APPENDIX

A. Results Visualization

We encourage the reader to visit our website (`robot-flare.github.io`) for visualizations of trajectories generated by FLaRe both in simulation and in the real world, including performances visualization, behavior analysis, and failure mode analysis.

B. Hyperparameter

Training and Model Details	
Parameter	**Value**
Total Rollouts	32
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
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 IV: Hyperparameters for training and model architecture.

C. Number of Training Steps

The base SPOC model that we evaluted and fine-tuned upon is trained for 50k gradient update steps on a total of 100k episodes of demonstrations across the CHORES tasks, where the training hyperparameter and training data is exactly the same as in the original SPOC paper.

For navigation tasks that do not involve manipulating objects (i.e. ObjectNav and RoomVisit), FLaRe performs RL fine-tuning for 20M steps, while all other fair-comparison baseline methods perform RL training for 60M steps. For mobile manipulation tasks (i.e. Fetch and Pickup), FLaRe performs RL fine-tuning for 50M steps, while all other fair-comparison baseline methods perform RL training for 100M steps. For adaptation tasks, we run FLaRe fine-tuning for 50M steps on ObjNavRelAttr and ObjNavAfford, and 20M steps on RoomNav. For cross-embodiment, we run FLaRe for 30M steps.

All of the aforementioned experiments use the same base SPOC mode, with exactly the same set of hyperparameters.

D. CHORES Benchmark

A big portion of FLaRe's evaluation is carried out on the CHORES benchmark. We provided detailed information about this benchmark, including the observation space, action space, and task specifications.

1) Observation Space: The observation space of CHORES consists of two ego-centric 384 × 224 RGB camera pointing towards orthogonal directions, where one points towards the direction of navigation and the other points at the arm. Additionally, a natural language text instruction is re-sampled at the start of each episode and appended to the observation to specify what the robot should be doing.

2) Action Space: The action space of CHORES consists of 20 discrete actions: Move Base (± 20 cm), Rotate Base (±6°, ±30°), Move Arm (x, z) (±2 cm, ±10 cm), Rotate Grasper (±10°), pickup, dropoff, done with subtask, and terminate.

3) Tasks Specifications: We describe the CHORES tasks in Table. V. For each task, if the robot exceeds the maximum steps, the episode is terminated and marked as failed.

For each task, we splited a total of 191,568 houses from ProcThor [46] into training and testing sets with a ratio of 10:1, to ensure that the test evaluation is conducted in unseen houses.

E. The SPOC Model

In this work, we use a slightly modified version of the SPOC model [7] inspired by Poliformer [10], where the transformer decoder block in SPOC is replaced by the decoder from Llama 2 LLM [64] to speed up training and inference. At each step, the SPOC model takes in the new observations consisting of two RGB images and a text instruction. Each of these images are separately passed through a frozen **vision transformer model** (DinoV2 [53]) to extract a set of visual tokens. These tokens, along with an embedding of the natural language instructions using a pre-train text encoder T5 [65], are summarized by a **transformer state encoder** to produce the observation representation. A **causal transformer decoder** then decodes the observations feature across all steps within the current episode into a belief vector that is passed through an actor head to generate the action prediction. We provide a visualization of our model in Fig. 7, and explain each of these components in detail below.

1) Vision Transformer Model: We use DINOv2 as the visual foundation backbone because of its remarkable ability to make dense predictions that generalize across sim and real. Our input to the visual backbone are two RGB observations i_a and i_b. $i_a \in \mathbb{R}^{H\times W\times 3}$ is captured by the navigation camera and $i_b \in \mathbb{R}^{H\times W\times 3}$ is captured by manipulation camera, where H and W are the height and width of the image. The visual backbone then produces a patch-wise representation $r \in \mathbb{R}^{\frac{H}{14}\times\frac{W}{14}\times h}$, where h is the hidden dimensions of the visual representations. r is then reshaped and projected to generate visual tokens $v_{\text{raw}} \in \mathbb{R}^{n_{\text{patch}}\times d_{\text{encoder}}}$. A learnable camera-type embedding is then added to this visual tokens to ensure the model can differentiate between the navigation and the

Task	Description & Example	Max Steps
ObjectNav	Locate an object category: “find a mug”	600
PickUp	Pick up a specified object in agent line of sight: “pick up a mug”	600
Fetch	Find and pick up an object: “locate a mug and pick up that mug”	600
RoomVisit	Traverse the house. “Visit every room in this 5-room house. Indicate when you have seen a new room and when you are done.”	1000

TABLE V: CHORES tasks.

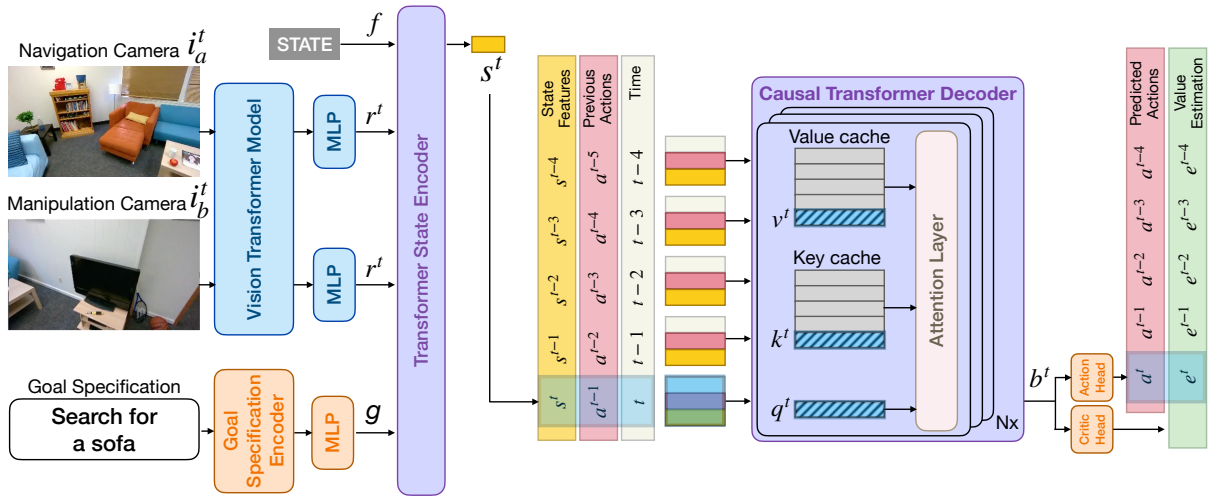

Fig. 7: A visualization of the network architecture of the transformer-based SPOC model that FLaRe fine-tunes upon.

manipulation cameras, resulting in the final visual features v. To ensure sim-to-real transfer, we freeze the DinoV2 weight throughout training.

2) Transformer State Encoder: This module summarizes the observations at each timestep as a vector $s \in \mathbb{R}^d$. The input to this encoder includes the visual representation v, the text feature g, and a learnable STATE token f. We concatenate these features together and feed them to a non-causal transformer encoder. This encoder then returns the output corresponding to the STATE token as the state feature vector. The transformer state encoder digests both visual and text features, and can thus be seen as generating a text-conditioned visual state representation.

3) Causal Transformer Decoder: To deal with partial observability and handle long-horizon tasks, SPOC uses a causal transformer decoder to perform explicit memory modeling over time. The causal transformer decoder consumes the visual representations generated by the transformer state encoder, additively combines them with sinusoidal temporal position encodings and learned previous time step action embeddings, and generates the belief vector used for action generation.

F. Real Robot Setup

Following SPOC [7], we equipped our Stretch RE-1 robot with two identical Intel RealSense 455 fixed cameras, namely the navigation and the manipulation camera. These cameras have a vertical field of view of 59° and are capable of capturing 1280×720 RGB-D images. Both of these cameras point slightly down, with the horizon at a nominal 30°, to optimize the agent's perspective of its functional workspace. The images returned by these cameras are first resized to 396 × 224, and the cropped to 384 × 224, to match the image observations during training.

Same as SPOC, we assess the performance of our models on ObjectNav and Fetch in a 6-room apartment also used in Phone2Proc [62], Pickup in RoboThor [66], and RoomVisit in both environments. The 6-room apartment contains environment variations wholly unseen at train time, including a new configuration (multiple rooms off a long corridor), two new room types (office and corridor), rooms with non-orthogonal wall alignment, and many unseen object instances. For each object in ObjectNav and Fetch, we tested three starting positions: once from the living room, once from the middle of the corridor, and once from the kitchen. We visualize these starting locations in Fig. 5. Below, we provide objects that we tested upon in the real world for each tasks.

1) ObjectNav: Target objects are Sofa, Bed, Chair, Apple, Vase, and Houseplant, each from three starting positions.

2) Fetch: Target objects are Apple, Vase, and Houseplant from the same three starting positions. In one small change from ObjectNav episodes, object instances are replaced with instances which better fit into Stretch's grasping envelope and in some cases at a better height for interaction, but availability and placement are nearly identical.

3) PickUp: Objects are placed on three different surfaces (coffee table, desk, and nightstand) at three different heights. Objects are Apple, Houseplant, Spray Bottle, Mug, and Vase.

4) RoomVisit: The full 6-room apartment is explored, and then partitioned into two 3-room apartments to evaluate the ability of SPOC to explore large and small spaces. We additionally explore a section of RoboTHOR and attached workroom as a novel 3-room apartment.