Habitat: A Platform for Embodied AI Research Supplemental Materials

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

1. 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 Matterport3D [3], Gibson [6], 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 [4]), 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 which 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 1. 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.
- 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`.

The performance of the simulation backend surpasses that of prior work operating on realistic reconstruction datasets by a large margin. Table 1 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. In addition, by employing OpenGL-CUDA interoperation we enable direct sharing of rendered image frames with ML

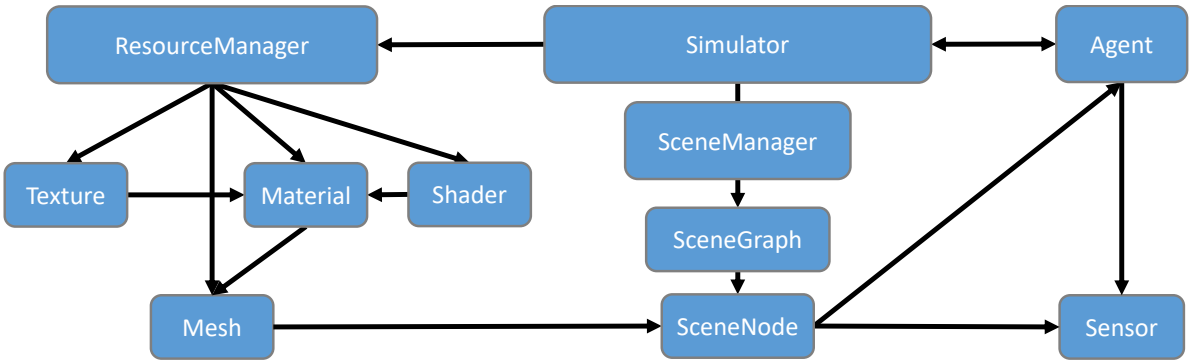


Figure 1: Architecture of `Habitat-Sim` main classes. The Simulator delegates management of all resources related to 3D environments to a ResourceManager that is responsible for loading and caching 3D environment data from a variety of on-disk formats. These resources are used within SceneGraphs at the 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.

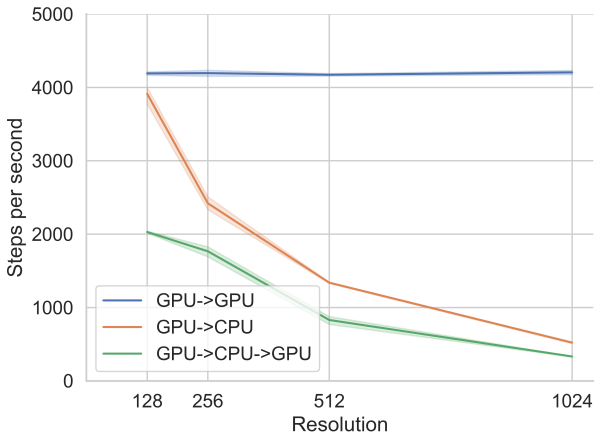


Figure 2: Performance of `Habitat-Sim` under different sensor frame memory transfer strategies for increasing image resolution. We see that 'GPU->GPU' is unaffected by image resolution while other strategies degrade rapidly.

frameworks such as PyTorch without a measurable impact on performance as the image resolution is increased (see Figure 2).

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 3, 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). 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())
```

[1]Note: The semantic sensor in Matterport3D 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.

	GPU→CPU→GPU			GPU→CPU			GPU→GPU		
Sensors / number of processes	1	3	5	1	3	5	1	3	5
RGB	2,346	6,049	7,784	3,919	8,810	11,598	4,538	8,573	7,279
RGB + depth	1,260	3,025	3,730	1,777	4,307	5,522	2,151	3,557	3,486
RGB + depth + semantics[1]	378	463	470	396	465	466	464	455	453

Table 1: 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 a frame resolution of 128x128, under different frame memory transfer strategies and with a varying number of concurrent simulator processes sharing the GPU. 'GPU-CPU-GPU' indicates passing of rendered frames from OpenGL context to CPU host memory and back to GPU device memory for use in optimization, 'GPU-CPU' only reports copying from OpenGL context to CPU host memory, whereas 'GPU-GPU' indicates direct sharing through OpenGL-CUDA interoperation.

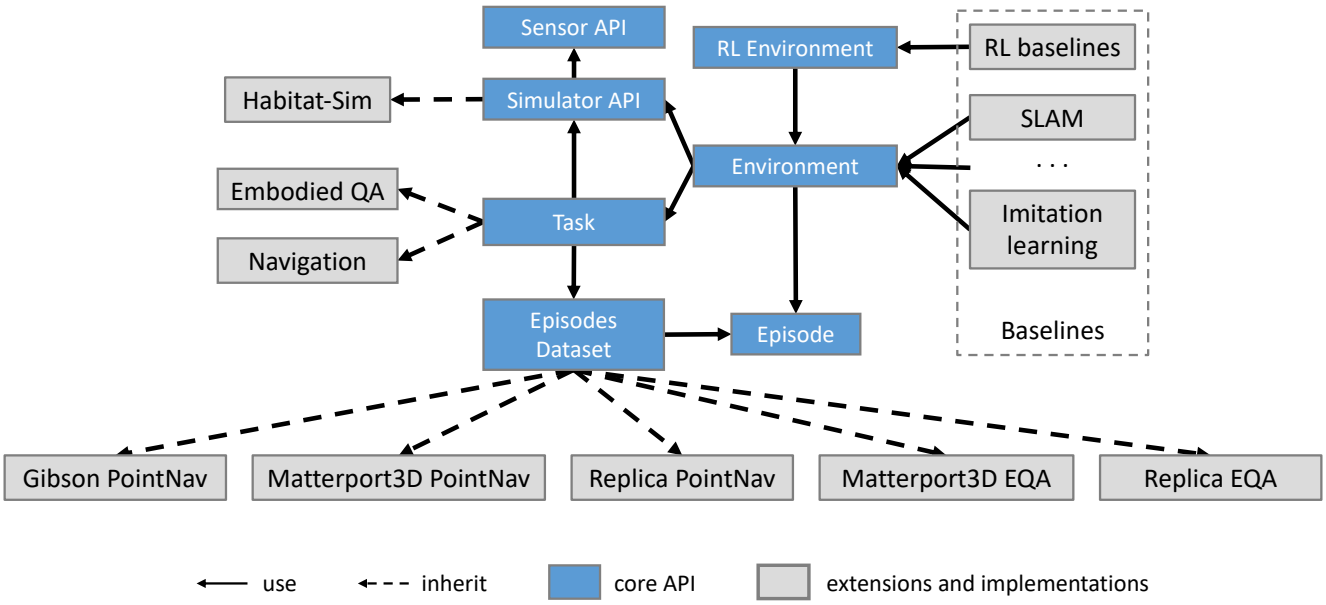


Figure 3: Architecture of `Habitat-API`. The core functionality defines fundamental building blocks such as the API for interacting with the simulator backend and receiving observations through `Sensors`. Concrete simulation backends, 3D datasets, and embodied agent baselines are implemented as extensions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

2. Additional Dataset Statistics

In Table 3 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 Matterport3D ones. Figure 4 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 Matterport3D dataset.

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

Table 2: Statistics of the PointGoal navigation datasets that we precompute for the Matterport3D and Gibson 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.

3. Additional Experimental Results

In order to confirm that the trends we observe for the experimental results presented in the paper hold for much larger amounts of experience, we scaled our experiments to 800M steps. We found that (1) the ordering of the visual

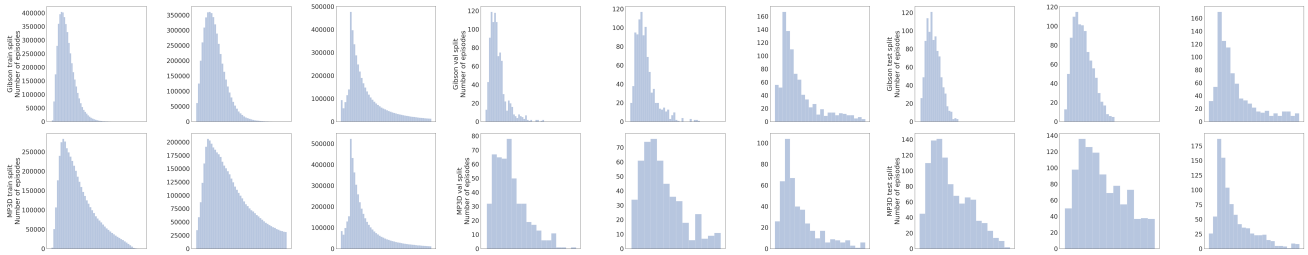


Figure 4: Statistics of PointGoal navigation episodes. From left: distribution over Euclidean distance between start and goal, distribution over geodesic distance along shortest path between start and goal, and distribution over the ratio of geodesic to Euclidean distance.

Dataset	Min	Median	Mean	Max
Matterport3D	18	90.0	97.1	281
Gibson	15	60.0	63.3	207

Table 3: Statistics of path length (in actions) for an oracle which greedily fits actions to follow the negative of geodesic distance gradient on the PointGoal navigation validation sets. This provides expected horizon lengths for a near-perfect agent and contextualizes the decision for a max-step limit of 500.

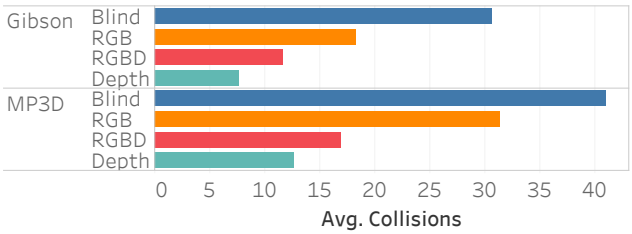


Figure 5: Average number of collisions during successful navigation episodes for the different sensory configurations of the RL (PPO) baseline agent on test set episodes for the Gibson and Matterport3D datasets. The `Blind` agent experiences the highest number of collisions, while agents possessing depth sensors (`Depth` and `RGBD`) have the fewest collisions on average.

inputs stays `Depth` > `RGBD` > `RGB` > `Blind`; (2) `RGB` is consistently better than `Blind` (by 0.06/0.03 `SPL` on Gibson/Matterport3D), and (3) `RGBD` outperforms SLAM on Matterport3D (by 0.16 `SPL`).

3.1. Analysis of Collisions

To further characterize the behavior of learned agents during navigation we plot the average number of collisions in Figure 5. 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.

3.2. Noisy Depth

To investigate the impact of noisy depth measurements on agent performance, we re-evaluated depth agents (without re-training) on noisy depth generated using a simple noise model: iid Gaussian noise ($\mu = 0$, $\sigma = 0.4$) at each pixel in inverse depth (larger depth = more noise). We observe a drop of 0.13 and 0.02 SPL for depth-RL and SLAM on Gibson-val (depth-RL still outperforms SLAM). Note that SLAM from [5] utilizes ORB-SLAM2, which is quite robust to noise, while depth-RL was trained without noise. If we increase σ to 0.1, depth-RL gets 0.12 SPL whereas SLAM suffers catastrophic failures.

4. Gibson Dataset Curation

We manually curated the full dataset of Gibson 3D textured meshes [6] to select meshes that do not exhibit significant reconstruction artifacts such as holes or texture quality 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 6) 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.

5. Reproducing Experimental Results

Our experimental results can be reproduced using the habitat-sim (commit d383c20) and `Habitat-Sim` (commit ec9557a) repositories. The code for running experiments is present under the folder `habitat-api/habitat_baselines`. Below is the shell script we used for our RL experiments:

```
# Note: parameters in {} are experiment specific.
# Note: use 8, 6 processes for Gibson, MP3D
#       respectively.

python habitat_baselines/train_ppo.py \
    --sensors {RGB_SENSOR,DEPTH_SENSOR} \
    --blind {0,1} --use-gae --lr 2.5e-4 \
    --clip-param 0.1 --use-linear-lr-decay \
    --num-processes {8,6} --num-steps 128 \
    --num-mini-batch 4 --num-updates 135000 \
    --use-linear-clip-decay \
```

For running SLAM please refer to habitat-api/habitat_baselines/slambased.

6. Example Navigation Episodes

Figure 7 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).

References

[1] 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. 1

[2] 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. 1

[3] 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. 1

[4] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018. 1

[5] Dmytro Mishkin, Alexey Dosovitskiy, and Vladlen Koltun. Benchmarking classic and learned navigation in complex 3D environments. *arXiv:1901.10915*, 2019. 4

[6] 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. 1, 4

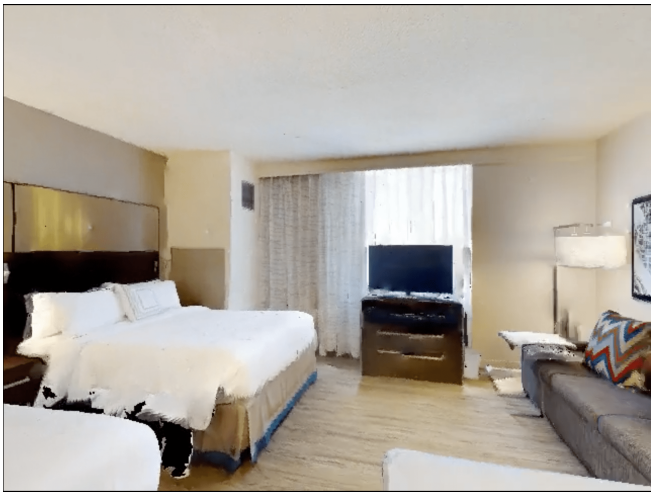
0: critical reconstruction artifacts, holes, or texture issues

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 6: 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.

Figure 7: Additional navigation example episodes for the 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 allowed agent steps is approached.

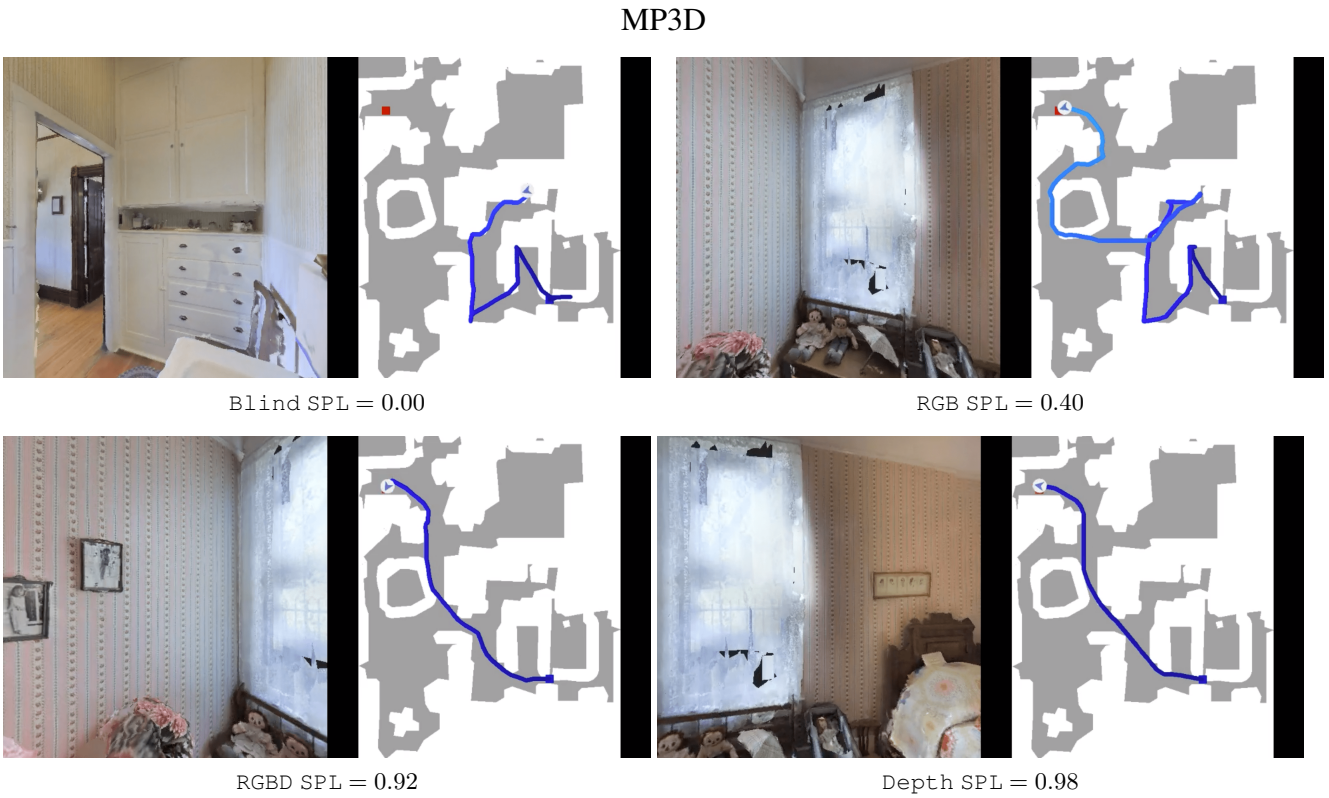


Figure 7: Additional navigation example episodes for the different sensory configurations of the RL (PPO) agent, visualizing trials from the Gibson and MP3D val sets. A **blue dot** and **red dot** indicate the starting and goal positions, and the **blue arrow** indicates final agent position. The **blue-green-red line** is the agent's trajectory. Color shifts from blue to red as the maximum number of allowed agent steps is approached.