Habitat: A Platform for Embodied AI Research

Manolis Savva[1,4]*, **Abhishek Kadian**[1]*, **Oleksandr Maksymets**[1]*, **Yili Zhao**[1], **Erik Wijmans**[1,2,3], **Bhavana Jain**[1], **Julian Straub**[2], **Jia Liu**[1], **Vladlen Koltun**[5], **Jitendra Malik**[1,6], **Devi Parikh**[1,3], **Dhruv Batra**[1,3]

[1]Facebook AI Research, [2]Facebook Reality Labs, [3]Georgia Institute of Technology, [4]Simon Fraser University, [5]Intel Labs, [6]UC Berkeley

https://aihabitat.org

Abstract

We present Habitat, a new platform for research in embodied artificial intelligence (AI). Habitat enables training embodied agents (virtual robots) in highly efficient photorealistic 3D simulation, before transferring the learned skills to reality. Specifically, Habitat consists of the following –

1. `Habitat-Sim`*: a flexible, high-performance 3D simulator with configurable agents, multiple sensors, and generic 3D dataset handling (with built-in support for SUNCG, Matterport3D, Gibson datasets).* `Habitat-Sim` *is fast: when rendering a scene from the Matterport3D dataset,* `Habitat-Sim` *achieves several thousand frames per second (fps) running single-threaded, and can reach over* 10,000 *fps multi-process on a single GPU, which is orders of magnitude faster than the closest simulator.*

2. `Habitat-API`*: a modular high-level library for end-to-end development of embodied AI algorithms – defining embodied AI tasks (*e.g. *navigation, instruction following, question answering), configuring and training embodied agents (via imitation or reinforcement learning, or via classic SLAM), and benchmarking using standard metrics [1].*

These large-scale engineering contributions enable us to answer scientific questions requiring experiments that were till now impracticable or 'merely' impractical. Specifically, in the context of point-goal navigation (1) we revisit the comparison between learning and SLAM approaches from two recent works [19, 15] and find evidence for the **opposite conclusion** *– that learning outperforms SLAM if scaled to an order of magnitude more experience than previous investigations, and (2) we conduct the first cross-dataset generalization experiments {train, test} × {SUNCG, Matterport3D, Gibson} for multiple sensors {blind, RGB, RGBD, D} and find that only agents with depth (D) sensors generalize across datasets. We hope that our open-source platform and these findings will advance research in embodied AI.*

*Denotes equal contribution.

1. Introduction

Imagine walking up to a home robot and asking *'Hey – can you go check if my laptop is on my desk? And if so, bring it to me.'* In order to be successful, such a robot would need a range of skills – visual perception (to recognize scenes and objects), language understanding (to translate questions and instructions into actions), and navigation in complex environments (to move and find things in a changing environment).

While there has been significant progress in the vision and language communities thanks to recent advances in deep representations [13, 10], much of this progress has been on 'internet AI' rather than embodied AI. The focus of the former is pattern recognition in images, videos, and text on *datasets* typically curated from the internet [9, 17, 3]. The focus of the latter is to enable action by an embodied agent (*e.g.* a robot) in an *environment*. This brings to the fore active perception, long-term planning, learning from interaction, and holding a dialog grounded in an environment.

A straightforward proposal is to train agents directly in the physical world – exposing them to all its richness. This is valuable and will continue to play an important role in the development of AI. However, we also recognize that training robots in the real world is *slow* (the real world runs no faster than real time and cannot be parallelized), *dangerous* (poorly-trained agents can unwittingly injure themselves, the environment, or others), *expensive* (the robot(s) and the environment(s) in which they execute cost money and time), *difficult to control* (it is hard to test corner-case scenarios as these are, by definition, infrequent and challenging/expensive to recreate), and *not easily reproducible* (replicating conditions across experiments and institutions is difficult).

We aim to support a complementary research program: training embodied agents in rich realistic simulators and then transferring the learned skills to reality. Simulations have a long and rich history in science and engineering (from aerospace to zoology). In the context of embodied AI, simulators help overcome the aforementioned challenges. Simulators can run orders of magnitude faster than real-

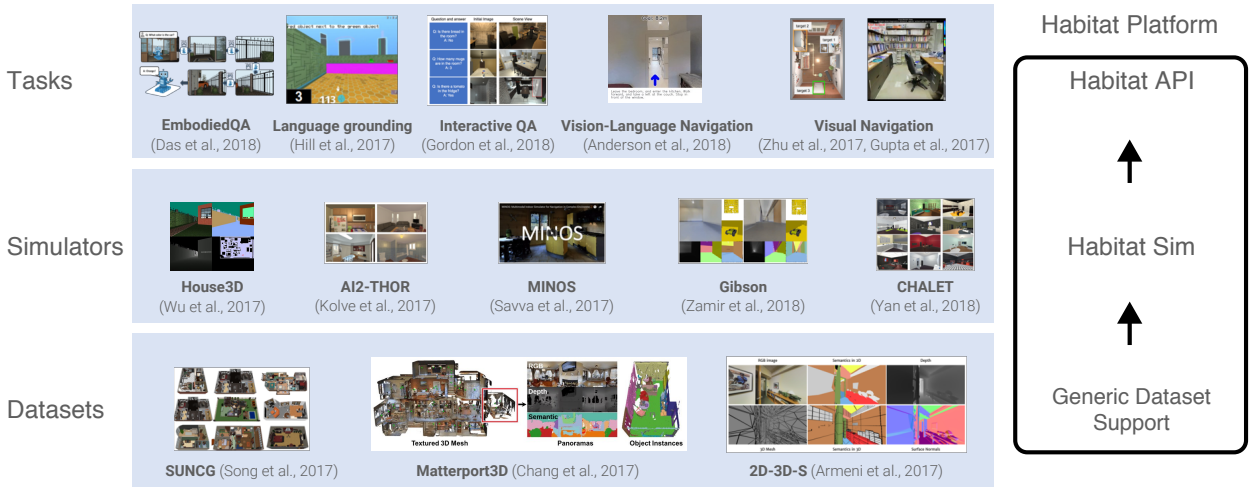


Figure 1: The 'software stack' for training embodied agents involves (1) *datasets* providing 3D assets with semantic annotations, (2) *simulators* that render these assets and within which an embodied agent may be simulated, and (3) *tasks* that define evaluatable problems that enable us to benchmark scientific progress. Prior work (highlighted in blue boxes) has contributed a variety of datasets, simulation software, and task definitions. We propose a unified embodied agent stack with the Habitat platform, including generic dataset support, a highly performant simulator (`Habitat-Sim`), and a flexible API (`Habitat-API`) allowing the definition and evaluation of a broad set of tasks.

time and can be parallelized over a cluster. Training in simulation is safe and cheap, and enables fair comparison and benchmarking of progress in a concerted community-wide effort. Once a promising approach has been developed and tested in simulation, it can be transferred to physical platforms that operate in the real world [5, 14].

Datasets have been a key driver of progress in computer vision, NLP, and other areas of AI [9, 17, 3]. As the community transitions to embodied AI, we believe that simulators will assume the role played previously by datasets. To support this transition, we aim to standardize the entire 'software stack' for training embodied agents (Figure 1): scanning the world and creating photorealistic 3D assets, developing the next generation of highly efficient and parallelizable simulators, specifying embodied AI tasks that enable us to benchmark scientific progress, and releasing modular high-level libraries for training and deploying embodied agents.

Specifically, Habitat consists of the following –

1. `Habitat-Sim`: a flexible, high-performance 3D simulator with configurable agents, multiple sensors, and generic 3D dataset handling (with built-in support for SUNCG, Matterport3D, and Gibson datasets).

2. `Habitat-API`: a modular high-level library for end-to-end development of embodied AI algorithms – defining embodied AI tasks (*e.g.* navigation, instruction following, question answering), configuring and training embodied agents (via imitation or reinforcement learning, or via classic SLAM), and benchmarking using standard metrics [1].

The Habitat architecture and implementation combine modularity and high performance. When rendering a scene from the Matterport3D dataset, `Habitat-Sim` achieves several thousand frames per second (fps) running single-threaded, and can reach over 10,000 fps multi-process on a single GPU, which is orders of magnitude faster than the closest simulator. `Habitat-API` allows us to train and benchmark embodied agents with different classes of methods and in different 3D scene datasets.

These large-scale engineering contributions enable us to answer scientific questions requiring experiments that were till now impracticable or 'merely' impractical. Specifically, in the context of goal-driven navigation [1], we make two scientific contributions:

1. We revisit the comparison between learning and SLAM approaches from two recent works [19, 15] and find evidence for the **opposite conclusion** – that learning outperforms SLAM if scaled to an order of magnitude more experience than previous investigations.

2. We conduct the first cross-dataset generalization experiments {train, test} × {SUNCG, Matterport3D, Gibson} for multiple sensors {Blind, RGB, RGBD, D} and find that only agents with depth sensors generalize well across datasets.

We hope that our open-source platform and these findings will advance and guide future research in embodied AI.

2. Related Work

The availability of large-scale 3D scene datasets [4, 25, 7] and community interest in active vision tasks led to a recent surge of work that resulted in the development of a variety of simulation platforms for indoor environments [16, 6, 12, 22, 27, 2, 28, 29, 21]. These platforms vary with respect to the 3D scene data they use, the embodied agent tasks they address and the evaluation protocols they implement.

This surge of activity is both thrilling and alarming. On the one hand, it is clearly a sign of the interest in embodied

AI across diverse research communities (computer vision, natural language processing, robotics, machine learning). On the other hand, this variety of simulation environments can cause fragmentation, replication of effort, and difficulty in reproduction and community-wide progress. Moreover, existing simulators exhibit several shortcomings:

– Tight coupling of task (*e.g.* navigation), simulation platform (*e.g.* House3D), and 3D dataset (*e.g.* SUNCG). Experiments with multiple tasks or datasets are impractical.
– Hard-coded agent configuration (*e.g.* size, action-space). Ablations of agent parameters and sensor types are not supported, making results hard to compare.
– Suboptimal rendering and simulation performance. Most existing indoor simulators operate at relatively low frame rates (10-100 fps), becoming a bottleneck in training agents and making large-scale learning infeasible. Takeaway messages from such experiments become unreliable – has the learning converged to trust the comparisons?
– Limited control of environment state. The structure of the 3D scene in terms of present objects cannot be programmatically modified (*e.g.* to test the robustness of agents).

Most critically, work built on top of any of the existing platforms is hard to reproduce independently from the platform, and thus hard to evaluate against work based on a different platform, even in cases where the target tasks and datasets are the same. This status quo is undesirable and motivates the Habitat effort. We aim to learn from the successes of previous frameworks and develop a unifying platform that combines their desirable characteristics while addressing their limitations. A common, unifying platform can significantly accelerate research by enabling code re-use and consistent experimental methodology. Moreover, a common platform enables us to easily carry out experiments testing agents based on different paradigms (learned vs. classical) and generalization of agents between datasets.

The experiments we carry out contrasting learned and classical approaches to navigation are similar to the recent work of Mishkin et al. [19]. However, the performance of the Habitat stack relative to MINOS [22] used in [19] – thousands vs. one hundred frames per second – allows us to evaluate agents that have been trained with significantly larger amounts of experience (75 million steps vs. five million steps). The trends we observe demonstrate that learned agents can begin to match and outperform classical approaches when provided with large amounts of training experience. Other recent work by Koijima and Deng [15] has also compared hand-engineered navigation agents against learned agents but their focus is on defining additional metrics to characterize the performance of agents and to establish measures of hardness for navigation episodes. To our knowledge, our experiments are the first to train navigation agents provided with multi-month experience in realistic indoor environments and contrast them against classical methods.

Figure 2: Example rendered sensor observations for three sensors (color camera, depth sensor, semantic instance mask) in three different environment datasets (from top: SUNCG [25], Matterport3D [7], and state-of-the-art photorealistic reconstruction [26]).

3. Habitat Platform

The development of Habitat is a long-term effort to enable the formation of a common task framework [11] for research into embodied agents, thereby supporting systematic research progress in this area.

Design requirements. The issues discussed in the previous section lead us to a set of requirements that we seek to fulfill.

– **Highly performant rendering engine**: resource-efficient rendering engine that can produce multiple channels of visual information (*e.g.* RGB, depth, semantic instance segmentation, surface normals, optical flow) for multiple concurrently operating agents.
– **Scene dataset ingestion API**: makes the platform agnostic to 3D scene datasets and allows users to use their own.
– **Agent API**: allows users to specify parameterized embodied agents with well-defined geometry, physics, and actuation characteristics.
– **Sensor suite API**: allows specification of arbitrary numbers of parameterized sensors (*e.g.* RGB, depth, contact, GPS, compass sensors) attached to each agent.
– **Scenario and task API**: allows portable definition of tasks and their evaluation protocols.
– **Implementation**: C++ backend with Python API and interoperation with common learning frameworks minimizes entry threshold.
– **Containerization**: enables distributed training in clusters and remote-server evaluation of user-provided code.
– **Humans-as-agents**: allows humans to function as agents in simulation in order to collect human behavior and investigate human-agent or human-human interactions.
– **Environment state manipulation**: programmatic control of the environment configuration in terms of the objects that are present and their relative layout.

Design overview. The above design requirements cut across several layers in the 'software stack' in Figure 1. A monolithic design is not suitable for addressing requirements at all levels. We, therefore, structure the Habitat platform to mirror this multi-layer abstraction.

At the lowest level is `Habitat-Sim`, a flexible, high-performance 3D simulator, responsible for loading 3D scenes into a standardized scene-graph representation, configuring agents with multiple sensors, simulating agent motion, and returning sensory data from an agent's sensor suite.

Generic 3D dataset API using scene graphs. `Habitat-Sim` employs a hierarchical scene graph to represent all supported 3D environment datasets, whether synthetic such as SUNCG [25], or based on real-world reconstructions such as Matterport3D [7] or Gibson [28]. The use of a uniform scene graph representation allows us to abstract the details of specific datasets, and to treat them in a consistent fashion. Scene graphs allow us to compose 3D environments through procedural scene generation, editing, or programmatic manipulation.

Rendering engine. The `Habitat-Sim` backend module is implemented in C++ and leverages the Magnum graphics middleware library[1] to support cross-platform deployment on a broad variety of hardware configurations. The simulator backend employs an efficient rendering pipeline that implements visual sensor frame rendering using a multi-attachment 'uber-shader' combining outputs for color camera sensors, depth sensors, and semantic mask sensors. By allowing all outputs to be produced in a single render pass, we avoid additional overhead when sensor parameters are shared and the same render pass can be used for all outputs. Figure 2 shows examples of visual sensors rendered in three different supported datasets. The same agent and sensor configuration was instantiated in a scene from each of the three datasets by simply specifying a different input scene.

Performance. `Habitat-Sim` achieves thousands of frames per second per simulator thread and is orders of magnitude faster than previous simulators for realistic indoor environments (which typically operate at tens or hundreds of frames per second) – see Table 1 for a summary and the appendix for more details. By operating at 10,000 frames per second we shift the bottleneck from simulation to optimization for network training. Based on TensorFlow benchmarks[2], many popular network architectures run at frame rates that are 10-100x lower on a single GPU. In practice, we have observed that it is often *faster to generate images using `Habitat-Sim` than to load images from disk*.

Efficient GPU throughput. Currently, frames rendered by `Habitat-Sim` are exposed as Python tensors through shared memory. Future development will focus on even higher rendering efficiency by entirely avoiding GPU-to-CPU memory copy overhead through the use of CUDA-GL interoperation and direct sharing of render buffers and textures as tensors. Our preliminary internal testing suggests that this can lead to a speedup by a factor of 2.

[1] https://magnum.graphics/

[2] https://www.tensorflow.org/guide/performance/benchmarks

	1 proc			5 procs		
Sensors / Resolution	128	256	512	128	256	512
RGB	4,093	1,987	848	10,592	3,574	2,629
RGB + depth	2,050	1,042	423	5,223	1,774	1,348

Table 1: Performance of `Habitat-Sim` in frames per second for an example Matterport3D scene (id 17DRP5sb8fy) on an Intel Xeon E5-2690 v4 CPU and Nvidia Titan Xp GPU, measured at different frame resolutions and with a varying number of concurrent simulator processes sharing the GPU.

Above the simulation backend, the `Habitat-API` layer is a modular high-level library for end-to-end development in embodied AI. Setting up an embodied task involves specifying observations that may be used by the agent(s), using environment information provided by the simulator, and connecting the information with a task-specific episode dataset.

- `Task`: this class extends the simulator's `Observations` class and action space with task-specific ones. The criteria of episode termination and measures of success are provided by the `Task`. For example, in goal-driven navigation, `Task` provides the goal and evaluation metric [1]. To support this kind of functionality the `Task` has read-only access to `Simulator` and `Episode-Dataset`.
- `Episode`: a class for episode specification that includes the initial position and orientation of an `Agent`, scene id, goal position, and optionally the shortest path to the goal. An episode is a description of an instance of the task.
- `Environment`: the fundamental environment concept for Habitat, abstracting all the information needed for working on embodied tasks with a simulator.

More details about the architecture of the Habitat platform, performance measurements, and examples of API use are provided in the appendix.

4. PointGoal Navigation at Scale

To demonstrate the utility of the Habitat platform design, we carry out experiments to test for generalization of goal-directed visual navigation agents between datasets of different environments and to compare the performance of learning-based agents against classical agents as the amount of available training experience is increased.

Task definition. We use the PointGoal task (as defined by Anderson *et al*. [1]) as our experimental testbed. This task is ostensibly simple to define – an agent is initialized at a random starting position and orientation in an environment and asked to navigate to target coordinates that are provided

relative to the agent's position; no ground-truth map is available and the agent must use only its sensory input to navigate. However, in the course of our experiments, we realized that this task leaves space for subtle choices that (a) can make a significant difference in experimental outcomes and (b) are either not specified or inconsistent across papers, making comparison difficult. We attempt to be as descriptive as possible about these seemingly low-level choices; we hope the Habitat platform will help iron out these inconsistencies.

Agent embodiment and action space. The agent is physically embodied as a cylindrical primitive shape with diameter 0.2m and height 1.5m. The action space consists of four actions: `turn_left`, `turn_right`, `move_forward`, and `stop`. These actions are mapped to idealized actuations that result in 10 degree turns for the turning actions and linear displacement of 0.25m for the `move_forward` action. The `stop` action allows the agent to signal that it has reached the goal. To be clear, Habitat software supports noisy actuations; experiments in this paper are conducted in the noise-free setting as our analysis focuses on other factors.

Collision dynamics. Some previous works [2] use a coarse irregular navigation graph where an agent effectively 'teleports' from one location to another (1-2m apart). Others [8] use a fine-grained regular grid (0.01m resolution) where the agent moves on unoccupied cells and there are no collisions or partial steps. In Habitat and our experiments, we use a more realistic collision model – the agent navigates in a continuous state space[3] and motion can produce collisions resulting in partial (or no) progress along the direction intended – simply put, it is possible for the agent to 'slide' along a wall or obstacle. Crucially, the agent may choose `move_forward` (0.25m) and end up in a location that is *not* 0.25m forward of where it started; thus, odometry is not trivial even in the absence of actuation noise.

Goal specification: static or dynamic? One conspicuous underspecification in the PointGoal task [1] is whether the goal coordinates are *static* (*i.e.* provided once at the start of the episode) or *dynamic* (*i.e.* provided at *every* time step). The former is more realistic – it is difficult to imagine a real task where an oracle would provide precise dynamic goal coordinates. However, in the absence of actuation noise and collisions, every step taken by the agent results in a known turn or translation, and this combined with the initial goal location is functionally equivalent to dynamic goal specification. We hypothesize that this is why recent works [15, 19, 12] used dynamic goal specification. We follow and prescribe the following conceptual delineation – as a *task*, we adopt static PointGoal navigation; as for the *sensor suite*, we equip our agents with an idealized GPS sensor. This orients us towards a realistic task (static PointGoal navigation), disentangles simulator design (actuation noise, collision dynamics) from the task definition, and allows us to compare techniques by sensors used (RGB, depth, GPS, compass, contact sensors).

Sensory input. The agents are endowed with a single color vision sensor placed at a height of 1.5m from the center of the agent's base and oriented to face 'forward'. This sensor provides RGB frames at a resolution of 256^2 pixels and with a field of view of 90 degrees. In addition, an idealized depth sensor is available, in the same position and orientation as the color vision sensor. The field of view and resolution of the depth sensor match those of the color vision sensor. We designate agents that make use of the color sensor by `RGB`, agents that make use of the depth sensor by `Depth`, and agents that make use of both by `RGBD`. Agents that use neither sensor are denoted as `Blind`. All agents are equipped with an idealized GPS, but not a compass – *i.e.*, they have access to their location coordinates but not orientation. Note that this is a *weaker* sensor suite than used by prior work assuming access to both [12].

Episode specification. We initialize the agent at a starting position and orientation that are sampled uniformly at random from all navigable positions on the floor of the environment. The goal position is chosen such that it lies on the same floor and there exists a navigable path from the agent's starting position. During the episode, the agent is allowed to take up to 500 actions. After each action, the agent receives a set of observations from the active sensors.

Evaluation. The navigation episode is considered successful if and only if the agent issues a `stop` action within 0.2m of the target coordinates, as measured by a geodesic distance along the shortest path from the agent's position to the goal position. If the agent takes 500 actions without the above condition being met the episode ends and is considered unsuccessful. Performance is measured using the 'Success weighted by Path Length' (`SPL`) metric [1]. For an episode where the geodesic distance of the shortest path is l and the agent traverses a distance p, `SPL` is defined as $S \cdot {}^{l}/_{\max(p,l)}$, where S is a binary indicator of success.

Episode dataset preparation. We create PointGoal navigation episode-datasets for SUNCG [25], Matterport3D [7] and Gibson [28] scenes. We randomly subsample 990 scenes in SUNCG, and followed the publicly available (train/-val/test) splits for Matterport3D. Note that similar to recent works [8, 19, 15], there is no overlap between train, val, test scenes. For Gibson scenes, we obtained textured 3D surface meshes from the Gibson authors [28], manually annotated each scene on its reconstruction quality (small/big holes, floating/irregular surfaces, poor textures), and curated a subset of 106 scenes (out of 572); see the appendix for details. An episode is defined by the unique id of the scene, the starting position, and orientation of the agent, and the goal position. Additional metadata such as the geodesic distance along the shortest path (`GDSP`) from start position to goal

[3] Up to machine precision.

position is also included. While generating episodes, we restrict the GDSP to be between 1m and 30m. An episode is trivial if there is an obstacle-free, straight line between the start and goal positions. A good measure of the navigation complexity of an episode is the ratio of GDSP to Euclidean distance between start and goal positions (notice that GDSP can only be larger than Euclidean distance). If the ratio is nearly 1, it indicates there are few obstacles, and the episode is easy; if the ratio is larger than 1, the episode is difficult because strategic navigation is required. To keep the navigation complexity of the precomputed episodes reasonably high, we perform aggressive rejection sampling for episodes with the above ratio falling in the range $[1, 1.1]$. Following this, there is a significant decrease in the number of straight-line episodes (from 37% to 10% for the Gibson dataset generation). This step was not performed in any previous studies; we find that without this bias controlling, all metrics appear inflated. Gibson scenes have smaller physical dimensions compared to the Matterport3D scenes. This is reflected in the resulting PointGoal dataset – average GDSP of episodes in Gibson scenes is smaller than that of Matterport3D scenes. SUNCG scenes are physically the largest and navigationally most complex; see appendix for details.

Baselines. We compare the following baselines:

- **Random** chooses an action randomly among `turn_left`, `turn_right`, and `move_forward` with uniform distribution. The agent calls the `stop` action when within 0.2m of the goal (computed using the difference of static goal and dynamic GPS coordinates).
- **Forward only** always calls the `move_forward` action, and calls the `stop` action when within 0.2m of the goal.
- **Goal follower** moves towards the goal direction. If it is not facing the goal (more than 15 degrees off-axis), it performs `turn_left` or `turn_right` to align itself; otherwise, it calls `move_forward`. The agent calls the `stop` action when within 0.2m of the goal.
- **RL (PPO)** is an agent trained with reinforcement learning, specifically proximal policy optimization [24]. We experiment with RL-agents equipped with different visual sensors: no visual input (`Blind`), `RGB` input, `Depth` input, and RGB with depth (`RGBD`). The model consists of a CNN that produces an embedding for visual input, which together with the relative goal vector is used by an actor (GRU) and a critic (linear layer). The CNN has the following architecture: {Conv 8×8, ReLU, Conv 4×4, ReLU, Conv 3×3, ReLU, Linear, ReLU}. Let r_t denote the reward at timestep t, d_t be the geodesic distance to goal at timestep t, s a success reward and λ a time penalty (to encourage efficiency). All models were trained with the following reward function:

$$r_t = \begin{cases} s + d_{t-1} - d_t + \lambda & \text{if goal is reached} \\ d_{t-1} - d_t + \lambda & \text{otherwise} \end{cases}$$

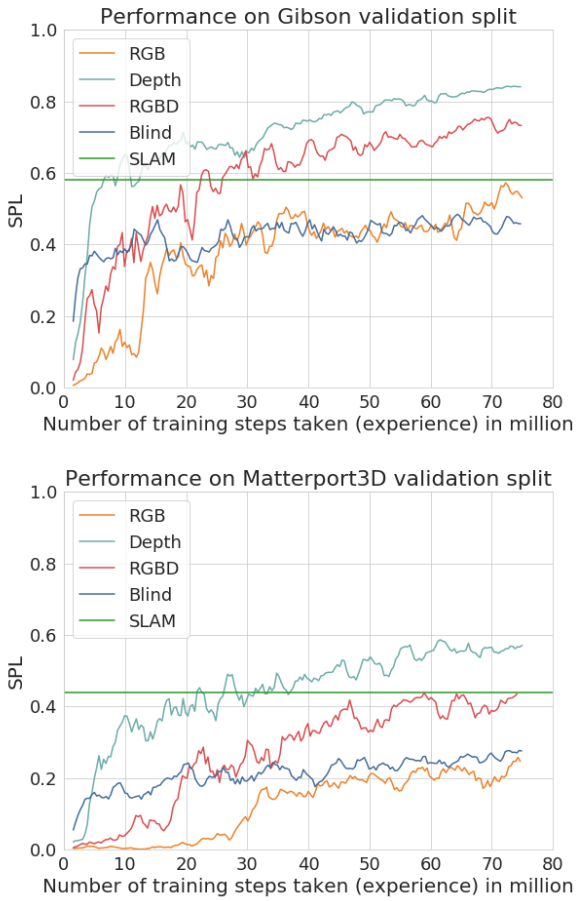


Figure 3: Average SPL of agents on the val set over the course of training. Previous work [19, 23] has analyzed performance at 5-10 million steps. Interesting trends emerge with more experience: i) `Blind` agents start performing much better (initially outperforming `RGB` and `RGBD`) but saturate quickly; ii) `Depth` and `RGBD` learning-agents match the and outperform classical SLAM.

In our experiments s is set to 10 and λ is set to -0.01.

- **SLAM [19]** is an agent implementing a classical robotics navigation pipeline (including components for localization, mapping, and planning), using RGB and depth sensors. We use the classic agent by Mishkin *et al*. [19] which leverages the ORB-SLAM2 [20] localization pipeline, with the same parameters as reported in the original work.

Training procedure. When training learning-based agents, we first divide the scenes in the training set equally among 6 concurrently running simulator worker threads. Each thread establishes blocks of 500 training episodes for each scene in its training set partition and shuffles the ordering of these blocks. Training continues through shuffled copies of this array. For the experiments reported here, we train until 75 million agent steps are accumulated across all worker threads. This is 15x larger than the experience used in previous investigations [19, 15]. Training agents to 75 million steps took (in sum over all three datasets): 320 GPU-hours for `Blind`,

Sensors	Baseline	Gibson SPL	Gibson Succ	MP3D SPL	MP3D Succ	SUNCG SPL	SUNCG Succ
Blind	Random	0.02	0.03	0.01	0.01	0.02	0.03
	Forward only	0.00	0.00	0.00	0.00	0.00	0.00
	Goal follower	0.23	0.23	0.12	0.12	0.23	0.24
	RL (PPO)	0.42	0.62	0.25	0.35	0.35	0.54
RGB	RL (PPO)	0.46	0.64	0.30	0.42	0.42	0.59
Depth	RL (PPO)	**0.79**	**0.89**	**0.54**	**0.69**	**0.55**	**0.72**
RGBD	RL (PPO)	0.70	0.80	0.42	0.53	0.42	0.57
	SLAM [19]	0.51	0.62	0.39	0.47	–	–

Table 2: Performance of baseline methods on the PointGoal task [1] tested on the Gibson [28], MP3D [7] and SUNCG [25] test sets under multiple sensor configurations. RL models have been trained for 75 million steps (except RGBD SUNCG for 45 million steps). We report average rate of episode success and SPL [1].

566 GPU-hours for RGB, 475 GPU-hours for Depth, and 906 GPU-hours for RGBD (overall 2267 GPU-hours).

5. Results and Findings

We seek to answer two questions: i) how do learning-based agents compare to classical SLAM and hand-coded baselines as the amount of training experience increases and ii) how well do learned agents generalize across 3D datasets.

It should be tacitly understood, but to be explicit – 'learning' and 'SLAM' are broad families of techniques (and not a single method), are not necessarily mutually exclusive, and are not 'settled' in their development. We compare representative instances of these families to gain insight into questions of scaling and generalization, and do not make any claims about intrinsic superiority of one or the other.

Learning vs SLAM. To answer the first question we plot agent performance (SPL) on validation (*i.e.* unseen) episodes over the course of training in Figure 3 (top: Gibson, bottom: Matterport3D). SLAM [19] does not require training and thus has a constant performance (0.59 on Gibson, 0.42 on Matterport3D). All RL (PPO) agents start out with far worse SPL, but RL (PPO) Depth, in particular, improves dramatically and matches the classical baseline at approximately 10M frames (Gibson) or 30M frames (Matterport3D) of experience, continuing to improve thereafter. Notice that if we terminated the experiment at 5M frames as in [19, 15] we would also conclude that SLAM [19] dominates. Interestingly, RGB agents do not significantly outperform Blind agents; we hypothesize because both are equipped with GPS sensors. Indeed, qualitative results (Figure 5) suggest that Blind agents 'hug' walls and implement 'wall following' heuristics. In contrast, RGB sensors provide a high-dimensional complex signal that may be prone to overfitting to train environments due to the variety across scenes (even within the same dataset). We also notice in Figure 3 that *all methods* perform better on Gibson than Matterport3D. This is consistent with our previous analysis that Gibson contains smaller scenes and shorter episodes.

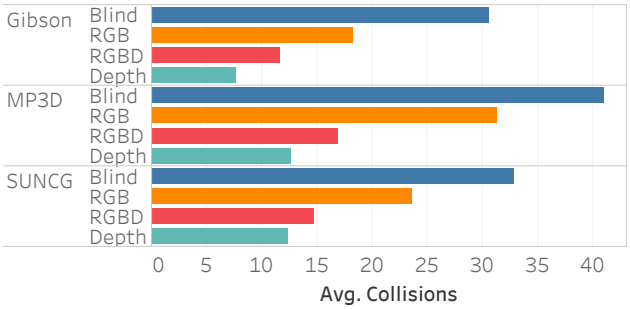


Figure 4: Average number of collisions during successful navigation episodes for the different sensory configurations of the RL (PPO) baseline agent on test set episodes for the Gibson, MP3D and SUNCG datasets. The Blind agent experiences the highest number of collisions, while agents possessing depth sensors (Depth and RGBD) have the fewest collisions on average.

Next, for each agent and dataset, we select the best-performing checkpoint on validation and report results on test in Table 2. We observe that uniformly across all datasets, RL (PPO) Depth performs the best, outperforming RL (PPO) RGBD (by 0.09-0.16 SPL), SLAM (by 0.15-0.28 SPL), and RGB (by 0.13-0.33) in that order. These differences are an order of magnitude larger than standard error bars of ± 0.01 SPL around the performance of these agents. Random and forward only agents have very low performance, while the hand-coded goal follower and Blind baseline see modest performance.[4]

To further characterize the behavior of learned agents during navigation we plot the average number of collisions in Figure 4. We see that Blind incurs a much larger number of collisions than other agents, providing evidence for 'wall-following' behavior. Depth-equipped agents have the lowest number of collisions, while RGB agents are in between.

In Figure 5 we plot example trajectories for the RL (PPO) agents to qualitatively contrast their behavior in the same episode. Consistent with the aggregate statistics, we observe that Blind collides with obstacles and follows along walls, while Depth is the most efficient.

Generalization across datasets. Our findings so far are that RL (PPO) agents significantly outperform SLAM [19]. This prompts our second question – are these findings dataset specific or do learned agents generalize across datasets? We report exhaustive comparisons in Figure 6 – specifically, average SPL for all combinations of {train, test} $\times$ {SUNCG, Matterport3D, Gibson} for all agents {Blind, RGB, RGBD, Depth }. Rows indicate (agent, train set) pair, columns indicate test set. We find a number of interesting trends. First, nearly all agents suffer

[4] We omit SUNCG SLAM results as the implementation we used [19] was tuned for realistic scenes (MP3D, Gibson) and performs unrepresentatively poorly when naively applied to synthetic scenes in SUNCG.

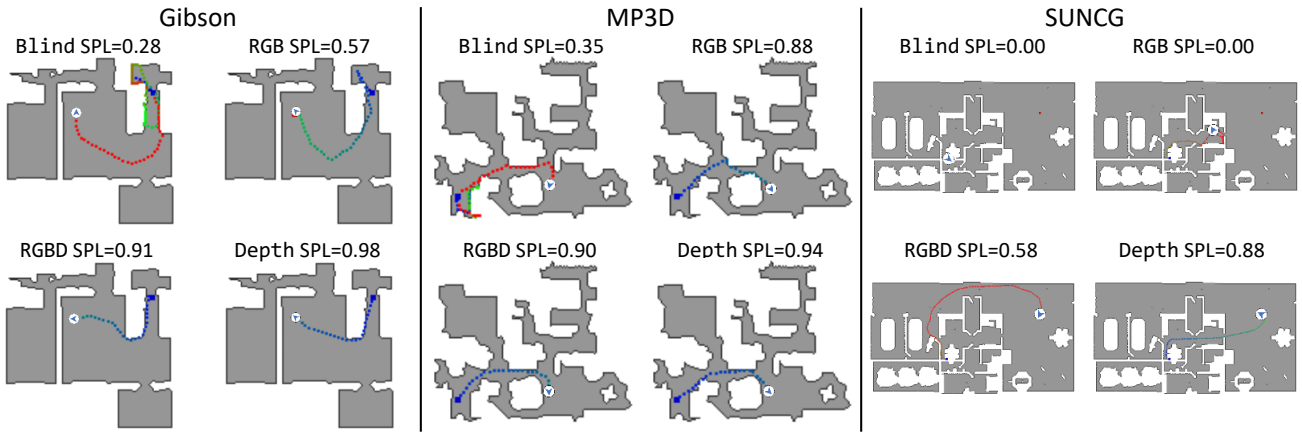


Figure 5: Navigation examples for the different sensory configurations of the RL (PPO) agent, visualizing trials from the Gibson, MP3D and SUNCG val sets. A **blue dot** and **red dot** indicate the starting and goal positions, and the **blue arrow** indicates final agent position. The **blue-green-red line** is the agent's trajectory. Color shifts from blue to red as the maximum number of allowed agent steps is approached.

		Gibson	MP3D	SUNCG
Blind	Gibson	0.42	0.34	0.38
	MP3D	0.28	0.25	0.25
	SUNCG	0.35	0.29	0.35
RGB	Gibson	0.46	0.40	0.28
	MP3D	0.25	0.30	0.17
	SUNCG	0.25	0.24	0.42
Depth	Gibson	0.79	0.68	0.64
	MP3D	0.56	0.54	0.49
	SUNCG	0.50	0.48	0.55
RGBD	Gibson	0.70	0.53	0.48
	MP3D	0.44	0.42	0.35
	SUNCG	0.42	0.38	0.42

Figure 6: Generalization of agents between datasets. We report average SPL for a model trained on the source dataset in each row, as evaluated on test episodes for the target dataset in each column.

a drop in performance when trained on one dataset and tested on another, *e.g.* `RGBD` Gibson→Gibson 0.70 vs `RGBD` Gibson→Matterport3D 0.53 (drop of 0.17). `Depth` agent generalizes the best across datasets – interestingly, not only between Gibson↔MatterPort3D *but also to/from SUNCG*, a synthetic dataset. `Depth` trained on SUNCG achieves 0.50, 0.48, 0.55 on Gibson, Matterport3D, SUNCG respectively. This is promising evidence that agents trained in synthetic environments can generalize to scanned environments, laying the groundwork for simulation-to-reality transfer experiments. In contrast, `RGB` and `RGBD` agents suffer a significant performance degradation, while the `Blind` agent is effected the least (as we would expect).

Second, we find a potentially counter-intuitive trend – agents trained on Gibson consistently outperform their counterparts trained on Matterport3D and SUNCG, *even when evaluated on Matterport3D and SUNCG*. For instance, `Depth` Gibson→SUNCG is 0.64, while `Depth` SUNCG→SUNCG is 0.55. This is true for all combinations except that `RGB` Gibson→SUNCG is worse than SUNCG→SUNCG. We believe the reason is the previously noted observation that Gibson scenes are smaller and episodes are shorter (lower `GDSP`) than Matterport3D and SUNCG. Gibson agents are trained on 'easier' episodes and encounter positive reward more easily during random exploration, thus bootstrapping learning. Consequently, for a fixed computation budget Gibson agents are stronger universally (not just on Gibson). This finding suggests that visual navigation agents could benefit from curriculum learning.

These scientific insights are enabled by the large-scale engineering that went into Habitat, which made these experiments as 'trivial' as a change in the evaluation dataset name.

6. Habitat Challenge

Challenges drive progress. The history of AI sub-fields indicates that the formulation of the right questions, the creation of the right datasets, and the coalescence of communities around the right challenges drives scientific progress. Our goal is support this process for embodied AI. Habitat Challenge is an autonomous navigation challenge that aims to benchmark and advance efforts in goal-directed visual navigation.

One difficulty in creating a challenge around embodied AI tasks is the transition from static predictions (as in passive perception) to sequential decision making (as in sensorimotor control). In traditional 'internet AI' challenges (*e.g.* ImageNet [9], COCO [17], VQA [3]), it is possible to release a static testing dataset and ask participants to simply upload their predictions on this set. In contrast, embodied AI tasks typically involve sequential decision making and agent-driven control, making it infeasible to pre-package a testing dataset. Essentially, embodied AI challenges require participants to *upload code not predictions*. The uploaded agents can then be evaluated in novel (unseen) test environments.

Challenge infrastructure. We leverage the frontend and challenge submission process of the EvalAI platform, and build backend infrastructure ourselves. Participants in Habitat Challenge are asked to upload Docker containers [18]

with their agents via EvalAI. The submitted agents are then evaluated on a live AWS GPU-enabled instance. Specifically, contestants are free to train their agents however they wish (any language, any framework, any infrastructure). In order to evaluate these agents, participants are asked to derive from a base Habitat Docker container and implement a specific interface to their model – agent's action taken given an observation from the environment at each step. This dockerized interface enables running the participant code on new environments.

More details regarding the challenge are available at the `https://aihabitat.org/challenge` website. We look forward to supporting the community in establishing a benchmark to evaluate the state-of-the-art in methods for embodied navigation agents.

7. Future Work

We described the design and implementation of the Habitat platform. Our goal is to unify existing community efforts and to accelerate research into embodied AI. This is a long-term effort that will succeed only by full engagement of the broader research community. To this end, we have open-sourced the entire Habitat platform stack.

Experiments enabled by the generic dataset support and the high performance of the Habitat stack indicate that i) learning-based agents can match and exceed the performance of classical visual navigation methods when trained for long enough and ii) learned agents equipped with depth sensors generalize well between different 3D environment datasets in comparison to agents equipped with only RGB.

Feature roadmap. Our near-term development roadmap will focus on incorporating physics simulation and enabling physics-based interaction between mobile agents and objects in 3D environments. `Habitat-Sim`'s scene graph representation is well-suited for integration with physics engines, allowing us to directly control the state of individual objects and agents within a scene graph. Another planned avenue of future work involves procedural generation of 3D environments by leveraging a combination of 3D reconstruction and virtual object datasets. By combining high-quality reconstructions of large indoor spaces with separately reconstructed or modelled objects, we can take full advantage of our hierarchical scene graph representation to introduce controlled variation in the simulated 3D environments.

Lastly, we plan to focus on distributed simulation settings that involve large numbers of agents potentially interacting with one another in competitive or collaborative scenarios.

Acknowledgments

The Habitat project would not have been possible without the support and contributions of many individuals. We thank Fei Xia and Amir R. Zamir for preparing the Gibson dataset for the Habitat Challenge. We thank Dmytro Mishkin and Alexey Dosovitskiy for contributing the classic baseline. We thank Xinlei Chen, Georgia Gkioxari, Daniel Gordon, Leonidas Guibas, Saurabh Gupta, Or Litany, Marcus Rohrbach, Amanpreet Singh, Devendra Singh Chaplot, Yuandong Tian, and Yuxin Wu for many helpful conversations and guidance on the design and development of the Habitat platform.

References

[1] P. Anderson, A. X. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018. 1, 2, 4, 5, 7, 10

[2] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018. 2, 5, 10

[3] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: Visual Question Answering. In *ICCV*, 2015. 1, 2, 8

[4] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 2

[5] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall. Learning to drive from simulation without real world labels. In *ICRA*, 2019. 2

[6] S. Brodeur, E. Perez, A. Anand, F. Golemo, L. Celotti, F. Strub, J. Rouat, H. Larochelle, and A. C. Courville. HoME: A household multimodal environment. *arXiv:1711.11017*, 2017. 2

[7] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *International Conference on 3D Vision (3DV)*, 2017. 2, 3, 4, 5, 7, 10

[8] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied Question Answering. In *CVPR*, 2018. 5, 10

[9] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 2, 8

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018. 1

[11] D. Donoho. 50 years of data science. In *Tukey Centennial Workshop*, 2015. 3

[12] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017. 2, 5

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[14] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 2019. 2

[15] N. Kojima and J. Deng. To learn or not to learn: Analyzing the role of learning for navigation in virtual environments. 2019. 1, 2, 3, 5, 6, 7

[16] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv:1712.05474*, 2017. 2

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 1, 2, 8

[18] D. Merkel. Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014. 8

[19] D. Mishkin, A. Dosovitskiy, and V. Koltun. Benchmarking classic and learned navigation in complex 3D environments. *arXiv:1901.10915*, 2019. 1, 2, 3, 5, 6, 7

[20] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5), 2017. 6

[21] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. VirtualHome: Simulating household activities via programs. In *CVPR*, 2018. 2

[22] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*, 2017. 2, 3

[23] A. Sax, B. Emi, A. R. Zamir, L. J. Guibas, S. Savarese, and J. Malik. Mid-level visual representations improve generalization and sample efficiency for learning active tasks. *arXiv:1812.11971*, 2018. 6

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017. 6

[25] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2017. 2, 3, 4, 5, 7, 10

[26] T. Whelan, M. Goesele, S. J. Lovegrove, J. Straub, S. Green, R. Szeliski, S. Butterfield, S. Verma, and R. Newcombe. Reconstructing scenes with mirror and glass surfaces. *ACM Transactions on Graphics (TOG)*, 2018. 3

[27] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building generalizable agents with a realistic and rich 3D environment. *arXiv:1801.02209*, 2018. 2

[28] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, 2018. 2, 4, 5, 7, 10, 11

[29] C. Yan, D. Misra, A. Bennnett, A. Walsman, Y. Bisk, and Y. Artzi. CHALET: Cornell house agent learning environment. *arXiv:1801.07357*, 2018. 2

A. Habitat Platform Details

As described in the main paper, Habitat consists of the following components:

- `Habitat-Sim`: a flexible, high-performance 3D simulator with configurable agents, multiple sensors, and generic 3D dataset handling (with built-in support for SUNCG [25], Matterport3D [7], Gibson [28], and other datasets). `Habitat-Sim` is fast – when rendering a realistic scanned scene from the Matterport3D dataset, `Habitat-Sim` achieves several thousand frames per second (fps) running single-threaded, and can reach over 10,000 fps multi-process on a single GPU.
- `Habitat-API`: a modular high-level library for end-to-end development of embodied AI – defining embodied AI tasks (*e.g.* navigation [1], instruction following [2], question answering [8]), configuring embodied agents (physical form, sensors, capabilities), training these agents (via imitation or reinforcement learning, or via classic SLAM), and benchmarking their performance on the defined tasks using standard metrics [1]. `Habitat-API` currently uses `Habitat-Sim` as the core simulator, but is designed with a modular abstraction for the simulator backend to maintain compatibility over multiple simulators.

Key abstractions. The Habitat platform relies on a number of key abstractions that model the domain of embodied agents and tasks that can be carried out in three-dimensional indoor environments. Here we provide a brief summary of key abstractions:

- `Agent`: a physically embodied agent with a suite of `Sensors`. Can observe the environment and is capable of taking actions that change agent or environment state.
- `Sensor`: associated with a specific `Agent`, capable of returning observation data from the environment at a specified frequency.
- `SceneGraph`: a hierarchical representation of a 3D environment that organizes the environment into regions and objects. Can be programmatically manipulated.
- `Simulator`: an instance of a simulator backend. Given actions for a set of configured `Agents` and `SceneGraphs`, can update the state of the `Agents` and `SceneGraphs`, and provide observations for all active `Sensors` possessed by the `Agents`.

These abstractions connect the different layers of the platform. They also enable generic and portable specification of embodied AI tasks.

Habitat-Sim. The architecture of the `Habitat-Sim` backend module is illustrated in Figure 7. The design of this module ensures a few key properties:

- Memory-efficient management of 3D environment resources (triangle mesh geometry, textures, shaders) ensuring shared resources are cached and reused.
- Flexible, structured representation of 3D environments using `SceneGraphs`, allowing for programmatic manipulation of object state, and combination of objects from different environments (*e.g.* SUNCG objects in Matterport3D homes).
- High-efficiency rendering engine with multi-attachment render pass to reduce overhead for multiple sensors.
- Arbitrary numbers of `Agents` and corresponding `Sensors` that can be linked to a 3D environment by attachment to a `SceneGraph`.

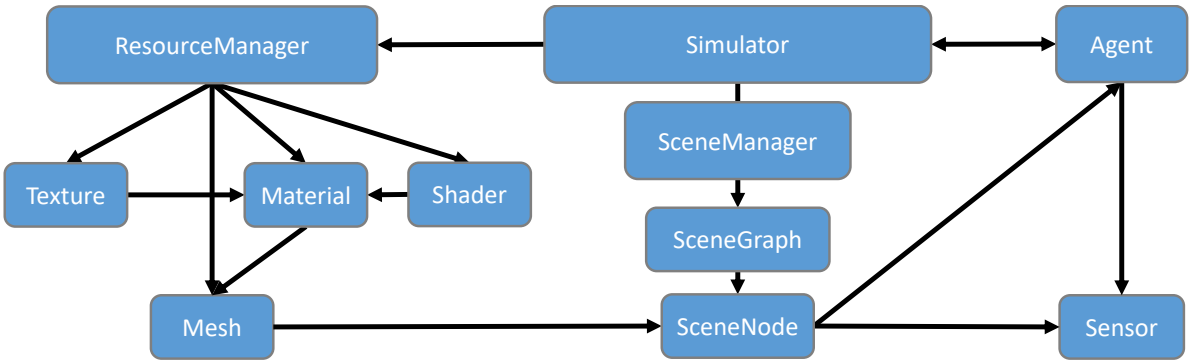


Figure 7: Architecture of `Habitat-Sim` main classes. The Simulator delegates management of all resources related to 3D environments to a ResourceManager that is responsible for loading and caching 3D environment data from a variety of on-disk formats. These resources are used within SceneGraphs at the level of individual SceneNodes that represent distinct objects or regions in a particular Scene. Agents and their Sensors are instantiated by being attached to SceneNodes in a particular SceneGraph.

The performance of the simulation backend surpasses that of prior work operating on realistic reconstruction datasets by a large margin. Table 3 reports performance statistics on a test scene from the Matterport3D dataset. Single-thread performance reaches several thousand frames per second (fps), while multi-process operation with several simulation backends can reach over 10,000 fps on a single GPU.

Habitat-API. The second layer of the Habitat platform (`Habitat-API`) focuses on creating a general and flexible API for defining embodied agents, tasks that they may carry out, and evaluation metrics for those tasks. When designing such an API, a key consideration is to allow for easy extensibility of the defined abstractions. This is particularly important since many of the parameters of embodied agent tasks, specific agent configurations, and 3D environment setups can be varied in interesting ways. Future research is likely to propose new tasks, new agent configurations, and new 3D environments.

The API allows for alternative simulator backends to be used, beyond the `Habitat-Sim` module that we implemented. This modularity has the advantage of allowing incorporation of existing simulator backends to aid in transitioning from experiments that previous work has performed using legacy frameworks. The architecture of `Habitat-API` is illustrated in Figure 8, indicating core API functionality and functionality implemented as extensions to the core.

Above the API level, we define a concrete embodied task such as visual navigation. This involves defining a specific dataset configuration, specifying the structure of episodes (*e.g.* number of steps taken, termination conditions), training curriculum (progression of episodes, difficulty ramp), and evaluation procedure (*e.g.* test episode sets and task metrics).

[5]Note: The semantic sensor in MP3D requires using additional 3D meshes with significantly more geometric complexity, leading to reduced performance. We expect this to be addressed in future versions, leading to speeds comparable to RGB + depth.

An example of loading a pre-configured task (PointNav) and stepping through the environment with a random agent is shown in the code below.

```
import habitat

# Load embodied AI task (PointNav)
# and a pre-specified virtual robot
config = habitat.get_config(config_file=
                            "pointnav.yaml")
env = habitat.Env(config)

observations = env.reset()

# Step through environment with random actions
while not env.episode_over:
    observations = \
        env.step(env.action_space.sample())
```

B. Additional Dataset Statistics

In Table 4 we summarize the train, validation and test split sizes for all three datasets used in our experiments. We also report the average geodesic distance along the shortest path (`GDSP`) between starting point and goal position. As noted in the main paper, Gibson episodes are significantly shorter than MP3D and SUNCG episodes.

Figure 9 visualizes the episode distributions over geodesic distance (`GDSP`), Euclidean distance between start and goal position, and the ratio of the two (an approximate measure of complexity for the episode). We again note that Gibson episodes have more episodes with shorter distances, leading to the dataset being overall easier than the MP3D and SUNCG datasets.

C. Gibson Dataset Curation

We manually curated the full dataset of Gibson 3D textured meshes [28] to select meshes that do not exhibit significant reconstruction artifacts such as holes or texture quality

	1 proc			3 procs			5 procs		
Sensors / Resolution	128	256	512	128	256	512	128	256	512
RGB	4,093	1,987	848	10,638	3,428	2,068	10,592	3,574	2,629
RGB + depth	2,050	1,042	423	5,024	1,715	1,042	5,223	1,774	1,348
RGB + depth + semantics[5]	439	346	185	502	385	336	500	390	367

Table 3: Performance of `Habitat-Sim` in frames per second for an example Matterport3D scene (id 17DRP5sb8fy) on a Xeon E5-2690 v4 CPU and Nvidia Titan Xp GPU, measured at different frame resolutions and with a varying number of concurrent simulator processes sharing the GPU.

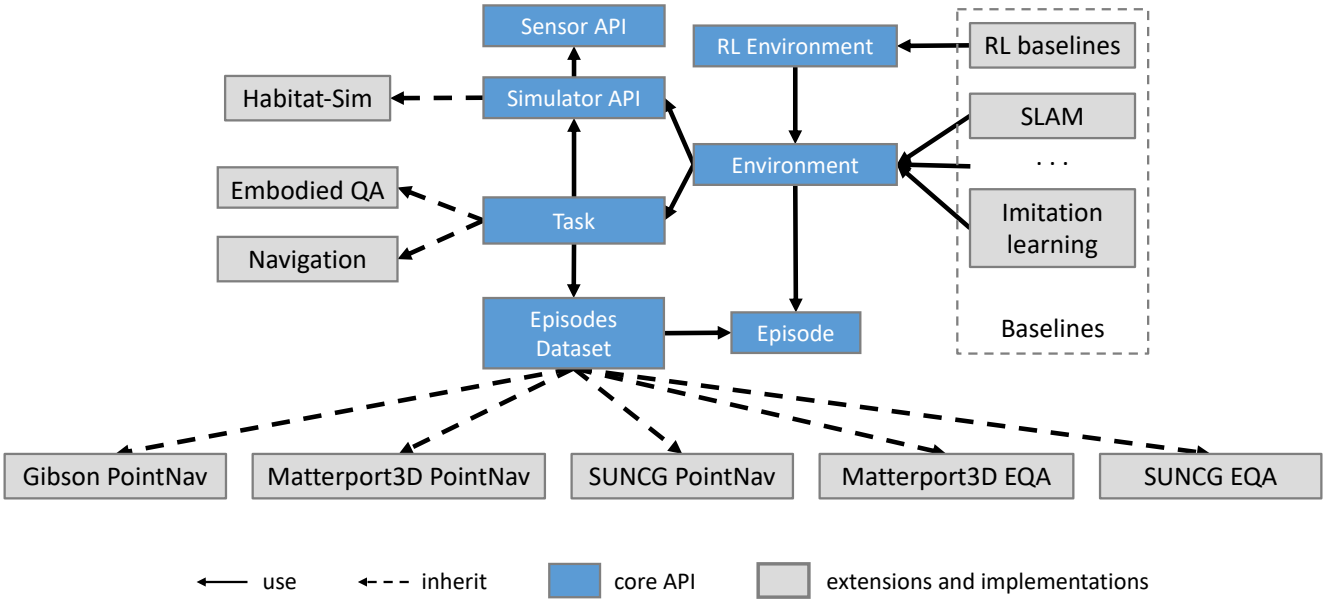


Figure 8: Architecture of `Habitat-API`. The core functionality defines fundamental building blocks such as the API for interacting with the simulator backend and receiving observations through `Sensors`. Concrete simulation backends, 3D datasets, and embodied agent baselines are implemented as extensions to the core API.

Dataset	scenes (#)	episodes (#)	average `GDSP` (m)
Matterport3D	58 / 11 / 18	4.8M / 495 / 1008	11.5 / 11.1 / 13.2
Gibson	72 / 16 / 10	4.9M / 1000 / 1000	6.9 / 6.5 / 7.0
SUNCG	990 / 99 / 100	0.9M / 905 / 1000	11.1 / 10.6 / 10.4

Table 4: Statistics of the PointGoal navigation datasets that we precompute for the Matterport3D, Gibson, and SUNCG datasets: total number of scenes, total number of episodes, and average geodesic distance between start and goal positions. Each cell reports train / val / test split statistics.

issues. A key issue that we tried to avoid is the presence of holes or cracks in floor surfaces. This is particularly problematic for navigation tasks as it divides seemingly connected navigable areas into non-traversable disconnected components. We manually annotated the scenes (using the 0 to 5 quality scale shown in Figure 10) and only use scenes with a rating of 4 or higher, i.e., no holes, good reconstruction, and negligible texture issues to generate the dataset episodes.

D. Example Navigation Episodes

Figure 11 visualizes additional example navigation episodes for the different sensory configurations of the RL (PPO) agents that we describe in the main paper. `Blind` agents have the lowest performance, colliding much more frequently with the environment and adopting a 'wall hugging' strategy for navigation. `RGB` agents are less prone to collisions but still struggle to navigate to the goal position successfully in some cases. In contrast, depth-equipped agents are much more efficient, exhibiting fewer collisions, and navigating to goals more successfully (as indicated by the overall higher `SPL` values).

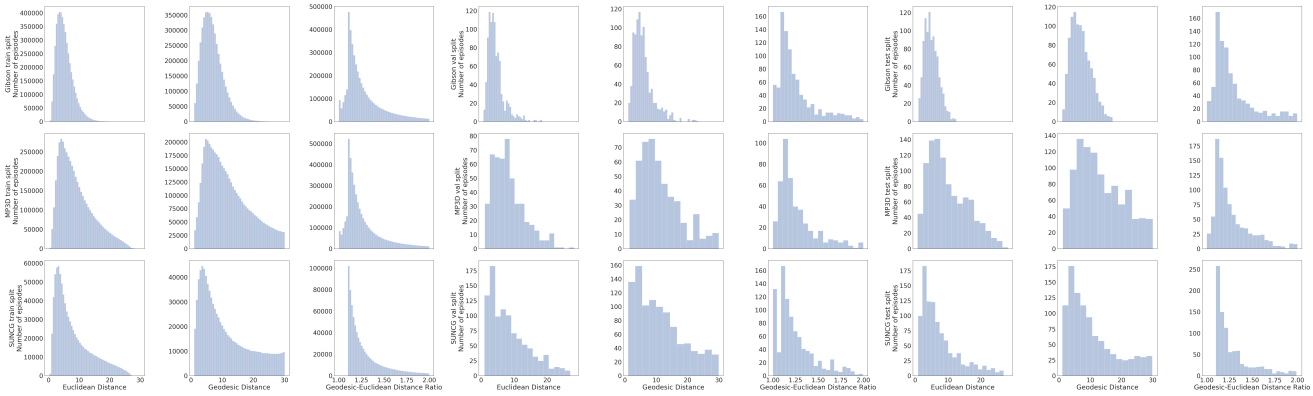


Figure 9: Statistics of PointGoal navigation episodes. From left: distribution over Euclidean distance between start and goal, distribution over geodesic distance along shortest path between start and goal, and distribution over the ratio of geodesic to Euclidean distance.

0: critical reconstruction artifacts, holes, or texture issues

1: big holes or significant texture issues and reconstruction artifacts

2: big holes or significant texture issues, but good reconstruction

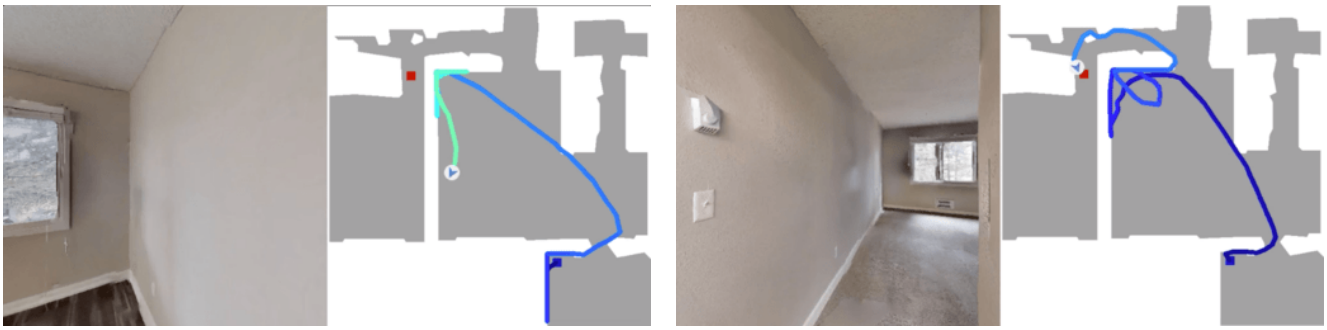
3: small holes, some texture issues, good reconstruction

4: no holes, some texture issues, good reconstruction

5: no holes, uniform textures, good reconstruction

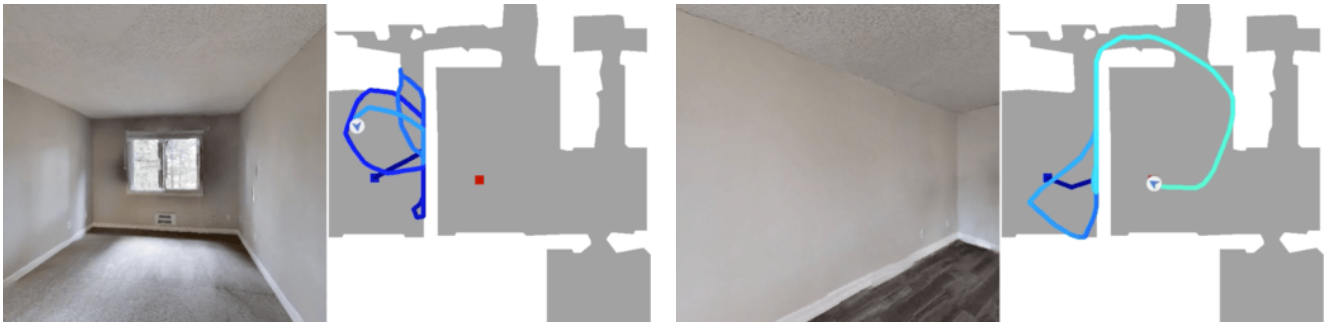
Figure 10: Rating scale used in curation of 3D textured mesh reconstructions from the Gibson dataset. We use only meshes with ratings of 4 or higher for the Habitat Challenge dataset.

Gibson



Blind SPL = 0.00 RGB SPL = 0.45

RGBD SPL = 0.82 Depth SPL = 0.88



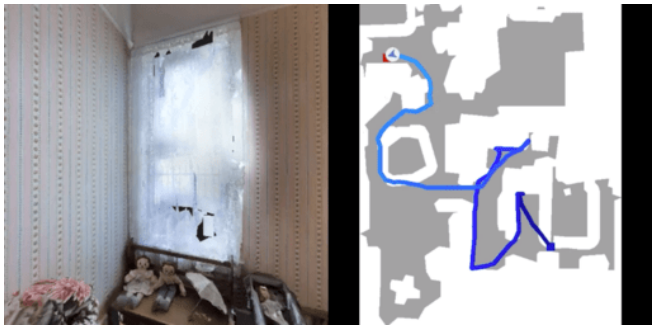
Blind SPL = 0.00 RGB SPL = 0.29

RGBD SPL = 0.49 Depth SPL = 0.96

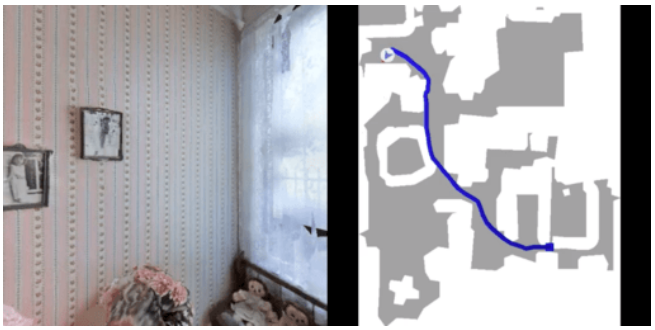
Figure 11: Additional navigation example episodes for the different sensory configurations of the RL (PPO) agent, visualizing trials from the Gibson, MP3D and SUNCG val sets. A **blue dot** and **red dot** indicate the starting and goal positions, and the **blue arrow** indicates final agent position. The **blue-green-red line** is the agent's trajectory. Color shifts from blue to red as the maximum number of allowed agent steps is approached.

MP3D

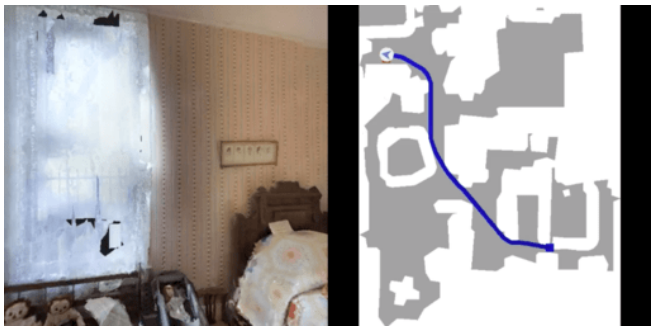
Blind SPL = 0.00



RGB SPL = 0.40

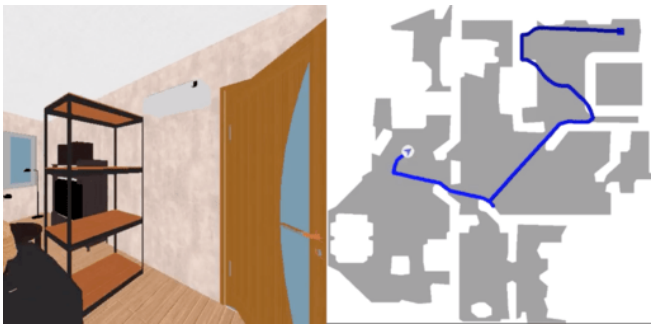


RGBD SPL = 0.92



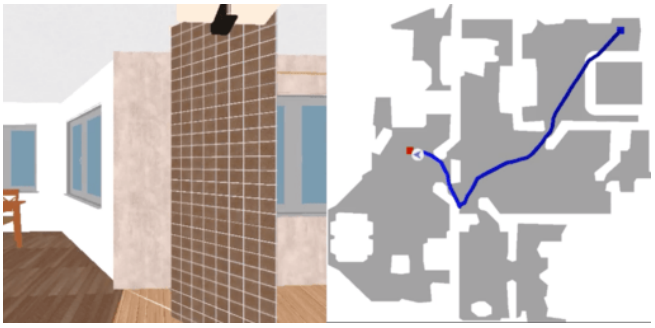
Depth SPL = 0.98

SUNCG

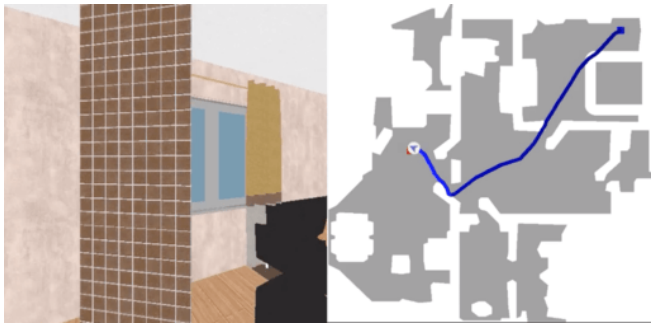


Blind SPL = 0.53

RGB SPL = 0.59



RGBD SPL = 0.80



Depth SPL = 0.84

Figure 11: Additional navigation example episodes for the different sensory configurations of the RL (PPO) agent, visualizing trials from the Gibson, MP3D and SUNCG val sets. A **blue dot** and **red dot** indicate the starting and goal positions, and the **blue arrow** indicates final agent position. The **blue-green-red line** is the agent's trajectory. Color shifts from blue to red as the maximum number of allowed agent steps is approached.