SPOC: Imitating *Shortest Paths* in Simulation Enables Effective Navigation and Manipulation in the Real World

Supplementary Material

A. Training details

SPOC uses SIGLIP image and text encoders that produce 84×768 ($n_{\text{patch}} \times d_{\text{image}}$) features and 768 dimension feature per text token. We use 3-layer transformer encoder and decoder with 512 dimensional hidden state (d_{visual}) and 8 attention heads for the goal-conditioned visual encoder and action decoder, respectively. We use a context window of 100. All models are trained with a batch size of 224 on 8×A100 GPUs (80 GB memory / GPU) with the AdamW optimizer and a learning rate of 0.0002. Single-task models are trained for 20k iterations, while multi-task models are trained for 50k iterations. Unlike RL which involves simulation in the training loop, IL has the advantage of saving the training episodes to disk offline. Using 16-bit mixed precision training SPOC trains at approximately 1.2 hours per 1000 iterations, an FPS of ≈3500, compared to an FPS of ≈175 for RL implemented using AllenAct [73]. We find that data augmentation both during training and testing is critical for model performance, both in simulation and real, when training on MP4 compressed videos. In particular, we always apply color jitter, gaussian blurring, and random cropping while posterization and sharpness adjustments are applied with specified probabilities to the RGB frames from both cameras. In addition to the RGB frames, we incorporate an `object-in-hand` sensor to indicate whether the agent is holding an object. For details on how these sensors are impremented in real experiments, see Sections C.1 and C.2. In the decoder, we additively combine the embedding of `object-in-hand` sensor with visual representations, sinusoidal temporal position encodings, and learned embeddings of previous action. This sensor is important for tasks involving manipulation, such as FETCH and PICKUP, where it enables the agent to learn when to issue a terminate action to end an episode.

For the model with detection (SPOC w/ GT Det or SPOC w/ DETIC), we encode the coordinates of the bounding boxes using sinusoidal positional encoding followed by a linear layer with LayerNorm and ReLU. We also add coordinate type embeddings to differentiate the 10 coordinates (5 per camera - x_1, y_1, x_2, y_2, and area). The coordinate encodings are then concatenated with the two image features and text features before feeding into Transformer Encoder $\mathcal{E}_{\text{visual}}$. When no object is detected in the image, we use 1000 as the dummy coordinate value and set area to zero.

A.1. RL training details.

We train our RL baselines using DD-PPO with 64 samplers on a single machine with 8 A6000 GPUs, closely following the ProcTHOR implementation. During training, we use the Adam optimizer with a fixed learning rate of `3e-4`. As DD-PPO is an on-policy algorithm, a replay buffer is not utilized during training. The step count for DD-PPO is set at 128. The same data augmentations are applied during both training and eval. The max episode length is 600 steps for all tasks except RoomVisit which has a 1000 step max. The RL baselines train for *double the wall-clock time* compared to SPOC, totaling around 40M environment steps.

We trained SPOC for two days and RL baselines for four days both using a single machine with 8 GPUs, resulting in 384 GPU hours for SPOC and 768 GPU hours for RL baselines

B. Data generation

In this section, we describe our CHORES benchmark, a challenging collection of thousands of tasks corresponding to 10 task types in virtual household scenarios. We first focus on the definition of the procedural houses and then proceed to detail the rationale and details of the included task types.

B.1. Houses

Houses are procedurally generated using ProcTHOR [18], a powerful procedural environment generator for the THOR simulator [39]. ProcTHOR can generate complete household environments by sampling floorplans, adding doors and windows, placing large object types and adding decorations and lights. The differences with respect to the original ProcTHOR houses are (1) the use of specific layouts specifications; and (2) the inclusion of a much larger set of 3D objects imported from Objaverse [17] and complemented with extended annotations for scale, standard pose, and descriptions.

B.1.1 Layouts

House layouts are sampled from the 14 specifications shown in Table 10, where the numbers between parentheses after each layout identifier indicate the relative frequencies of the layout specifications, and the numbers after the dashes are area weights for the rooms or groups of rooms within the corresponding parent groups in the hierarchy. The room types included in the layout specifications are

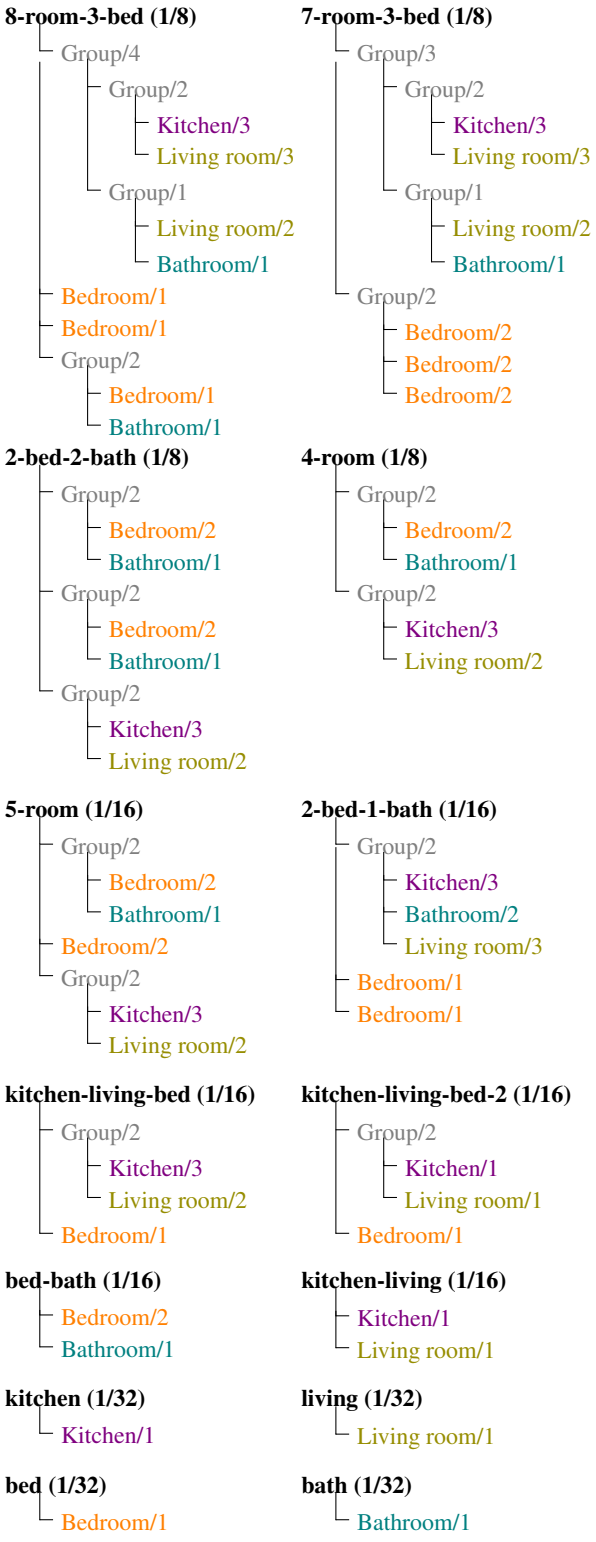


Table 10. **House generation layouts**. The 14 included layouts comprise large-sized, medium-sized and small-sized houses, including one-room houses. The relative frequency of each layout is shown between parentheses after each layout identifier.

kitchen, *bathroom*, *living room*, and *bedroom*.

The resulting layout distribution covers large, medium, and small houses, as well as single-room houses. The average number of rooms per house is 4.5. A total of 191,568 houses are sampled from this distribution, with ratio of 10:1:1 across training, validation, and test. A sample house from each of the 14 layout specifications is shown in Fig. 6.

B.1.2 Assets

Combining assets already present in THOR with a subset of high-quality assets from Objaverse [17], an asset collection we call *ObjaverseHome*, 41,133 3D assets are available to be placed in procedural houses, including structural elements like doors and windows, large pieces of furniture and appliances, or small objects to be placed on free surfaces. For all *ObjaverseHome* assets, we extended the available annotations with scale, standard pose, descriptions, category, and nearest WordNet synset (from the 2022 version of the *Open English WordNet* [23, 47]). The relative frequency of assets assigned to each synset is illustrated in Fig. 4. The total number of synsets used to label the *ObjaverseHome* and THOR assets (equivalent to the number of object categories) is 863. In some tasks, we are interested in the agent being able to recognize more generic identifiers for a target object, which can be obtained in WordNet by retrieving hypernyms of the target object's labeled synset. In combination with the 863 used to label the assets, and excluding too generic hypernyms like *entity.n.01*, *structure.n.01*, or *product.n.02*, up to 1,371 synsets can be used to refer to the 41,133 assets, besides other open-vocabulary referring expression modalities like affordances or descriptions included in CHORESNAV. In addition to synsets shown in Fig. 4, Fig. 7 illustrates the relative frequency of hypernyms that may be used to refer to assets for tasks like OBJNAVAFFORD and OBJNAVRELATTR. See Sec. B.2.1 for details on how these synsets are used to specify targets for CHORESNAV.

Affordances. Using GPT-3.5, we extract five short descriptions of common usages for each synset, given the synset's definition, and a score indicating the confidence in the correctness of the usage. An example query is shown in Table 11. For our 863 synsets, the total number of unique affordances we obtain is 4,315. Some examples are *"Giving on special occasions"* for *present.n.02*, *"Cooking food on stovetop"* for *burner.n.02*, or *"Frozen summer treat on stick"* for *ice_lolly.n.01*.

Once the collection of synset affordances is complete, we refine it by asking GPT-3.5 to determine whether each affordance can be reasonably associated to each *ObjaverseHome* asset labeled with the synset it generated from, this

Figure 6. **Variety of layouts.** Sample houses for each of the 14 layout specifications.

Describe, in five words or less each, up to five common usages of an object with category *"stockpot"* (definition: *"a large pot for making stock, soup, or stew"*). Format each line of your response as [CONFIDENCE] [USAGE DESCRIPTION] where [CONFIDENCE] is a number between 0 and 10 indicating your confidence in the correctness of the usage (10 being most confident).

Table 11. Example GPT query for synset uses.

time conditioning on the asset's description, with a query like the example in Table 12. Please note that we do not perform this refinement step for THOR assets[3].

[3]In order to allow material randomization without compromising the validity of the descriptions and also taking into account the relative uniformity of THOR assets within their respective categories, we decided not to include descriptions for THOR assets.

For each of the following possible uses of an object with category *aerosol* and description *"A blue maya brand can of acrylic spray paint."*, indicate whether each use is a common use of the object. Format each line of your response as [INDEX] [CONFIDENCE] where [CONFIDENCE] is a number between 0 and 10 indicating your confidence in the correctness of the usage (10 being most confident) and [INDEX] is the index of the usage. *1. Applying hair spray.* *2. Dispensing air freshener.* *3. Dispensing fine particles.* *4. Using spray paint.*

Table 12. Example GPT query for object uses filtering.

Finally, we generate embeddings for each affordance using GPT-3's *ada* [5] to create clusters of similar affordances, corresponding to sets of embeddings with cosine similar-

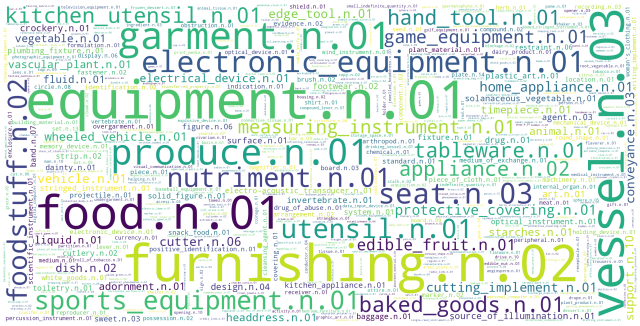


Figure 7. **Hypernyms of synsets in procedural houses**. Word map of hypernyms with font size scaled by the counts of corresponding hyponyms among the synsets in Fig. 4.

ity $\geq$0.96. One example of such clusters would be the one composed by the affordances *"Adding flavor to recipes"*, *"Adding flavor to dishes"*, and *"Enhancing the flavor of dishes"*. In total, we find 343 such clusters, with an average size of 2.78 affordances. This information, combined with the full set of affordances and synsets/assets they apply to (those for which the confidence score is $\geq$6) is used to sample OBJNAVAFFORD tasks as shown in Sec. B.3.

B.2. Tasks

We define ten task types for domestic environments requiring the agent to learn and use different behaviors and understand different goal specifications. The behaviors included in our datasets and benchmark are navigation, pick up, fetch, and visit (all rooms in the house). Regarding goal specifications, we refer to goals by object types, affordances, instance descriptions, local references, room types, or relative attributes within a room.

The ten tasks we include in our benchmark serve as indicators of the agent's ability to execute different combinations of elemental navigation, exploration, manipulation, and vision-language skills. Table 13 summarizes the sets of elemental skills required to solve each of the tasks.

B.2.1 Goal specifications

The *navigate* behavior is also combined with a diversity of goal specifications in CHORESNAV, eliciting the agent to identify goal objects through different referring expression types.

Please note that we select target object types by their annotated synset among the ones in the *Open English WordNet* [23, 47], which in practice falls back to choosing the lowest hypernym for the sampled synset shared by any objects present in the scene[4]. For example, an *armoire* asset coarsely labelled as *furniture.n.01*, can coexist in a house with a chair labeled as *chair.n.01* and a table labeled as *table.n.02*, so any of three assets shall be referred to by *furniture.n.01*, which is the lowest common hypernym.

[4]Despite an effort to annotate all objects in the scene with the most precise possible synset, more generic hypernyms are sometimes used.

		Exploration	Object identif.	Set identif.	Affordance identif.	Feature identif.	Manipulation	Set completion
1	OBJNAV	✓	✓					
2	PICKUP		✓				✓	
3	FETCH	✓	✓				✓	
4	ROOMVISIT	✓		✓				✓
5	OBJNAVROOM	✓	✓	✓				
6	OBJNAVRELATTR	✓	✓	✓				
7	OBJNAVAFFORD	✓	✓		✓			
8	OBJNAVLOCALREF	✓	✓	✓				
9	OBJNAVDESC	✓	✓			✓		
10	ROOMNAV	✓		✓				

Table 13. **Skill sets required for all task types**. Different sets of skills are required to solve each task. *Set identification* encompasses the ability to recognize a room type as well as explicit local configurations or relative attributes of objects within a room.

OBJECTTYPE goal. The natural language instruction given to the agent refers to the most precise lemma for the chosen synset (*i.e.* the one used less often by other synsets). Any object in the scene that can be addressed by the lemma in the instruction is a valid goal for the task.

OBJECT-ROOM goal. Similar to *object type*, but further constraining the valid goal objects to be present at the beginning of the task in a room of a specific type. This specification type implies that more instances of the target type also exist in at least one room of any of the other types.

RELATIVEATTRIBUTE goal. We consider six relative attributes regarding objects of some given type in a specific room type: *highest/lowest*, *largest/smallest*, and *nearest/furthest* to/from some anchor type (for which a unique object exists in the room), implying the presence of more than one object of the target type in the specified room.

AFFORDANCE goal. For affordances, we provide the agent with a hypernym covering a subset of the objects in the scene such that some of the objects in the subset are plausible candidates to provide the requested affordance while some others might not, *i.e.*, the agent must be able to determine which objects correspond to the given hypernym, and also which of these are also providing the requested affordance.

LOCALREFERENCE goal. We consider two types of local reference specifications: (1) an object of given type near two other objects of other types, and (2) an object of given type on top of an object of another type. In either case, this task specification implies than more than one object of the target type exists in the house.

DESCRIPTION goal. In this case, the specification is an open-vocabulary description of the target object type, assuming uniqueness of the valid target, meaning that no other object in the house responds to the given description. All OBJNAVDESC tasks use *ObjaverseHome* assets as targets, since we do not include asset descriptions for THOR assets.

ROOM goal. In ROOMNAV, the goal is just a room type, which can correspond to one or more of the rooms in the house.

B.3. Sampling Methods

We first describe a set of criteria applied onto potential targets to decide upon their acceptance as task goals, and then proceed to outline each task sampler.

Filtering. In order for objects to be valid targets, we impose certain constraints dependent on how the agent needs to interact with the object. For example, in **navigation** tasks, we impose that the target objects are below a maximum height of 1.1 m, that a path can be computed by THOR to an interactable location near the objects, and that the bounding boxes for the objects have (1) a largest face with diagonal larger than 10 cm; and (2) a middle dimension of at least 4 cm. In **manipulation** tasks, we additionally impose that the object can actually be picked up and the maximum dimension of its bounding box is less than 50 cm.

ROOMVISIT **sampling.** The agent is just spawned at a random location within a room sampled uniformly among the house rooms.

OBJNAV **sampling.** We attempt to balance the distribution of target synsets by keeping counters of the number of times we have sampled each synset and sequentially selecting candidate synsets from a random sequence of synsets available in the scene sampled with replacement with weights given by the inverse of the respective counts. If any of the scene objects with the sampled synset passes the *navigation* filters described above, we accept the task. The agent is then spawned as in ROOMVISIT sampling.

FETCH **sampling.** Similar to OBJNAV sampling, but using the *manipulation* filter, also enforcing the object to be valid for pickup.

PICKUP **sampling.** Similar to FETCH sampling, but with the agent spawned at a random location where any of the target objects is at interactable distance, and oriented such that the manipulation camera is aligned with the object's center.

OBJNAVROOM **sampling.** Similar to OBJNAV sampling, but enforcing that the target synset has multiple instances in the house, and that all instances of the target synset for the target room type are in a single room, besides having at least one instance of the target type in any rooms of any other type. If several rooms are feasible, we randomly pick one.

OBJNAVRELATTR **sampling.** In addition to the target type count in OBJNAV sampling, we also keep counts of the number of times we sample different relative attribute types (among *smallest, largest, highest, lowest, nearest to, furthest from*). Similar to OBJNAVROOM sampling, we enforce that the target synset can be found in a specific room type, but this time without enforcing that the object is also present in a different room type. We also enforce that multiple instances of the target synset are present in a single instance of the room type. Then, we try relative attribute types prioritizing by lowest counters. For *smallest* (*largest*), we accept the task if the smallest (largest) object of type in the room has a bounding box with diagonal at least twice (at most half) of the second largest (smallest) object of given synset in the room. For *highest*, we check that the bottom of the bounding box of the highest object of given synset in the room is the highest placed among the objects of the given synset, and also that the top of the second highest box is less than half way between the bottom and top of the highest one. The criterion for *lowest* is the reciprocal. For *furthest from*, we first extract *anchor* objects in each room, which can be *beds, counter tops, dining tables, fridges, sinks, sofas, televisions,* or *toilets*, with the condition that only one instance exists in the room. Then, for the given synset, we check whether the bounding box distance from the second furthest object from the anchor is less than 70% of the distance of the furthest one. The criterion for *nearest to* is built similarly. If the target object (and potentially the anchor) pass the *navigation* filters, the task is accepted.

OBJNAVAFFORD **sampling.** We constrain target synsets to be hypernyms of the synsets in the scene and sample the available hypernyms following the same procedure as in OBJNAV sampling. For each synset in the scene child of the

target hypernym, we collect all confident affordances and merged affordances from the clusters defined above, and list the objects in the scene that provide each affordance. We then randomly sample an affordance and check that at least one of the objects providing the affordance pass the *navigation* filter to accept the task.

OBJNAVLOCALREF **sampling.** This is similar to OBJNAVRELATTR sampling, but having two possible modes: *near* two reference synsets and *on* one reference synset. For *near*, we search for two additional synsets such that two instances in the house have bounding boxes at a distance <50 cm from the bounding box of the current target synset, such that no other such synset triplet is present in the scene and no other such triplet is present in the scene with bounding box distances <2 m, to avoid ambiguity. We constrain the reference instances to not be any of *floors, walls, doors, windows, shelves, drawers, beds, counter tops* or *sofas*. For *on*, we search for a unique combination of the target synset lying on an instance of a reference synset, which we constrain not to be any of *floors, walls, doors, windows, shelves* or *drawers*. In both cases, the target object and the references must pass the *navigation* filters.

OBJNAVDESC **sampling.** Similar to OBJNAV sampling, but constraining the assets for the sampled synset to be part of *ObjaverseHome* (*i.e.*, containing a natural language description), and ensuring that a single instance of the sampled asset is present in the scene. The *navigation* filter must be passed by the target object.

ROOMNAV **sampling.** In this case we keep counts of the number of times we sample different room types, and sample prioritized room types.

B.4. Planners

We define planners able to solve each of the task types with a two-fold goal: (1) generate expert trajectories for supervised learning; and (2) validate the feasibility of a sampled task specification. We first define two subroutines for navigation and manipulation that are reused by several planners and then proceed to outline each planner. Unless otherwise specified, if any phase in the planner fails to complete during the execution of the planner, the trajectory (and task) is discarded.

Navigation subroutine. This subroutine allows the agent to approximately follow a shortest path to a target (noting that the actual shortest path is not necessarily feasible given the discrete set of actions available for the agent). It allows to recompute the path if the agent gets stuck given to discretization errors.

Pick-up subroutine. This subroutine assumes the target object is at interactable distance with the agent's manipulation camera aligned. It executes the following steps: (1) first move the agent arm up until the target object's height; (2) rotate the wrist until the grabber is oriented away from the agent's body; (3) move the arm out until reaching the object's depth; (4) rotate the wrist until the gripper is near the target object; (5) move the arm down until the grabber is in contact with the object; (6) close the gripper; (7) slightly lift the arm with the object in hand. The pickup subroutine also includes an option to retry pickup in case of either failure to grab the object during simulation or random choice (20% probability). This enables the agent to learn local corrections for small deviations from the gold trajectory.

NAVIGATE planner. This planner is used by all tasks in CHORESNAV with the exception of ROOMNAV, *i.e.*, for navigation with all possible goal specifications other than *room* goal. Given the agent spawned in a random location within the house and a list of valid target object identifiers in the scene, the planner selects the nearest target in terms of geodesic distance, and executes the navigation subroutine on the chosen path. Once the final position is reached, the planner proceeds to align the navigation camera with the target object and signal task termination.

ROOMNAV planner. Similar to navigate, the planner determines the nearest room center in terms of geodesic distance and approximately follows the nearest path to the center of the nearest room of the given type via the navigation subroutine. Once the center is reached, the planner signals task termination.

PICKUP planner. The first phase executes navigation to an interactable location for the nearest target object. Then, the agent rotates to align the manipulation camera with the target object and proceeds to execute the pick-up subroutine (with at most one failed pickup phase). Once the object is in the gripper, the planner signals task termination.

FETCH planner. The fetch planner assumes the target object is in view, and initially allows to move the base closer to the target (as in regular navigation), and then proceeds with the pick-up subroutine described above before singnaling task termination.

ROOMVISIT planner. This planner builds a traversal of the rooms in the house via depth first search and approximately follows the path to each room's center using the navigation subroutine. After reaching each room's center a subtask completion action is issued, followed by a task termination signal upon visitng the last room in the house.

Benchmark	Categories	Tasks	Train			Eval.	
			Hs.	Ep.	Fr.	Hs.	Ep.
CHORES -$\mathbb{S}$	15	OBJNAV	10K	99K	5M	200	200
		PICKUP	8K	65K	3M	171	171
		FETCH	13K	114K	12M	172	172
		ROOMVISIT	9K	94K	20M	200	200
CHORES -$\mathbb{L}$	863	OBJNAV	10K	99K	6M	216	216
		PICKUP	10K	92K	4M	173	173
		FETCH	9K	85K	9M	179	179
		ROOMVISIT	9K	94K	20M	200	200

Table 14. Number of unique houses, episodes, and frames (training only) in CHORES.

Benchmark	Categories	Tasks	Train			Eval.	
			Hs.	Ep.	Fr.	Hs.	Ep.
CHORESNAV -$\mathbb{S}$	15	OBJNAV	10K	99K	5M	200	200
		OBJNAVROOM	2K	21K	1M	181	181
		OBJNAVRELATTR	1K	13K	1M	194	194
		OBJNAVAFFORD	2K	23K	1M	197	197
		OBJNAVLOCALREF	2K	19K	2M	142	142
		OBJNAVDESC	1K	13K	1M	144	144
		ROOMNAV	2K	22K	1M	200	200
CHORESNAV -$\mathbb{L}$	863	OBJNAV	10K	99K	6M	216	216
		OBJNAVROOM	10K	93K	6M	156	156
		OBJNAVRELATTR	9K	91K	8M	217	217
		OBJNAVAFFORD	10K	99K	5M	228	228
		OBJNAVLOCALREF	9K	92K	7M	254	254
		OBJNAVDESC	10K	95K	6M	268	268
		ROOMNAV	9K	87K	5M	200	200

Table 15. Number of unique houses, episodes, and frames (training only) in CHORESNAV.

B.5. Exploration behavior

Our exploration planner for investigating shortest-path usability searches rooms in a depth-first manner, as in our ROOMVISIT planner, with one addition: instead of simply navigating to the center of the room, it iteratively navigates toward unseen objects in its current room until at least 75% of objects in that room have been seen, at which point it searches the next room for its target object. Our expert exploration is object-centric - it has access to and uses substantially more privileged information. When a target object is seen during this process, it immediately navigates towards it. This is in contrast to Frontier-Based Exploration [59, 80], where empty sections of an explicit map are filled greedily.

B.6. Benchmark data

In Tables 14 and 15 we list the number of episodes, houses, and expert trajectory frames for the train split of CHORES and CHORESNAV. We also list the number of episodes and houses in the evaluation split. The number of unseen assets present in the Evaluation splits for the four combinations of {CHORES, CHORESNAV } with {-$\mathbb{S}$, -$\mathbb{L}$ } is shown in Table 16.

Regarding the variety in behaviors and natural language instructions, in Fig. 9 we show the distribution of planner trajectory lengths for all tasks in each of the training splits, and in Fig. 8 we show the distribution of target synsets across all splits. Beyond this variety, please note

Benchmark	Unseen assets	% unseen assets	Houses
CHORES -$\mathbb{S}$	1,746	24.0%	727
CHORES -$\mathbb{L}$	1,740	24.2%	737
CHORESNAV -$\mathbb{S}$	2,129	23.5%	1,209
CHORESNAV -$\mathbb{L}$	2,079	22.1%	1,413

Table 16. Unseen assets in evaluation houses.

that some task types introduce a large number of additional concepts in the natural language instructions: for example, each OBJNAVDESC instruction uses a unique natural language description of the target asset.

Regarding the complexity of the (virtual) training and evaluation environments, in Fig. 10 we show the distribution of the number of rooms for houses included in each of the splits. Note that we only count each house once, regardless of how many times it appears in the corresponding dataset. Similarly, in Fig. 11 we show the corresponding distribution of house areas.

Curation of benchmark To ensure a robust and representative evaluation dataset, we initially curate a subset of 3,000 episodes from distinct validation houses for each task. A recursive rejection process is then applied, prioritizing tasks with higher uniqueness across dimensions such as object type/synset, lemma, hypernym, and task-specific parameters. For example, in OBJNAVROOM, balancing extends to the target object's room, while ROOMVISIT is balanced on the number of rooms in the house. Manual final filtration of benchmark tasks concludes with average of 195 instances per task, enhancing the reliability of our evaluation framework.

C. Sim-to-real transfer

C.1. Hardware design.

The physical and simulated embodiment of our agent is based on the Hello Robot Stretch RE-1 [37] mobile manipulator. We equip the Stretch with two identical Intel RealSense 455 fixed cameras, namely the *navigation* and the *manipulation* camera, both with a vertical field of view of 59° and capable of 1280×720 RGB-D image capture. The navigation camera is placed looking in the agent's forward direction and points slightly down, with the horizon at a nominal 30°. The manipulation camera is placed 90° in clockwise direction apart from the navigation camera around the vertical axis (*i.e.*, it looks to the right of the robot's forward direction, at the manipulator) and also points slightly down, also with a nominal 30° horizon. A 30° horizon for each camera was chosen to optimize the agent's perspective of its functional working space and potential navigation/manipulation targets. Quirks of fabrication and attachment mean that real horizons may have some

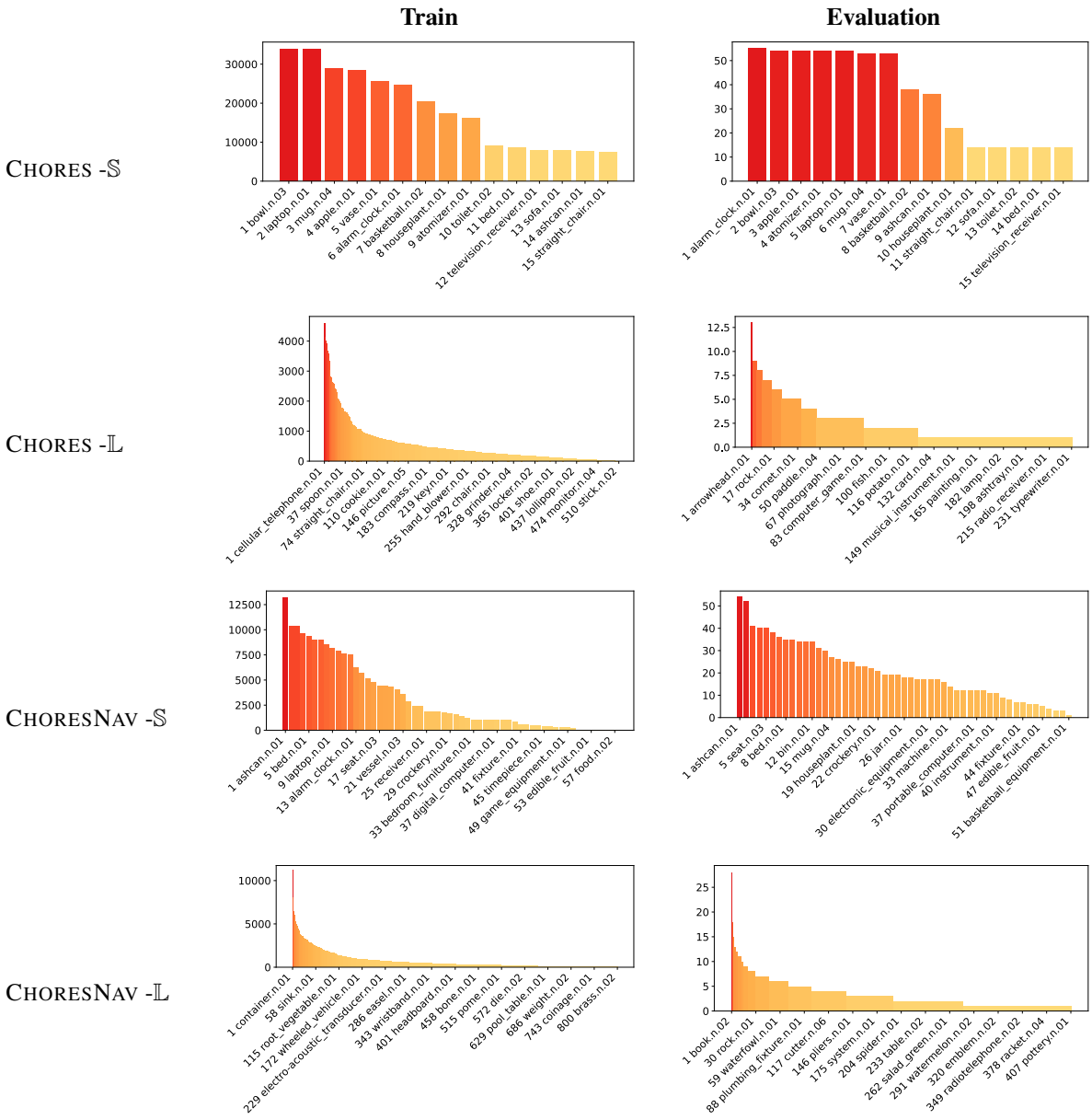


Figure 8. **Frequency of target synsets** for CHORES -𝕊, CHORES -𝕃, CHORESNAV -𝕊, and CHORESNAV -𝕃

variability up to $\pm 3°$. We calibrated all platforms with an Aruco marker setup (shown in Fig. 12) that matched in simulation and real to validate the real horizons were near-nominal and varied the simulated camera horizons in training data generation. The Stretch is equipped with the standard lift and telescoping arm, which allows reaching at heights of up to 110 cm and distances of up to 86.7 cm from the vertical line passing through the manipulation camera's center of projections. The STL files for 3D printing the dual-camera mount will be made available. The mount as designed may be adjusted for horizons in $[0°, 60°]$.

C.2. Sensors.

The primary input is the RGB sensors corresponding to both Intel 455 cameras, which for our purposes to match simulation return resized images of 396×224 pixels. The RGB images are then cropped to 384×224. A binary ObjectInHand sensor is provided by determining if gripper effort (provided by Stretch API) corresponds to a positive grasp with force applied as opposed to an open gripper.

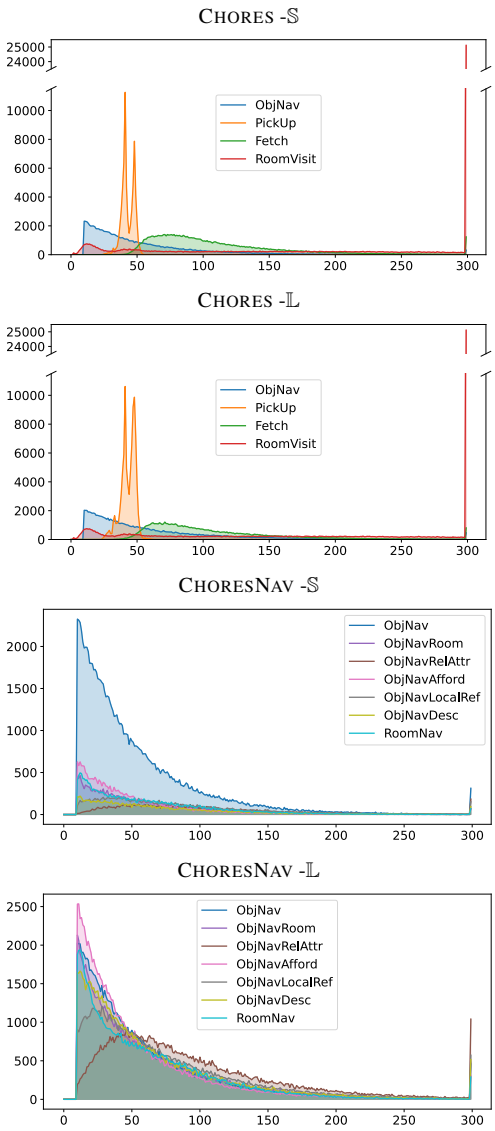


Figure 9. **Distribution of planner trajectory lengths** in training episodes for all task types. *x axis:* trajectory lengths; *y axis:* frequency. For plot legibility, we have clamped all episode lengths to have a max value of 300 in these plots (in training, these episodes can be up to 1000 steps long), this explains the spike at length 300 for some tasks.

C.3. Real Evaluation Environments

We asses the performance of our models on OBJNAV and FETCH in a 6-room apartment also used in Phone2Proc[19], Pickup in RoboThor [16], and Explore 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. These factors, combined with traditional sim-to-real challenges like substantial texture variations and dynamic lighting changes, contribute to a comprehensive evaluation of the proposed method's robustness and generalization capabilities. The environments are showcased in agent trajectory videos highlighted in the supplementary website included along with this PDF.

C.4. Grasping

In the real world we leverage detection and depth for heuristic last-step grasp planning using arm motions alone. If the gripper isn't close enough or grasping would require base motions, the grasp is considered failed. For potentially successful attempts, we use detection with DETIC and instance segmentation from FastSAM [86], aligning the gripper with the projected object center and approaching from above. This approach, adapted for the Stretch RE1's limited dexterity, lacks considerations for object disturbance or protrusions and is susceptible to segmentation or detection issues, impacting overall success. We therefore report manipulation task success based on policy success (object proximity) and full success (both policy and heuristic grasping) in Table 9.

C.5. Real task evaluation details

Object-oriented episodes each have three starting positions which are shared across objects, *e.g.* "navigate to an apple" is evaluated three times: once from the living room, once from the middle of the corridor, and once from the kitchen.

OBJNAV. Target objects are Sofa, Bed, Chair, Apple, Vase, and Houseplant, each from three starting positions.

FETCH. Target objects are Apple, Vase, and Houseplant from the same three starting positions. In one small change from OBJNAV 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.

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. Grasping and success are as described in C.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.

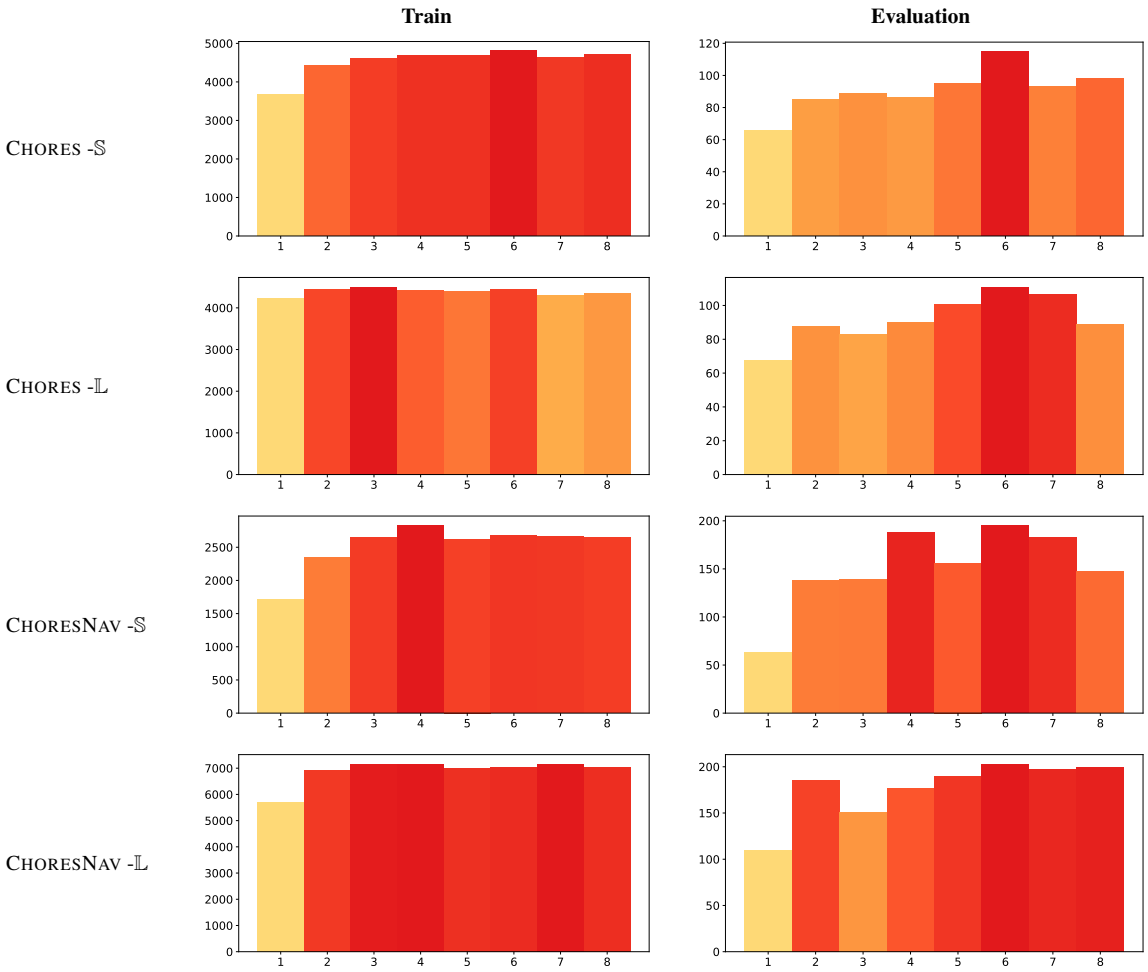


Figure 10. **Distribution of number of rooms** in houses used in CHORES -S, CHORES -L, CHORESNAV -S, and CHORESNAV -L. *x axis:* number of rooms in the house; *y axis:* number of houses

D. Discussion

D.1. Failure Analysis

As highlighted in Lines 448-462 of the main paper, our experiments using ground truth object detection demonstrated a notable improvement of 32% over our RGB-only model. This significant enhancement shows that the majority of the failures is due to perception problems. One promising approach involves training models to utilize the output from off-the-shelf object detection models, such as DETIC [87], as input for object recognition. Our real-world evaluations have shown this method to be effective.

We also hypothesize that pretraining our end-to-end models on tasks that necessitate a more object-aware policy could result in further improvements. This approach could enhance the model's ability to recognize and interact with objects in its environment more effectively.

Additionally, our analysis revealed that in the fetch task, the agent fails to pick up the object 29% of the time when it is in proximity. These failures are primarily due to the robotic arm knocking over the object or missing it due to a slight distance between the grasper and the target object. Improving the precision of the arm's movements and its ability to recover from unsuccessful grasps could significantly enhance performance in such tasks.

Furthermore, within the CHORESNAV tasks, the task involving open vocabulary object descriptions yielded the lowest success rate. Although a success rate of 30.6% is remarkable, we believe that enriching the model's visual and language representations could substantially benefit its performance. These findings suggest valuable directions for future research in improving the effectiveness of robotic task execution.

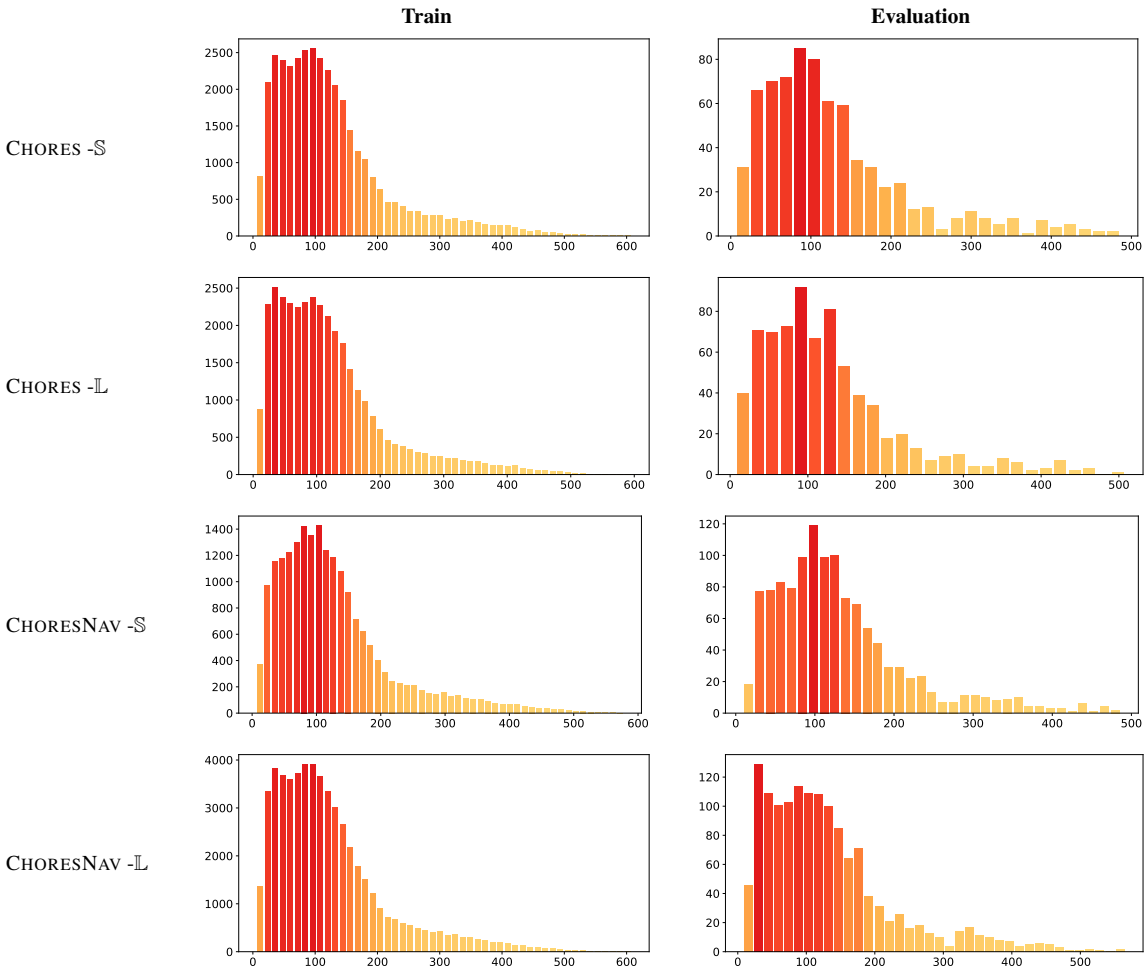
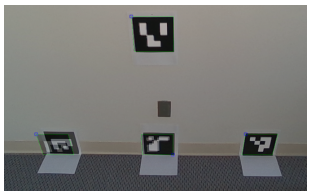
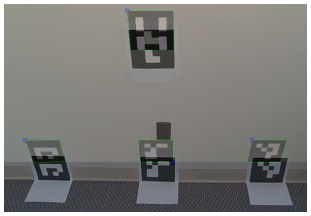


Figure 11. **Distribution of house areas** (in m^2) in CHORES -S, CHORES -L, CHORESNAV -S, and CHORESNAV -L.



(a) Manipulation camera view of calibration setup with best alignment



(b) Manipulation camera view of calibration setup with 3° of misalignment

Figure 12. Simulated and real images of a fixed arrangement of Aruco markers was used to ensure correct camera horizon.

D.2. Limitations

A primary limitation of our study is the reliance on an off-the-shelf grasping planner. Integrating this component into our full end-to-end learning pipeline could potentially enhance the model's efficiency and robustness, particularly in selecting varied grasp positions on objects. Such integration necessitates accommodating physical grasping within the simulation environment.

Another limitation is the robot's operational speed and dexterity. Our project is constrained by the capabilities of commercially available robotic hardware. The development of faster, more affordable, and lighter robots by the broader academic and industrial communities would be greatly beneficial for advancing research in this domain.

These limitations offer avenues for future work and underscore the potential for innovation in robotic systems.

D.3. Why are the shortest path trajectories sufficient?

Our intuition is: the huge diversity of our procedural scenes, large-scale data collection, use of pretrained visual encoders, and extensive visual augmentations enable sufficient state coverage so that generalization is possible. Even given the above, it is surprising that our agent learns to backtrack; we conjecture that our agent acquires a set of skills by imitating expert trajectories. During training, the agent is exposed to trajectories including various skills, such as navigating between rooms, moving towards objects, and avoiding collisions. During evaluation, even if the agent has not observed backtracking behavior, it has encountered all the necessary subskills. As the agent navigates the environment, it switches between different modes (or skills) to complete the task.

D.4. Author Contributions

$^{\Omega}$ Core technical contributors; † Team lead.

Kiana Ehsani.$^{\Omega}$
Contributed to SPOC modeling and training. Led developing expert planners for data generation, hardware calibration and real world evaluation. Trained and evaluated SPOC policies using detection models.

Tanmay Gupta.$^{\Omega}$
Led the development of the SPOC model and the IL training pipeline.

Rose Hendrix.$^{\Omega}$
Contributed to developing expert planners and samplers for data generation, evaluating SPOC in the real world, and refining the CHORES benchmark.

Jordi Salvador.$^{\Omega}$
Contributed to developing expert planners and samplers for data generation.

Luca Weihs.$^{\Omega}$
Designed the distributed data collection pipeline, led annotation and Objaverse↔AI2-THOR integration efforts, contributed to expert planners/samplers, ran experiments.

Kuo-Hao Zeng.$^{\Omega}$
Implemented multi-node training experiments, participated in model development, and implemented RL baselines.

Kunal Pratap Singh.
Contributed to initial stages of data collection, evaluator design, and RL training.

Yejin Kim.
Contributed to real-world experiments and designed the real-world grasping heuristic.

Winson Han.
Provided AI2-THOR integration support.

Alvaro Herrasti.
Designed framework enabling loading optimized Objaverse assets into AI2-THOR at runtime.

Ranjay Krishna.
Brainstorming and high-level discussion/direction.

Dustin Schwenk.
Designed and ran human Objaverse annotation collection via Amazon Mechanical Turk.

Eli VanderBilt.
Created the Objaverse asset optimization pipeline which converts arbitrary input GLB files into a highly optimized format compatible with high-speed rendering.

Aniruddha Kembhavi.†
Led the project. Contributed to model and experimental design. Provided direction on the benchmark and real world evaluation. Wrote the paper.