Habitat 2.0:
Training Home Assistants to Rearrange their Habitat

Andrew Szot², Alex Clegg¹, Eric Undersander¹, Erik Wijmans¹,², Yili Zhao¹, John Turner¹, Noah Maestre¹, Mustafa Mukadam¹, Devendra Chaplot¹, Oleksandr Maksymets¹, Aaron Gokaslan¹, Vladimir Vondrús, Sameer Dharur², Franziska Meier¹, Wojciech Galuba¹, Angel Chang¹, Zsolt Kira², Vladlen Koltun³, Jitendra Malik¹,², Manolis Savva⁴, Dhruv Batra¹,²
¹Facebook AI Research, ²Georgia Tech, ³Intel Research, ⁴Simon Fraser University ⁵UC Berkeley

Abstract

We introduce Habitat 2.0 (H2.0), a simulation platform for training virtual robots in interactive 3D environments and complex physics-enabled scenarios. We make comprehensive contributions to all levels of the embodied AI stack – data, simulation, and benchmark tasks. Specifically, we present: (i) ReplicaCAD: an artist-authored, annotated, reconfigurable 3D dataset of apartments (matching real spaces) with articulated objects (e.g., cabinets and drawers that can open/close); (ii) H2.0: a high-performance physics-enabled 3D simulator with speeds exceeding 25,000 simulation steps per second (850 × real-time) on an 8-GPU node, representing 100 × speed-ups over prior work; and, (iii) Home Assistant Benchmark (HAB): a suite of common tasks for assistive robots (tidy the house, stock groceries, set the table) that test a range of mobile manipulation capabilities. These large-scale engineering contributions allow us to systematically compare deep reinforcement learning (RL) at scale and classical sense-plan-act (SPA) pipelines in long-horizon structured tasks, with an emphasis on generalization to new objects, receptacles, and layouts. We find that (1) flat RL policies struggle on HAB compared to hierarchical ones; (2) a hierarchy with independent skills suffers from ‘hand-off problems’, and (3) SPA pipelines are more brittle than RL policies.

1 Introduction

Consider a home assistant robot illustrated in Fig. 1 – a mobile manipulator (Fetch [1]) performing tasks like stocking groceries into the fridge, clearing the table and putting dishes into the dishwasher, fetching objects on command and putting them back, etc. Developing such embodied intelligent

Figure 1: A mobile manipulator (Fetch robot) simulated in Habitat 2.0 performing rearrangement tasks in a ReplicaCAD apartment – (left) opening a drawer before picking up an item from it, and (right) placing an object into the bowl after navigating to the table. Best viewed in motion at https://sites.google.com/view/habitat2.

*Correspondence to: aszot3@gatech.edu

systems is a goal of deep scientific and societal value. Training and testing such robots in hardware directly is slow, expensive, and difficult to reproduce. We aim to advance the entire ‘research stack’ for developing such embodied agents in simulation – (1) data: curating house-scale interactive 3D assets (e.g. kitchens with cabinets, drawers, fridges that can open/close) that support studying generalization to unseen objects, receptacles, and home layouts, (2) simulation: developing the next generation of high-performance photo-realistic 3D simulators that support rich interactive environments, (3) tasks: setting up challenging representative benchmarks to enable reproducible comparisons and systematic tracking of progress over the years. To support this long-term research agenda, we present:

- **ReplicaCAD**: an artist-authored fully-interactive recreation of ‘FRL-apartment’ spaces from the Replica dataset [2] consisting of 111 unique layouts of a single apartment background with 92 authored objects including dynamic parameters, semantic class and surface annotations, and collision proxies, representing 900+ person-hours of professional 3D artist effort. ReplicaCAD (illustrated in figures and videos) was created with consent, compensation to artists, and will be shared for free under a Creative Commons license for non-commercial use with attribution (CC-BY-NC)

- **Habitat 2.0 (H2.0)**: a high-performance physics-enabled 3D simulator built upon the open-source Habitat-Sim [3] and Bullet physics [4] projects H2.0 supports piecewise-rigid objects (e.g. door, cabinets, and drawers that can rotate about an axis or slide), articulated robots (e.g. mobile manipulators like Fetch [1], fixed-base arms like Franka [5], quadrupeds like AlienGo [6]), and rigid-body mechanics (kinematics and dynamics); and represents approximately 2 years of development effort. The design philosophy of H2.0 is to prioritize performance (or speed) over the breadth of simulation capabilities. H2.0 by design and choice does not support non-rigid dynamics (deformables, fluids, films, cloths, ropes), physical state transformations (cutting, drilling, welding, melting), audio or tactile sensing – many of which are capabilities provided by other simulators [7–9]. The benefit of this focus is that we were able to design and optimize H2.0 to be exceedingly fast – simulating a Fetch robot interacting in ReplicaCAD scenes at 1400 steps per second (SPS), where each ‘step’ involves rendering 1 RGBD observation (128×128 pixels) and simulating rigid-body dynamics for 1/1000 sec. Thus, 30 SPS would be considered ‘real time’ and 1200 SPS is 40× real-time. H2.0 also scales well – achieving 8,200 SPS (273× real-time) multi-process on a single GPU and 26,000 SPS (850× real-time) on a single node with 8 GPUs. For reference, existing simulators typically achieve 10–400 SPS (see Tab. 1). These 100× simulation-speedups correspond to cutting experimentation time from 6 months to under 2 days, unlocking experiments that were hitherto infeasible, allowing us to answer questions that were hitherto unanswerable. As we will show, they also directly translate to training-time speed-up and accuracy improvements from training models (for object rearrangement tasks) with more experience.

- **Home Assistant Benchmark (HAB)**: a suite of common tasks for assistive robots (TidyHouse, PrepareGroceries, SetTable) that are specific instantiations of the generalized rearrangement problem [10]. Specifically, a mobile manipulator (Fetch) is asked to rearrange a list of objects from initial to desired positions – picking/placing objects from receptacles (counter, sink, sofa, table), opening/closing containers (drawers, fridges) as necessary. We use the GeometricGoal specification prescribed by Batra et al. [10] – i.e., initial and desired 3D (center-of-mass) position of each target object \( i \) to be rearranged \( \left( s^i_0, s^i_\ast \right)_{i=1}^N \). The choice of GeometricGoal is deliberate – we aim to create the PointNav [11] equivalent for mobile manipulators. As witnessed in the navigation literature, such a task becomes the testbed for exploring ideas [12–20] and a starting point for more semantic tasks [21–23]. The robot operates entirely from onboard sensing – head- and arm-mounted RGB-D cameras, proprioceptive joint-position sensors (for the arm), and egomotion sensors (for the mobile base) – and may not access any privileged state information (no prebuilt maps, no 3D models of rooms or objects, no physically-implausible sensors providing knowledge of mass, friction, articulation of containers, etc.). Notice that an object’s center-of-mass provides no information about its size or orientation. The target object may be located inside a container (drawer, fridge), on top of supporting surfaces (shelf, table, sofa) of varying heights and sizes, and surrounded by clutter; all of which must be sensed and maneuvered. Receptacles like drawers and fridge start closed, meaning that the agent must open and close articulated objects to succeed. An episode is considered successful if all target objects are placed within 15cm of their desired positions (without considering orientation). The robot uses continuous end-effector control for the arm and velocity control for the base. We deliberately focus on gross motor control (the base and arm) and not fine motor control (the gripper), following the ‘abstracted grasping’ recommendations from [10]. Specifically, once the end-effector reaches 15cm (or closer) to an object, a discrete grasp action becomes available that, if executed, snaps the object
Sensors-to-actions’ policies trained with reinforcement learning (RL) at scale, and classical sense-
What We hope our work will serve as a benchmark for many years to come. H2.0 is free, open-sourced
while Magnum \[ ] indicated by
Table 1: High-level comparison of different simulators. Note: Speeds were taken directly from respective
what a user expects to experience on a single instance of the simulator (no parallelization).
into its parallel-jaw gripper\(^2\). We conduct a systematic study of two distinct techniques – monolithic ‘sensors-to-actions’ policies trained with reinforcement learning (RL) at scale, and classical sense-plan-act pipelines (SPA) [27] – with a particular emphasis on systematic generalization to new objects, receptacles, apartment layouts (not just robot starting pose). Our findings include:

1. Flat vs hierarchical: Monolithic RL policies successfully learn diverse individual skills (pick/place, navigate, open/close drawer). However, crafting a combined reward function and learning scheme that elicits chaining of such skills for the long-horizon HAB tasks remained out of our reach. We saw significantly stronger results with a hierarchical approach that assumes knowledge of a perfect task planner (via STRIPS [28]) to break it down into a sequence of skills.

2. Hierarchy cuts both ways: However, a hierarchy with independent skills suffers from ‘hand-off problems’ where a succeeding skill isn’t set up for success by the preceding one – e.g., navigating to a bad location for a subsequent manipulation, only partially opening a drawer to grab an object inside, or knocking an object out of reach that is later needed.

3. Britteness of SensePlanAct: For simple skills, SPA performs just as well as monolithic RL. However, it is significantly more brittle since it needs to map all obstacles in the workspace for planning. More complex settings involving clutter, challenging receptacles, and imperfect navigation can poorly frame the target object and obstacles in the robot’s camera, leading to incorrect plans.

We hope our work will serve as a benchmark for many years to come. H2.0 is free, open-sourced under the MIT license, and under active development. We believe it will reduce the community’s reliance on commercial lock-ins [29, 30] and non-photorealistic simulation engines [31–33]. Setup and usage instructions for Habitat 2.0 are at https://github.com/facebookresearch/habitat-lab.

2 Related Work

What is a simulator? Abstractly speaking, a simulator has two components: (1) a physics engine that evolves the world state \( s \) over time \( s_0 \rightarrow s_{t+1} \), and (2) a renderer that generates sensor observations \( o \) from states: \( s_t \rightarrow o_t \). The boundary between the two is often blurred as a matter of convenience. Many physics engines implement minimal renderers to visualize results, and some rendering engines include integrations with a physics engine. PyBullet [38], MuJoCo [29], DART [39], ODE [40], PhysX/FleX [41, 42], and Chrono [43] are primarily physics engines with some level of rendering, while Magnum [44], ORRB [45], and PyRender [46] are primarily renderers. Game engines like Unity [47] and Unreal [48] provide tightly coupled integration of physics and rendering. Some simulators [3, 49, 50] involve largely static environments – the agent can move but not change the state of the environment (e.g. open cabinets). Thus, they are heavily invested in rendering with fairly lightweight physics (e.g. collision checking with the agent approximated as a cylinder).

How are interactive simulators built today? Either by relying on game engines [7, 51, 52] or via a ‘homebrew’ integration of existing rendering and physics libraries [8, 35, 37, 53]. Both options have problems. Game engines tend to be optimized for human needs (high image-resolution, \(~60\) FPS, persistent display) not for AI’s needs [54] (10fps, low-res, ‘headless’ deployment on a cluster). Reliance on them leads to limited control over the performance characteristics. On the other

\(^2\)To be clear, H2.0 fully supports the rigid-body mechanics of grasping; the abstract grasping is a task-level simplification that can be trivially undone. Grasping, in-hand manipulation, and goal-directed releasing of a grasp are all challenging open research problems [24–26] that we believe must further mature in the fixed-based close-range setting before being integrated into a long-horizon home-scale rearrangement problem.

\(^3\)Alternatively, \((s_t, a_t) \rightarrow s_{t+1}\) in the presence of an agent taking action \(a_t\).
hand, they represent decades of knowledge and engineering effort whose value cannot be discounted. This is perhaps why ‘homebrew’ efforts involve a high-level (typically Python-based) integration of existing libraries. Unfortunately but understandably, this results in simulation speeds of 10-100s of SPS, which is orders of magnitude sub-optimal. H2.0 involved a deep low-level (C++) integration of rendering (via Magnum [44]) and physics (via Bullet [4]), enabling precise control of scheduling and task-aware optimizations, resulting in substantial performance improvements.

Object rearrangement. Task and motion planning [55] and mobile manipulation have a long history in AI and robotics, whose full survey is beyond the scope of this document. Batra et al. [10] provide a good summary of historical background of rearrangement, a review of recent efforts, a general framework, and a set of recommendations that we adopt here. Broadly speaking, our work is distinguished from prior literature by a combination of the emphasis on visual perception, lack of access to state, systematic generalization, and the experimental setup of visually-complex and ecologically-realistic home-scale environments. We now situate w.r.t. a few recent efforts. [56] study replanning in the presence of partial observability but do not consider mobile manipulation. [53] tackle ‘interactive navigation’, where the robot can bump into and push objects during navigation, but does not have an arm. Some works [57–60] abstract away gross motor control entirely by using symbolic interaction capabilities (e.g. a ‘pick up X’ action) or a ‘magic pointer’ [10]. We use abstracted grasping but not abstract manipulation; in our tasks agents use a continuous action space for the arm and base. [20] develop hierarchical methods for mobile manipulation, combining RL policies for goal-generation and motion-planning for executing them. We use the combination of planning and learning – using task-planning to generate goals and RL for skills. ManipulaTHOR [34] is perhaps the most similar to our work. Their task involves moving a single object from one location to another, excluding interactions with container objects (opening a drawer or fridge to place an object inside). We will see that rearrangement of multiple objects while handling containment is a much more challenging task. Interestingly, our experiments show evidence for the opposite conclusion reached therein – monolithic end-to-end trained RL methods are outperformed by a modular approach that is trained stage-wise to handle long-horizon rearrangement tasks.

3 Replica to ReplicaCAD: Creating Interactive Digital Twins of Real Spaces

We begin by describing our dataset that provides a rich set of indoor layouts for studying rearrangement tasks. Our starting point was Replica [2], a dataset of highly photo-realistic 3D reconstructions at room and building scale. Unfortunately, static 3D scans are unsuitable for studying rearrangement tasks because objects in a static scan cannot be moved or manipulated.

Figure 2: Left: The original Replica scene. Right: the artist recreated scene ReplicaCAD. All objects (furniture, mugs) including articulated ones (drawers, fridge) in ReplicaCAD are fully physically simulated and interactive.

Asset Creation. ReplicaCAD is an artist-created, fully-interactive recreation of ‘FRL-apartment’ spaces from the Replica dataset [2]. First, a team of 3D artists authored individual 3D models (geometry, textures, and material specifications) to faithfully recreate nearly all objects (furniture, kitchen utensils, books, etc.; 92 in total) in all 6 rooms from the FRL-apartment spaces as well as an accompanying static backdrop (floor and walls). ReplicaCAD like other efforts such as OpenRooms [61] and Scan2cad [62] creates a version of the scene with decoupled 3D models suited for physical simulation and interaction. Fig. 2 compares a layout of ReplicaCAD with the original Replica scan. Next, each object was prepared for rigid-body simulation by authoring physical parameters (mass, friction, restitution), collision proxy shapes, and semantic annotations. Several objects (e.g. refrigerator, kitchen counter) were made ‘articulated’ through sub-part segmentation (annotating fridge door, counter cabinet) and authoring of URDF files describing joint configurations.
(e.g. fridge door swings around a hinge) and dynamic properties (e.g. joint type and limits). For each large furniture object (e.g. table), we annotated surface regions (e.g. table tops) and containment volumes (e.g. drawer space) to enable programmatic placement of small objects on top of or within.

**Human Layout Generation.** Next, a 3D artist authored an additional 5 semantically plausible ‘macro variations’ of the scenes – producing new scene layouts consisting only of larger furniture from the same 3D object assets. Each of these macro variations was further perturbed through 20 ‘micro variations’ that re-positioned objects – e.g. swapping the locations of similarly sized tables or a sofa and two chairs. This resulted in a total of 105 scene layouts that exhibit major and minor semantically-meaningful variations in furniture placement and scene layout, enabling controlled testing of generalization. Illustrations of these variations can be found in Appendix A.

**Procedural Clutter Generation.** To maximize the value of the human-authored assets we also develop a pipeline that allows us to generate new clutter procedurally. Specifically, we dynamically populate the annotated supporting surfaces (e.g. table-top, shelves in a cabinet) and containment volumes (e.g. fridge interior, drawer spaces) with object instances from appropriate categories (e.g., plates, food items). These inserted objects can come from ReplicaCAD or the YCB dataset [63]. We compute physically-stable insertions of clutter offline (i.e., letting an inserted bowl ‘settle’ on a shelf) and then load these stable arrangements into the scene dynamically at run-time.

ReplicaCAD is fully integrated with the H2.0 and a supporting configuration file structure enables simple import, instancing, and programmatic alternation of any of these interactive scenes. Overall, ReplicaCAD represents 900+ person-hours of professional 3D artist effort so far (with augmentations in progress). It was created with consent, compensation to artists, and will be shared for free under a Creative Commons license for non-commercial use with attribution (CC-BY-NC) Further ReplicaCAD details, statistics, and comparison to prior work are in Appendix A.

# 4 Habitat 2.0 (H2.0): a Lazy Simulator

H2.0’s design philosophy is that performance (or speed) is more important than the breadth of simulation capabilities. H2.0 achieves fast physical simulations in large realistic 3D indoor scenes by being lazy – following the principle of only simulating what is absolutely needed. We instantiate this principle via three key ideas – localized physics and rendering (Sec. 4.1), interleaved physics and rendering (Sec. 4.2), and simplify-and-reuse (Appendix B.1). H2.0 is built using a tight integration of rendering from Habitat-Sim [3] and physics from the C++ Bullet library [4]. Further details, e.g. integration with motion planning libraries, are described in Appendix B.

## 4.1 Localized Physics and Rendering

Realistic indoor 3D scenes can span houses with multiple rooms (kitchen, living room), hundreds of objects (sofa, table, mug) and ‘containers’ (fridge, drawer, cabinet), and thousands of parts (fridge shelf, cabinet door). Simulating physics for every part at all times is slow and unnecessary. We leverage Bullet’s built-in island sleep system to minimize simulation overhead for idle objects. In addition, we make several optimizations: (1) We employ a navigation mesh to move the robot base kinematically (which has been show to transfer well to real the world [64]) rather than simulating wheel-ground contact. (2) For multi-body articulated furniture, we remove static parts (e.g. the walls and floor of a cabinet) from the Bullet multi-body and instead load these as separate static rigid objects. This improves the sleeping behavior of the entire simulation, for example, an idle object resting on the floor of the cabinet can sleep even while the cabinet door is moving. (3) We use the sleeping state of objects to optimize rendering by caching and re-using scene graph transformation matrices and frustum-culling results.

## 4.2 Interleaved rendering and physics

Most physics engines (e.g. Bullet) run on the CPU, while rendering (e.g. via Magnum) typically occurs on the GPU. After our initial optimizations, we found each to take nearly equal compute-time. This represents a glaring inefficiency – as illustrated in Fig. 3, at any given time either the CPU is sitting idle waiting for the GPU or vice-versa. Thus, interleaving them leads to significant gains. However, this is complicated by a sequential dependency – state transitions depend on robot actions $T: (s_t, a_t) \rightarrow s_{t+1}$, robot actions depend on the sensor

![Sequential vs. Interleaved Physics](image)

*Figure 3: Interleaved physics and rendering in H2.0.*
We benchmark using a Fetch robot, equipped with two RGB-D cameras (128 × 128 pixels) in ReplicaCAD scenes under two scenarios: (1) Idle: with the robot initialized in the center of the living room somewhat far from furniture or any other object and taking random actions, and (2) Interact: with the robot initialized fairly close to the fridge and taking actions from a pre-computed trajectory that results in representative interaction with objects. Each simulation step consists of 1 rendering pass and 4 physics-steps, each simulating 1/120 sec for a total of 1/30 sec. New joint position goals are set every 1/30 sec and a joint controller computes the joint torques to achieve the joint goals for the current joint state every 1/120 sec. This is a fairly standard experimental configuration in robotics (with 30 FPS cameras and 120 Hz control). In this setting, a simulator operating at 30 steps per (wallclock) second (SPS) corresponds to ‘real time’.

Benchmarking was done on machines with dual Intel Xeon Gold 6226R CPUs – 32 cores/64 threads (32C/64T) total – and 8 NVIDIA GeForce 2080 Ti GPUs. For single-GPU benchmarking processes are confined to 8C/16T of one CPU, simulating an 8C/16T single GPU workstation. For single-GPU multi-process benchmarking, 16 processes were used. For multi-GPU benchmarking, 64 processes were used with 8 processes assigned to each GPU. We used python-3.8 and gcc-9.3 for compiling with the robot initialized fairly close to the fridge and taking actions from a pre-computed trajectory (Bullet). We can clearly see the benefit of working with the low-level C++ Bullet rather than PyBullet (32C/64T) total – and 8 NVIDIA GeForce 2080 Ti GPUs. For single-GPU benchmarking processes are confined to 8C/16T of one CPU, simulating an 8C/16T single GPU workstation. For single-GPU multi-process benchmarking, 16 processes were used. For multi-GPU benchmarking, 64 processes were used with 8 processes assigned to each GPU. We used python-3.8 and gcc-9.3 for compiling with 30 FPS cameras and 120 Hz control. In this setting, a simulator operating at 30 steps per (wallclock) second (SPS) corresponds to ‘real time’.

### Table 2: Benchmarking H2.0 performance: simulation steps per second (SPS, higher better) over 10 runs and a 95% confidence-interval computed via standard error of the mean. Note that 8 processes do not fully utilize a 2080 Ti and thus multi-process multi-GPU performance may be better on machines with more CPU cores.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>1 Process</th>
<th>1 GPU</th>
<th>8 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
<td>Interact</td>
<td>Idle</td>
</tr>
<tr>
<td>H2.0 (Full)</td>
<td>1191 ±36</td>
<td>510 ±6</td>
<td>8186 ±47</td>
</tr>
<tr>
<td>- render opts.</td>
<td>781 ±9</td>
<td>282 ±2</td>
<td>6709 ±89</td>
</tr>
<tr>
<td>- physics opts.</td>
<td>271 ±3</td>
<td>358 ±6</td>
<td>2290 ±5</td>
</tr>
<tr>
<td>- all opts.</td>
<td>242 ±2</td>
<td>224 ±3</td>
<td>2223 ±3</td>
</tr>
</tbody>
</table>

Table 2 reports benchmarking numbers for H2.0. We make a few observations. The ablations for H2.0 (denoted by ‘- render opts’, ‘-physics opts’, and ‘-all opts.’) show that principles followed in our system design lead to significant performance improvements.

Our ‘Idle’ setting is similar to the benchmarking setup of iGibson [37], which reports 100 SPS. In contrast, H2.0 single-process with all optimizations turned off is 240% faster (242 vs 100 SPS). H2.0 single-process with optimizations on is ~1200% faster than iGibson (1191 vs 100 SPS). The comparison to iGibson is particularly illustrative since it uses the ‘same’ physics engine (PyBullet) as H2.0 (Bullet). We can clearly see the benefit of working with the low-level C++ Bullet rather than PyBullet and the deep integration between rendering and physics. This required deep technical expertise and
large-scale engineering over a period of 2 years. Fortunately, H2.0 will be publicly available so others do not have to repeat this work. A direct comparison against other simulators is not feasible due to different capabilities, assets, hardware, and experimental settings. But a qualitative order-of-magnitude survey is illustrative – A12-THOR [7] achieves 60/30 SPS in idle/interact, SAPIEN [35] achieves 200/400 SPS (personal communication), TDW [8] achieves 5 SPS in interact, and RLBenchmark [36] achieves between 1 and 60 SPS depending on the sensor suite (personal communication). Finally, H2.0 scales well – achieving 8,186 SPS (272× real-time) multi-process on a single GPU and 25,734 SPS (850× real-time) on a single node with 8 GPUs. These 100× simulation-speedups correspond to cutting experimentation time from 6-month cycle to under 2 days.

5 The Pick Task: a Base Case of Rearrangement

We first carry out systematic analyses on a relatively simple robotic manipulation task: picking up one object from a cluttered ‘receptacle’. This forms a ‘base case’ and an instructive starting point that we eventually expand to the more challenging Home Assistant Benchmark (HAB) (Sec. 6).

Task Definition: Pick ($s^0$). Fig. 4 illustrates an episode in the pick task. Our agent (a Fetch robot [1]) is spawned close to a receptacle (a table) that holds multiple objects (e.g. cracker box, bowl). The task for the robot is to pick up a target object with center-of-mass coordinates $s^0 \in \mathbb{R}^3$ (provided in robot’s coordinate system) as efficiently as possible without excessive collisions. We study systematic generalization to new clutter layouts on the receptacle, to new objects, and to new receptacles.

Agent embodiment and sensing. Fetch [1] is a wheeled base with a 7-DoF arm manipulator and a parallel-jaw gripper, equipped with two RGBD cameras (128×128-resolution 90 degree field of view) mounted on its ‘head’ and arm. The agent can sense its proprioceptive-state – arm joint angles (7-dim), end-effector position (3-dim), and base-egomotion (6-dim, also known as GPS+Compass in the navigation literature [3]). Note: the episodes in Pick are constructed such that the robot does not need to move its base. Thus, the egomotion sensor does not play a role in Pick but will be important in HAB tasks (Sec. 6).

Action space: gross motor control. The agent policy performs end-effector control at 30Hz. Specifically, it outputs the desired change in end-effector position ($\delta x, \delta y, \delta z$), which is fed into an inverse kinematics solver to derive desired states for all joints, which are used to set the desired joint-motor positions, which are achieved using PD control (details in Appendix C.3). In Pick, the base is fixed but in HAB, the agent controls the base by emitting linear and angular velocities ($v, w$).

Abstracted grasping. Once the robot’s end-effector reaches 15cm (or closer) to an object, a discrete grasp action becomes available that, if executed, snaps the object closest to the end-effector into the parallel-jaw gripper. The grasping is perfect and objects do not slide out. If an object is currently held in the gripper, calling the grasp action releases it and the object is physically simulated as falling. This follows the ‘abstracted grasping’ recommendations in Batra et al. [10] and is consistent with recent work [34]. For more details on the agent setup and results for other action spaces see Appendix E.7.

Evaluation. An object is considered successfully picked if the arm returns to a known ‘resting position’ with the target object grasped. An episode fails if the accumulated contact force experienced by the arm/body exceeds a threshold of 5k Newtons. Further details are provided in Appendix C.2.

Methods. We compare two methods representing two distinctive approaches to this problem:
1. MonolithicRL: a ‘sensors-to-actions’ policy trained end-to-end with reinforcement learning (RL). The visual input is encoded using a CNN, concatenated with embeddings of proprioceptive-sensing and goal coordinates, and fed to a recurrent actor-critic network, trained with DD-PPO [12] for 100 Million steps of experience (see Appendix C.4 for details). The actor outputs three continuous variables (change in end-effector position $\delta x, \delta y, \delta z$) and one discrete variable (grasp/release). This baseline translates our community’s most-successful paradigm yet from navigation to manipulation.
2. SensePlanAct (SPA) pipeline: Sensing consists of constructing an accumulative 3D point-cloud of the scene from depth sensors, which is then used for collision queries. Motion planning is
We now describe our benchmark of common household assistive robotic tasks. We stress that these
we find that the Depth sensor is the most important for policy performance. Furthermore, blind
Task Definition. With H2.0 we can compare how learning based systems generalize compared to SPA architectures. Tab. 3 shows the results of a systematic generalization study of 4 unseen objects, 3 unseen receptacles, and 20 unseen apartment layouts (from 1 unseen ‘macro variation’ in Replica-CAD). Full experiment details in Appendix C.1.

MonolithicRL generalizes fairly well from seen to unseen layouts (91.7 → 86.3%), significantly outperforming SPA (72.7%) and even SPA-Priv (80.0%). However, generalization to new objects is intriguing (91.7 → 74.7%) as a result of the new visual feature distribution and new object obstacles. Generalization to new receptacles is poor (91.7 → 52.7%), however, as highlighted by the performance drop of SPA (and corroborated by qualitative results), the unseen receptacles (shelf, armchair, dark table) appear to be objectively more difficult to pick up objects from since the shelf and armchair are tight constrained areas whereas the majority of the training receptacles, such as counters and tables, have no such constraints. (see Figure 9). We believe the performance of MonolithicRL will naturally improve as more 3D assets for receptacles become available; we cannot make any such claims for SPA.

We also use H2.0 to analyze sensor trade-offs at scale (100M steps of training). In Appendix E.2 we find that the Depth sensor is the most important for policy performance. Furthermore, blind agents operating from proprioceptive-state only perform surprisingly well by learning to ‘feel its way’ towards the goal by slowly moving the arm to not incur large collision forces. However, we find that sighted policies are more efficient, better avoid collisions, and achieve higher success as shown in Appendix E.3. We also analyze different camera placements on the Fetch robot in Appendix E.4 and find placing cameras on the arm to be most effective. We speculate this is due to self-occlusion from the head camera and strong ‘perception-action coupling’ with the arm-camera. For further analysis experiments, see Appendix E.7 for a comparison of different action spaces, Appendix E.6 for the effect of the time delay on performance, and Appendix E.5 for qualitative evidence of self-tracking.

6 Home Assistant Benchmark (HAB)
We now describe our benchmark of common household assistive robotic tasks. We stress that these tasks illustrate the capabilities of H2.0 but do not delineate them — a lot more is possible but not feasible to pack into a single coherent document with clear scientific takeaways.

Task Definition. We study three (families of) long-range tasks that correspond to common activities:

1. TidyHouse: Move 5 objects from random (unimpeded) locations back to where they belong (see Fig. 19). This task requires no opening or closing and no objects are contained.
   • Start: 5 target objects objects spawned in 6 possible receptacles (excluding fridge and drawer).
   • Goal: Each target object is assigned a goal in a different receptacle than the starting receptacle.
   • Task length: 5000 steps.

2. PrepareGroceries: Remove 2 objects from the fridge to the counters and place one object back in the fridge (see Fig. 20). This task requires no opening or closing and no objects are contained.
   • Start: 2 target objects in the fridge and one on the left counter. The fridge is fully opened.
   • Goal: The goal for the target objects in the fridge are on the right counter and light table. The goal for the other target object is in the fridge.
   • Task length: 4000 steps.

3. SetTable: Get a bowl from a drawer, a fruit from fridge, place the fruit in the bowl on the table (see Fig. 21).
   • Start: A target bowl object is in one of the drawers and a target fruit object in the middle fridge shelf. Both the fridge and drawer start closed.
   • Goal: The goal for the bowl is on the light table, the goal for the fruit is on top of the bowl. Both the fridge and drawer must be closed.
• Task length: 4500 steps.

The list is in increasing order of complexity – from no interaction with containers (TidyHouse), to picking and placing from the fridge container (PrepareGroceries), to opening and closing containers (SetTable). Note that these descriptions are provided purely for human understanding; the robot operates entirely from a GeometricGoal specification [16] – given by the initial and desired 3D (center-of-mass) position of each target object $i$ to be moved $(s^0_i, s^*_i)_{i=1}^N$. Thus, Pick $(s^0_i)$ is a special case where $N = 1$ and $s^*_i$ is a constant (arm resting) location. For each task episode, we sample a ReplicaCAD layout with YCB [63] objects randomly placed on feasible placement regions (see procedural clutter generation in Section 3). Each task has 5 clutter objects per receptacle. Unless specified, objects are sampled from the ‘food’ and ‘kitchen’ YCB item categories in the YCB dataset.

The agent is evaluated on unseen layouts and configurations of objects, and so cannot simply memorize. We characterize task difficulty by the required number of rigid-body transitions (e.g., picking up a bowl, opening a drawer). The task evaluation, agent embodiment, sensing, and action space remain unchanged from Section 3, with the addition of base control via velocity commands. Further details on the statistics of the rearrangement episodes, as well as the evaluation protocols are in Appendix F.

Methods. We extend the methods from Sec. 5 to better handle the above long-horizon tasks with a high-level STRIPS planner using a parameterized set of skills: Pick, Place, Open/Close fridge door, Open/Close drawer, and Navigate. The full details of the planner implementation and how methods are extended are in Appendix G. Here, we provide a brief overview.

1. **MonolithicRL**: Essentially unchanged from Sec. 5, with the exception of accepting a list of start and goal coordinates $(s^0_i, s^*_i)_{i=1}^N$, as opposed to just $s^0_i$.

2. **TaskPlanning+SkillsRL (TP+SPRL)**: a hierarchical approach that assumes knowledge of a perfect task planner (implemented with STRIPS [28]) and the initial object containment needed by the task planner to break down a task into a sequence of parameterized skills: Navigate, Pick, Place, Open/Close fridge door, Open/Close drawer. Each skill is functionally identical to MonolithicRL in Sec. 5 – taking as input a single 3D position, either $s^0_i$ or $s^*_i$. For instance, in the SetTable task, let $(a^0, a^*)$ and $(b^0, b^*)$ denote the start and goal positions of the apple and bowl, respectively. The task planner converts this task into:

<table>
<thead>
<tr>
<th>Open Drawer</th>
<th>Transport Bowl</th>
<th>Close Drawer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigate($a^0$), Open drawer($b^0$), Pick($b^0$), Navigate($b^<em>$), Place($b^</em>$), Navigate($b^0$), Close drawer($b^0$), Transport Apple</td>
<td>Close Fridge</td>
<td>Close Fridge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open Fridge</th>
<th>Transport Apple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigate($a^0$), Open fridge door($a^0$), Navigate($a^<em>$), Place($a^</em>$), Navigate($a^0$), Close fridge door($a^0$)</td>
<td>Close Fridge</td>
</tr>
</tbody>
</table>

Simply listing out this sequence highlights the challenging nature of these tasks.

3. **TaskPlanning+SensePlanAct (TP+SPA)**: Same task planner as above, with each skill implemented via SPA from Sec. 5 except for Navigate where the same learned navigation policy from TP+SPA is used. TP+SPA-Priv is analogously defined. Crafting an SPA pipeline for opening/closing unknown articulated containers is an open unsolved problem in robotics – involving detecting and tracking articulation [70, 71] without models, constrained full-body planning [72–74] without hand engineering constraints, and designing controllers to handle continuous contact [75, 76] – making it out of scope for this work. Thus, we do not report TP+SPA on SetTable.

### 6.1 Results and Findings

Figure 5 shows progressive success rates for different methods on all tasks. Due to the difficulty of the full task, for analysis, the X-axis lists the sequence of agent-environment interactions (pick, place, open, close) required to accomplish the task, same as that used by the task-planner. The number of interactions is a proxy for task difficulty and the plot is analogous to precision-recall curves (with the ideal curve being a straight line at 100%). Furthermore, since navigation is often executed between successive skills, we include versions of the task planning methods with an oracle navigation skill. We make the following observations:

1. **MonolithicRL** performs abysmally. We were able to train individual skills with RL to reasonable degrees of success (see Appendix H.2). However, crafting a combined reward function and learning

---

4This sequence from the task plan is useful for experimental analysis and debugging, but does not represent the only way to solve the task and should be disposed in future once methods improve on the full task.
Figure 5: Success rates for Home Assistant Benchmark tasks. Due to the difficulty of full HAB tasks, we analyze performance as completing a part of the overall task. For the TP methods that use an explicit navigation skill, we indicate with an arrow in the interaction names where navigation occurs and include versions for learned and oracle navigation. Results are on unseen layouts with mean and standard error computed for 100 episodes.

scheme that elicits chaining of such skills for a long-horizon task, without any architectural inductive bias about the task structure, remained out of our reach despite prolonged effort.

2. Learning a navigation policy to chain together skills is challenging as illustrated by the performance drop between learned and oracle navigation. In navigation for the sake of navigation (PointNav [11]), the agent is provided coordinates of the reachable goal location. In navigation for manipulation (Navigate), the agent is provided coordinates of a target object’s center-of-mass but needs to navigate to an unspecified non-unique suitable location from where the object is manipulable.

3. Compounding errors hurt performance of task planning methods. Even with the relatively easier skills in TidyHouse in Figure 5a all methods with oracle navigation gradually decrease in performance as the number of required interactions increases.

4. Sense-plan-act variants scale poorly to increasing task complexity. In the easiest setting, Tidy House with oracle navigation (Figure 5a), TP+SPA performs better than TP+SRL. However, this trend is reversed with learned navigation since TP+SPA methods, which rely on egocentric perception for planning, are not necessarily correctly positioned to sense the workspace. In the more complex task of PrepareGroceries (Figure 5b), TP+SRL outperforms TP+SPA both with and without oracle navigation due to the perception challenge of the tight and cluttered fridge. TP+SPA fails to find a goal configuration 3x more often and fails to find a plan in the allowed time 3x more often in PrepareGroceries than TidyHouse.

See Appendix H for individual skill success rates, learning curves, and SPA failure statistics.

7 Societal Impacts, Limitations, and Conclusion

ReplicaCAD was modeled upon apartments in one country (USA). Different cultures and regions may have different layouts of furniture, types of furniture, and types of objects not represented in ReplicaCAD; and this lack of representation can have negative social implications for the assistants developed. While H2.0 is a fast simulator, we find that the performance of the overall simulation+training loop is bottlenecked by factors like synchronization of parallel environments and reloading of assets upon episode reset. An exciting and complementary future direction is holistically reorganizing the rendering+physics+RL interplay as studied by [77–82]. Concretely, as illustrated in Figure 3, there is idle GPU time when rendering is faster than physics, because inference waits for both $o_t$ and $s_{t+1}$ to be ready despite not needing $s_{t+1}$. This is done to maintain compatibility with existing RL training systems, which expect the reward $r_t$ to be returned when the agent takes an action $a_t$, but $r_t$ is typically a function of $s_t$, $a_t$, and $s_{t+1}$. Holistically reorganizing the rendering+physics+RL interplay is an exciting open problem for future work.

We presented the ReplicaCAD dataset, the Habitat 2.0 platform and a home assistant benchmark. H2.0 is a fully interactive, high-performance 3D simulator that enables efficient experimentation involving embodied AI agents rearranging richly interactive 3D environments. Coupled with the ReplicaCAD data these improvements allow us to investigate the performance of RL policies against classical MP approaches for the suite of challenging rearrangement tasks we defined. We hope that the Habitat 2.0 platform will catalyze work on embodied AI for interactive environments.
8 Acknowledgements

Funding in direct support of this work: The Georgia Tech portion of this work supported by state funds from Georgia Tech (AS, ZK), NSF (DB), AFRL (DB), DARPA (DB), ONR YIPs (DB), ARO PECASE (DB), Amazon (DB). The SFU portion of this work is supported by a CIFAR AI Chair (AC), a Canada Research Chair (MS), and NSERC Discovery Grants (AC,MS). This work was also supported by FAIR (AC, EU, YZ, JT, NM, MM, DC, OM, AG, FM, WG) and Intel (VK).

Additional revenues related to this work: NSF (ZK), DARPA (ZK), ONR (ZK), NGA (ZK), Samsung (ZK), Airbus (ZK), consulting for Marble Inc (ZK), paid talk by Data Science Connect (ZK), sponsored research funding by FAIR (AC,MS).

References


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See second paragraph of Sec. 7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the first paragraph of Sec. 7.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and setup instructions can be found at https://github.com/facebookresearch/habitat-lab.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All methods and training details are described in detail in Sec. G.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Results in Sec. 5 are over three random seeds and 600 episodes each. Results in Sec. E.2 and Sec. E.4 are over 10 random seeds and 500 episodes each. Finally, results in Sec. 6.1 are over 100 episodes.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Compute resources for the benchmark are described in Section 4, for RL training in Appendix C.4.2, and for motion planning in Appendix D.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Cited [63] for use of the YCB objects.
   (b) Did you mention the license of the assets? [Yes] ReplicaCAD is released under the Creative Commons license and H2.0 is open-sourced under the MIT license
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The ReplicaCAD dataset will be made publicly available for free prior to publication under the Creative Commons license. Download instructions can be found at https://github.com/facebookresearch/habitat-lab.
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The released ReplicaCAD dataset includes furniture layouts and common kitchen items. There is no identifiable or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]