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

Abstract

We present Habitat, a platform for research in embodied artificial intelligence (AI). Habitat enables training embodied agents (virtual robots) in highly efficient photorealistic 3D simulation. Specifically, Habitat consists of:
(i) `Habitat-Sim`*: a flexible, high-performance 3D simulator with configurable agents, sensors, and generic 3D dataset handling.* `Habitat-Sim` *is fast – when rendering a scene from Matterport3D, it achieves several thousand frames per second (fps) running single-threaded, and can reach over* 10,000 *fps multi-process on a single GPU.*
(ii) `Habitat-API`*: a modular high-level library for end-to-end development of embodied AI algorithms – defining tasks (*e.g. *navigation, instruction following, question answering), configuring, training, and benchmarking embodied agents.*

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, 16] 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*} × {*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.*

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 [14, 11], 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 [10, 18, 4]. 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), *resource intensive* (the robot(s) and the environment(s) in which they execute demand resources and time), *difficult to control* (it is hard to test corner-case scenarios as these are, by definition, infrequent and challenging 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 (*e.g.* virtual robots) 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 – they can run orders of magnitude faster than real-time and can be parallelized over a cluster; training in simulation is safe, cheap, and enables fair comparison

*Denotes equal contribution.

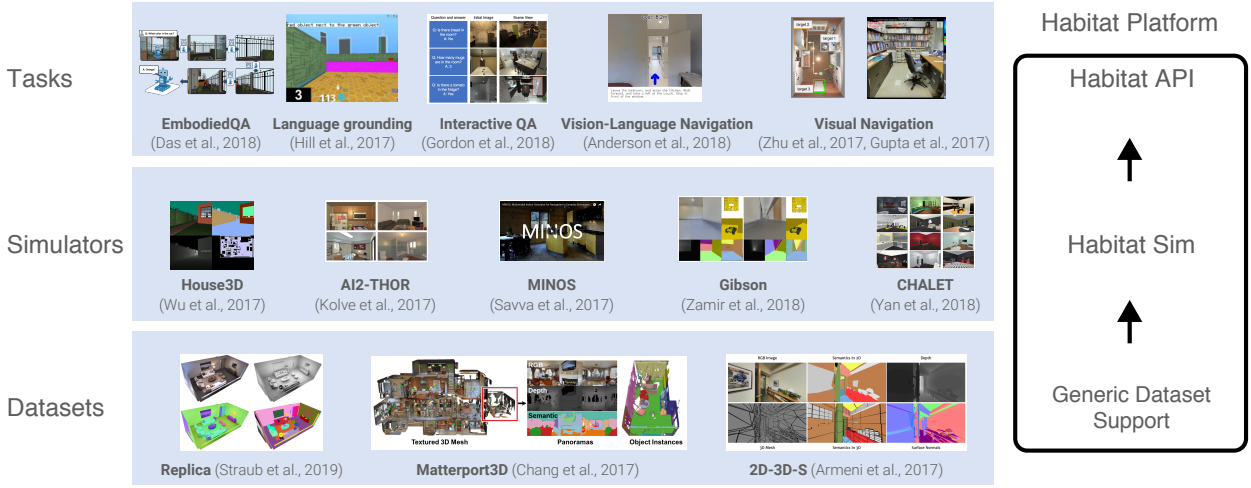

Figure 1: The 'software stack' for training embodied agents involves (1) *datasets* providing 3D assets with semantic annotations, (2) *simulators* that render these assets and within which an embodied agent may be simulated, and (3) *tasks* that define evaluatable problems that enable us to benchmark 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.

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 [6, 15].

Datasets have been a key driver of progress in computer vision, NLP, and other areas of AI [10, 18, 4, 1]. 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 Matterport3D, Gibson, and Replica 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 [2].

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 point-goal navigation [2], we make two scientific contributions:

1. We revisit the comparison between learning and SLAM approaches from two recent works [19, 16] 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} × {Matterport3D, Gibson} for multiple sensors {Blind[1], RGB, RGBD, D} × {GPS+Compass} and find that only agents with depth (D) 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 [5, 24, 8] 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 [17, 7, 13, 22, 26, 3, 27, 28, 21]. These platforms vary with respect to

[1] *Blind* refers to agents with no visual sensory inputs.

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, the existence of multiple differing 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.* GibsonEnv), and 3D dataset (*e.g.* Gibson). 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 [16] 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 two different environment datasets. A Matterport3D [8] environment is in the top row, and a Replica [25] environment in the bottom row.

3. Habitat Platform

The development of Habitat is a long-term effort to enable the formation of a common task framework [12] 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 datasets.
- **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 con-

trol 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. The sensor abstraction in Habitat allows additional sensors such as LIDAR and IMU to be easily implemented as plugins.

Generic 3D dataset API using scene graphs. `Habitat-Sim` employs a hierarchical scene graph to represent all supported 3D environment datasets, whether synthetic or based on real-world reconstructions. 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[2] 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 supplement for more details. By comparison, AI2-THOR [17] and CHALET [28] run at tens of fps, MINOS [22] and Gibson [27] run at about a hundred, and House3D [26] runs at about 300 fps. `Habitat-Sim` is 2-3 orders of magnitude faster. By operating at 10,000 frames per second we shift the bottleneck from simulation to optimization for network training. Based on TensorFlow benchmarks, many popular network architectures run at frame rates that are 10-100x lower on a single GPU[3]. In practice, we have observed that it is often *faster to generate images using `Habitat-Sim` than to load images from disk*.

[2] https://magnum.graphics/

	1 process			5 processes		
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. See the supplement for additional benchmarking results.

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.

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 [2]. 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 supplement.

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

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 classic agents as the amount of available training experience is increased.

Task definition. We use the PointGoal task (as defined by Anderson *et al*. [2]) 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 only use its sensory input to navigate. However, in the course of 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. Habitat supports noisy actuations but experiments in this paper are conducted in the noise-free setting as our analysis focuses on other factors.

Collision dynamics. Some previous works [3] use a coarse irregular navigation graph where an agent effectively 'teleports' from one location to another (1-2m apart). Others [9] 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[4] 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 [2] 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 [16, 19, 13] 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 and compass – *i.e*., they have access to their location coordinates, and implicitly their orientation relative to the goal position.

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. This threshold significantly exceeds the number of steps an optimal agent requires to reach all goals (see the supplement). After each action, the agent receives a set of observations from the active sensors.

Evaluation. A 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 [2]. 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 naviga-

[4] Up to machine precision.

tion episode-datasets for Matterport3D [8] and Gibson [27] scenes. For Matterport3D we followed the publicly available train/val/test splits. Note that as in recent works [9, 19, 16], there is no overlap between train, val, and test scenes. For Gibson scenes, we obtained textured 3D surface meshes from the Gibson authors [27], 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 supplement 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 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 or equal to the Euclidean distance). If the ratio is nearly 1, there are few obstacles and the episode is easy; if the ratio is much 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 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 near-straight-line episodes (episodes with a ratio in $[1, 1.1]$) – from 37% to 10% for the Gibson dataset generation. This step was not performed in any previous studies. We find that without this filtering, 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.

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 [23]. 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} (see supplement for details). 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:

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

  In our experiments s is set to 10 and λ is set to -0.01. Note that rewards are only provided in training environments; the task is challenging as the agent must generalize to unseen test environments.
- **SLAM [19]** is an agent implementing a classic 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 8 (Gibson), 6 (Matterport3D) 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. We do not hardcode the `stop` action to retain generality and allow for comparison with future work that does not assume GPS inputs. 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, 16]. Training agents to 75 million steps took (in sum over all three datasets): 320 GPU-hours for `Blind`, 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 classic 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.

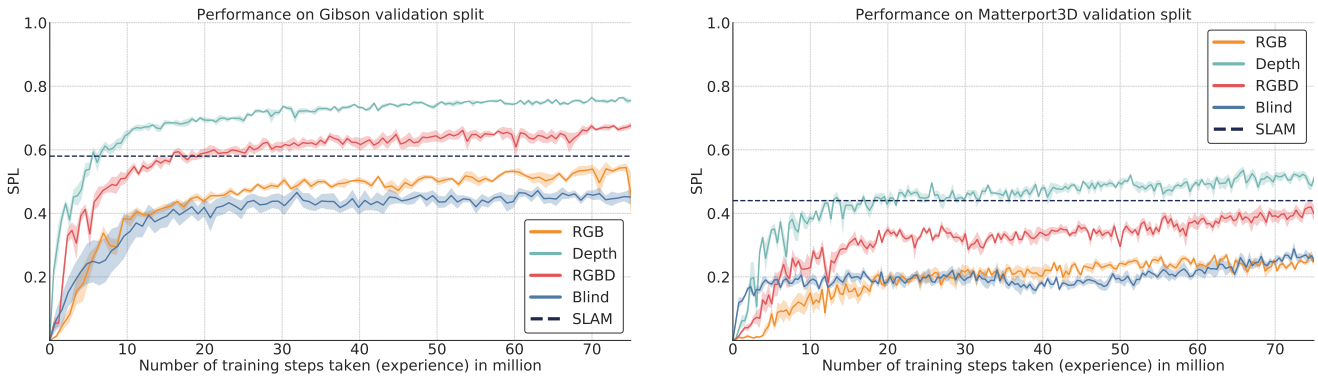

Figure 3: Average SPL of agents on the val set over the course of training. Previous work [19, 16] has analyzed performance at 5-10 million steps. Interesting trends emerge with more experience: i) `Blind` agents initially outperform `RGB` and `RGBD` but saturate quickly; ii) Learning-based `Depth` agents outperform classic SLAM. The shaded areas around curves show the standard error of SPL over five seeds.

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

Table 2: Performance of baseline methods on the PointGoal task [2] tested on the Gibson [27] and MP3D [8] test sets under multiple sensor configurations. RL models have been trained for 75 million steps. We report average rate of episode success and SPL [2].

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 classic 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] 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 4 and video in supplement) 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.

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 the datasets, RL (PPO) `Depth` performs 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 `SPL`) in that order (see the supplement for additional experiments involving noisy depth). We believe `Depth` performs better than `RGBD` because i) the PointGoal navigation task requires reasoning only about free space and depth provides relevant information directly, and ii) `RGB` has significantly more entropy (different houses look very different), thus it is easier to overfit when using `RGB`. We ran our experiments with 5 random seeds per run, to confirm that these differences are statistically significant. The differences are about an order of magnitude larger than the standard deviation of average SPL for all cases (*e.g.* on the Gibson dataset errors are, `Depth`: ± 0.015, `RGB`: ± 0.055, `RGBD`: ± 0.028, `Blind`: ± 0.005). Random and forward-only agents have very low performance, while the hand-coded goal follower and `Blind` baseline see modest performance.See the supplement for additional analysis of trained agent behavior.

In Figure 4 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 walls, while `Depth` is the most efficient. See the supplement and the video for more example trajectories.

Generalization across datasets. Our findings so far are that RL (PPO) agents significantly outperform SLAM [19]. This prompts our second question – are these findings

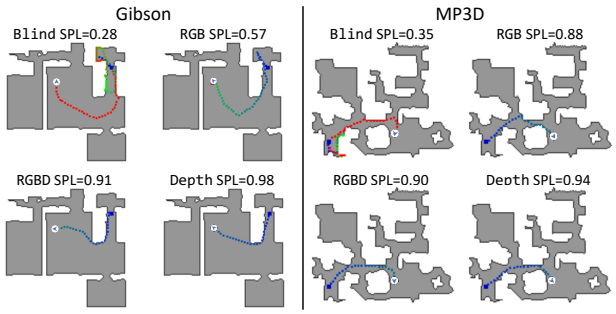

Figure 4: Navigation examples for different sensory configurations of the RL (PPO) agent, visualizing trials from the Gibson and MP3D val sets. A **blue dot** and **red dot** indicate the starting and goal positions, and the **blue arrow** indicates final agent position. The **blue-green-red line** is the agent's trajectory. Color shifts from blue to red as the maximum number of agent steps is approached. See the supplemental materials for more example trajectories.

		Gibson	MP3D
Blind	Gibson	0.42	0.34
	MP3D	0.28	0.25
RGB	Gibson	0.46	0.40
	MP3D	0.25	0.30
Depth	Gibson	0.79	0.68
	MP3D	0.56	0.54
RGBD	Gibson	0.70	0.53
	MP3D	0.44	0.42

Figure 5: 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.

dataset specific or do learned agents generalize across datasets? We report exhaustive comparisons in Figure 5 – specifically, average `SPL` for all combinations of {train, test} × {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 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). `RGB` and `RGBD` agents suffer a significant performance degradation, while the `Blind` agent is least affected (as we would expect).

Second, we find a potentially counter-intuitive trend – agents trained on Gibson consistently outperform their counterparts trained on Matterport3D, *even when evaluated on Matterport3D*. We believe the reason is the previously noted observation that Gibson scenes are smaller and episodes are shorter (lower `GDSP`) than Matterport3D. 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 insights are enabled by the engineering of Habitat, which made these experiments as simple as a change in the evaluation dataset name.

6. 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.

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 classic 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. We thank the reviewers for their helpful suggestions. The Habitat project would not have been possible without the support and contributions of many individuals. We are grateful to Angel Xuan Chang, Devendra Singh Chaplot, Xinlei Chen, Georgia Gkioxari, Daniel Gordon, Leonidas Guibas, Saurabh Gupta, Jerry (Zhi-Yang) He, Rishabh Jain, Or Litany, Joel Macey, Dmytro Mishkin, Marcus Rohrbach, Amanpreet Singh, Yuandong Tian, Yuxin Wu, Fei Xia, Deshraj Yadav, Amir Zamir, and Jiazhi Zhang for their help.

Licenses for referenced datasets.
Gibson: https://storage.googleapis.com/gibson_material/Agreement%20GDS%2006-04-18.pdf
Matterport3D: http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf.

References

[1] Phil Ammirato, Patrick Poirson, Eunbyung Park, Jana Košecká, and Alexander C Berg. A dataset for developing and benchmarking active vision. In *ICRA*, 2017.

[2] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018.

[3] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.

[4] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *ICCV*, 2015.

[5] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3D semantic parsing of large-scale indoor spaces. In *CVPR*, 2016.

[6] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. In *ICRA*, 2019.

[7] Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron C. Courville. HoME: A household multimodal environment. *arXiv:1711.11017*, 2017.

[8] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *International Conference on 3D Vision (3DV)*, 2017.

[9] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

[12] David Donoho. 50 years of data science. In *Tukey Centennial Workshop*, 2015.

[13] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 2019.

[16] Noriyuki Kojima and Jia Deng. To learn or not to learn: Analyzing the role of learning for navigation in virtual environments. *arXiv:1907.11770*, 2019.

[17] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv:1712.05474*, 2017.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[19] Dmytro Mishkin, Alexey Dosovitskiy, and Vladlen Koltun. Benchmarking classic and learned navigation in complex 3D environments. *arXiv:1901.10915*, 2019.

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

[21] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. VirtualHome: Simulating household activities via programs. In *CVPR*, 2018.

[22] Manolis Savva, Angel X. Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*, 2017.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[24] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2017.

[25] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv:1906.05797*, 2019.

[26] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3D environment. *arXiv:1801.02209*, 2018.

[27] Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, 2018.

[28] Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. CHALET: Cornell house agent learning environment. *arXiv:1801.07357*, 2018.