (b). Among the categories not mentioned above, `Chair` can be found in the Living room, Office, and Kitchen, and `HousePlant` can be found in the Living room, Office, Bathroom, and Kitchen.

D More Simulation Evaluations

Performance Variance. On CHORES-$\mathbb{S}$, since we follow SPOC [6] to apply test-time data augmentation and non-determinstic action sampling, we found that performance varies even using the same checkpoint, especially given that we are only evaluating on 200 episodes. As a result, we re-evaluate our POLIFORMER and SPOC*[4] 16 times and report mean success rate (mSR) and standard deviation (std). POLIFORMER achieves 82.5% mSR with 1.897 std, while SPOC* achieves 56.7% mSR with 2.697 std. This result indicates that POLIFORMER not only achieves a higher mSR than SPOC*, but also exhibits more reliably consistent behavior, *i.e.* a lower std, when run on the same episodes multiple times.

[4]SPOC* is similar to SPOC but is trained on more expert trajectories (2.3M vs 100k).

Inputs	Model	Loss	**EasyObjectNav**	**RegularObjectNav**	**HardObjectNav**
			Success (SEL)	Success (SEL)	Success (SEL)
RGB+text	SPOC [6]	IL	62.9 (40.5)	48.2 (38.9)	34.05 (27.4)
	SPOC*	IL	69.7 (43.3)	53.5 (34.3)	31.0 (19.6)
	PoliFormer	RL	**89.0** (**62.1**)	**82.6** (**71.8**)	**72.3** (**62.8**)
RGB +text+b-box	SPOC	IL	90.3 (67.7)	78.7 (62.6)	70.6 (52.5)
	PoliFormer	RL	**98.1** (**86.5**)	**90.4** (**79.6**)	**86.0** (**75.0**)
RGB+b-box	PoliFormer	RL	97.1 (83.2)	91.9 (79.8)	87.6 (75.0)

(a) Stretch RE-1 on Chores-$\mathbb{S}$

Table 4: Large-scale evaluation results with different difficulty tiers. We evaluate performance on 2,000 episodes per tier.

Larger Scale Simulation Benchmark using Stretch RE-1. To further analyze PoliFormer's performance through different difficulty settings, we construct 3 different levels of Object Goal Navigation benchmarks, `EasyObjectNav`, `RegularObjectNav`, and `HardObjectNav`, where each level contains 2k episodes, using Stretch RE-1. We construct these differentiated tasks by ensuring the oracle expert path length between the agent and target is 1 to 3 meters long for `EasyObjectNav`, greater than 3 meters for `RegularObjectNav`, and larger than 10 meters for `HardObjectNav`. The results are shown in Tab. 4. We observe that every model performs better as the agent is closer to the target at the episode start. In addition, on `EasyObjectNav` the agent barely needs exploration to find the target. Thereby, we find that PoliFormer lagging behind PoliFormer-BoxNav by $\sim$9% could result from a *Recognition Issue*. Moreover, the gap on `HardObjectNav` is widened to $\sim$13.7%, and it could result from an additional *Exploration Issue*. The performance gap between `HardObjectNav` and `EasyObjectNav` could also support that an *Exploration Issue* exists, but not just the *Recognition Issue*.

E Additional Discussion on Limitations

Depth Sensor. It is important to note that PoliFormer is not equipped with a depth sensor (which has been proven to be effective for manipulation). While the lack of depth sensor does not affect our agent's performance on navigation, we acknowledge that integrating the depth sensor into our visual representation is an interesting direction for future work, especially when considering mobile-manipulation extensions.

Discretized Action Space. To have a fair comparison with baselines, we use the same discretized action space in this work (see Sec. C). The discretized action space might not be efficient and realisitc in many real-world scenarios where the agent must act in a timely manner.

Cross-embodiment. In this paper, we demonstrate that we can train PoliFormer using LoCoBot and Stretch RE-1. However, we have not yet explored training a single PoliFormer for both embodiments. We leave this interesting research direction as future work.

Further Scaling. Our training and validation curves strongly suggest that even further scaling of model parameters and training time may lead to even more masterful models than those we have trained in this work. This perspective is exciting and we hope to enable further scaling with more computation resources and better visual foundation models in the near future.

Failure Analysis. The main mode of failure for PoliFormer is the agent's limited memory. PoliFormer clearly demonstrates memorization capabilities and is able to perform long-horizon tasks by exploring large indoor scenes without access to explicit mapping. However, as the trajectories get longer (specifically after visiting more than 4 rooms in an environment), the agent's recollection of the rooms it has explored detoriates and the robot might re-visit rooms that it has explored previously.