Embodied Question Answering in Photorealistic Environments with Point Cloud Perception

Erik Wijmans[1†], Samyak Datta[1†], Oleksandr Maksymets[2†], Abhishek Das[1],
Georgia Gkioxari[2], Stefan Lee[1], Irfan Essa[1], Devi Parikh[1,2], Dhruv Batra[1,2]
[1]Georgia Institute of Technology [2]Facebook AI Research
[1]{etw, samyak, abhshkdz, steflee, irfan, parikh, dbatra}@gatech.edu
[2]{maksymets, gkioxari}@fb.com

Abstract

To help bridge the gap between ***internet vision****-style problems and the goal of* ***vision for embodied perception*** *we instantiate a large-scale navigation task – Embodied Question Answering [1] in photo-realistic environments (Matterport 3D). We thoroughly study navigation policies that utilize 3D point clouds, RGB images, or their combination. Our analysis of these models reveals several key findings. We find that two seemingly naive navigation baselines, forward-only and random, are strong navigators and challenging to outperform, due to the specific choice of the evaluation setting presented by [1]. We find a novel loss-weighting scheme we call Inflection Weighting to be important when training recurrent models for navigation with behavior cloning and are able to out perform the baselines with this technique. We find that point clouds provide a richer signal than RGB images for learning obstacle avoidance, motivating the use (and continued study) of 3D deep learning models for embodied navigation.*

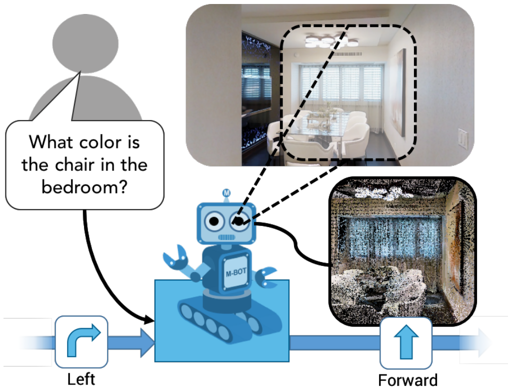


Figure 1: We extend EmbodiedQA [1] to *photorealstic* environments, our agent is spawned in a *perceptually* and *semantically* novel environment and tasked with answering a question about that environment. We examine the agent's ability to navigate the environment and answer the question by perceiving its environment through point clouds, RGB images, or a combination of the two.

1. Introduction

Imagine asking a home robot *'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 an agent would need a range of artificial intelligence (AI) skills – visual perception (to recognize objects, scenes, obstacles), language understanding (to translate questions and instructions into actions), and navigation of potentially novel environments (to move and find things in a changing world). Much of the recent success in these areas is due to large neural networks trained on massive human-annotated datasets collected from the web. However, this static paradigm of *'internet vision'* is poorly suited for training embodied agents. By their nature, these agents engage in *active perception* – observing the world and then performing actions that in turn dynamically change what the agent perceives. What are needed then are richly annotated, photo-realistic environments where agents may learn about the consequence of their actions on future perceptions while performing high-level goals.

To this end, a number of recent works have proposed goal-driven, perception-based tasks situated in simulated environments to develop such agents [1–10]. While these tasks are set in semantically realistic environments (*i.e.* having realistic layouts and object occurrences), most are based in synthetic environments (on SUNCG [11] or Unity 3D models [12]) that are perceptually quite different from what agents embodied in the real world might experience. Firstly, these environments lack visual realism both in terms of the fidelity of textures, lighting, and object geometries but also with respect to the rich in-class variation of ob-

[†] denotes equal contribution

jects[1]. Secondly, these problems are typically approached with 2D perception (RGB frames) despite the widespread use of depth-sensing cameras (RGB-D) on actual robotic platforms [13–15].

Contributions. We address these points of disconnect by instantiating a large-scale, language-based navigation task in photorealistic environments and by developing end-to-end trainable models with point cloud perception – from raw 3D point clouds to goal-driven navigation policies.

Specifically, we generalize the recently proposed Embodied Question Answering (EmbodiedQA) [1] task (originally proposed in synthetic SUNCG scenes [11]) to the photorealistic 3D reconstructions from Matterport 3D (MP3D) [16]. In this task, an agent is spawned at a random location in a novel environment (*e.g*. a house) and asked to answer a question (*'What color is the car in the garage?'*). In order to succeed, the agent needs to navigate from egocentric vision alone (without an environment map), locate the entity in question (*'car in the garage'*), and respond with the correct answer (*e.g*. *'orange'*).

We introduce the MP3D-EQA dataset, consisting of 1136 questions and answers grounded in 83 environments. Similar to [1], our questions are generated from functional programs operating on the annotations (objects, rooms, and their relationships) provided in MP3D; however, MP3D lacks color annotations for objects, which we collect from Amazon Mechanical Turk in order to generate *'What color . . . ?'* questions. The MP3D environments provide significantly more challenging environments for our agent to learn to navigate in due to the increased visual variation.

We present a large-scale exhaustive evaluation of design decisions, training a total of 16 navigation models (2 architectures, 2 language variations, and 4 perception variations), 3 visual question answering models, and 2 perception models – ablating the effects of perception, memory, and goal-specification. Through this comprehensive analysis we demonstrate the complementary strengths of these perception modalities and highlight surprisingly strong baselines in the EmbodiedQA experimental setting.

Our analysis reveals that the seemingly naive baselines, forward-only and random, are strong navigators in the default evaluation setting presented in [1] and challenging to beat, providing insight to others working in this space that models can perform surprisingly well without learning any meaningful behavior. We also find that point clouds provide a richer signal than RGB images for learning obstacle avoidance, motivating continued study of utilizing depth information in embodied navigation tasks.

We find a novel weighting scheme we call *Inflection Weighting* – balancing the contributions to the cross-entropy loss between *inflections*, where the ground truth action differs from the previous one, and non-inflections – to be an effective technique when performing behavior cloning with a shortest path expert. We believe this technique will be broadly useful any time a recurrent model is trained on long sequences with an imbalance in symbol continuation versus symbol transition probabilities, *i.e.* when $P(X_t = x \mid X_{t-1} = x) >> P(X_t \neq x \mid X_{t-1} = x)$.

To the best of our knowledge, this is the first work to explore end-to-end-trainable 3D perception for goal-driven navigation in photo-realistic environments. With the use of point clouds and realistic indoor scenes, our work lays the groundwork for tighter connection between embodied vision and goal-driven navigation, provides a testbed for benchmarking 3D perception models, and hopefully brings embodied agents trained on simulation one step closer to real robots equipped with 2.5D RGB-D cameras.

2. Related Work

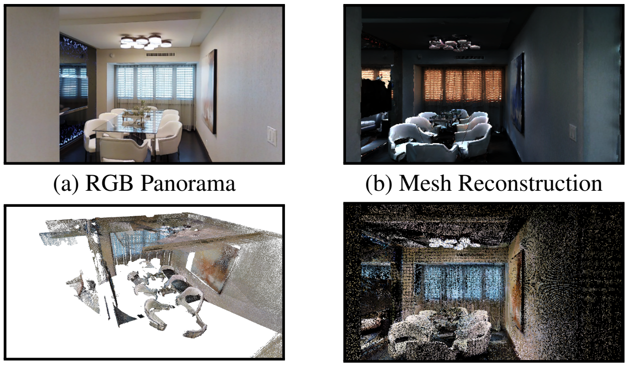
Embodied Agents and Environments. End-to-end learning methods – to predict actions directly from raw pixels [17] – have recently demonstrated strong performance. Gupta *et al*. [2] learn to navigate via mapping and planning. Sadeghi *et al*. [18] teach an agent to fly using simulated data and deploy it in the real world. Gandhi *et al*. [19] collect a dataset of drone crashes and train self-supervised agents to avoid obstacles. A number of new challenging tasks have been proposed including instruction-based navigation [6, 7], target-driven navigation [2, 4], embodied/interactive question answering [1, 9], and task planning [5].

A prevailing problem in embodied perception is the lack of a standardized, large-scale, diverse, real-world benchmark – essentially, there does not yet exist a COCO [20] for embodied vision. A number of synthetic 3D environments have been introduced, such as DeepMind Lab [21] and VizDoom [22]. Recently, more visually stimulating and complex datasets have emerged which contain actionable replicas of 3D indoor scenes [3, 23–25]. These efforts make use of synthetic scenes [25, 26], or scans of real indoor houses [16, 27] and are equipped with a variety of input modalities, *i.e*. RGB, semantic annotations, depth, *etc*.

The closest to our work is the EmbodiedQA work of Das *et al*. [1], who train agents to predict actions from egocentric RGB frames. While RGB datasets are understandably popular for 'internet vision', it is worth stepping back and asking – *why must an embodied agent navigating in 3D environments be handicapped to perceive with a single RGB camera?* We empirically show that point cloud representations are more effective for navigation in this task. Moreover, contrary to [1, 9] that use synthetic environments, we extend the task to real environments sourced from [16].

3D Representations and Architectures. Deep learning has been slower to impact 3D computer vision than its 2D counterpart, in part due to the increased complexity of representing 3D data. Initial success was seen with volumetric

[1] To pervert Tolstoy, each ugly lamp is ugly in its own way.



(a) RGB Panorama

(b) Mesh Reconstruction

(c) Point Cloud

(d) RGB-D Render

Figure 2: Illustration of mesh construction errors and what point clouds are able to correct. Notice the warping of flat surfaces, the extreme differences in color, and texture artifacts from reflections.

CNN's [28–30]. These networks first discretize 3D space with a volumetric representation and then apply 3D variants of operations commonly found in 2D CNN's – convolutions, pooling, etc. Volumetric representations are greatly limited due to the sparsity of 3D data and the computational cost of 3D convolutions. Recent works on 3D deep learning have proposed architectures that operate directly on point clouds. Point clouds are a challenging input for deep learning as they are naturally a set of points with no canonical ordering. To overcome the ordering issue, some utilize symmetric functions, PointNet(++) [31, 32], and A-SCN [33]. Others have used clever internal representations, such as SplatNet [34], Kd-Net [35], and O-CNN [36].

3. Questions in Environments

In this work, we instantiate the Embodied Question Answering (EQA) [1] task in realistic environments from the Matterport3D dataset [16].

3.1. Environments

The Matterport3D dataset consists of 90 home environments captured through a series of panoramic RGB-D images taken by a Matterport Pro Camera (see sample panoramas in Fig. 2a). The resulting point clouds are aligned and used to reconstruct a 3D mesh (like those shown in Fig. 2b) that is then annotated with semantic labels. The Matterport3D dataset is densely annotated with semantic segmentations of 40 object categories for ∼50,000 instances. Room type is annotated for over 2050 individual rooms.

These reconstructions offer high degrees of perceptual realism but are not perfect however and sometimes suffer from discoloration and unusual geometries such as holes in surfaces. In this work, we examine both RGB and RGB-D perception in these environments. For RGB, we take renders from the mesh reconstructions and for point clouds we operate directly on the aligned point clouds. Fig. 2c and Fig. 2d show the point cloud rendered for an agent looking at the scene shown in Fig. 2a.

Simulator. To enable agents to navigate in MatterPort3D environments, we develop a simulator based on MINOS [23]. Among other things, MINOS provides occupancy checking, RGB frame rendering from the mesh, and shortest path calculation (though we reimplement this for higher accuracy and speed). It does not however provide access to the underlying point clouds. In order to render 2.5D RGB-D frames, we first construct a global point cloud from all of the panoramas provided in an environment from the dataset. Then, the agent's current position, camera parameters (field of view, and aspect ratio), and the mesh reconstruction are used to determine which points are within its view. See the supplementary for full details on this.

3.2. Questions

Following [1], we programmatically generate templated questions based on the Matterport3D annotations, generating questions of the following three types:

location: *What room is the* <OBJ> *located in?*
color: *What color is the* <OBJ> *?*
color_room: *What color is the* <OBJ> *in the* <ROOM> *?*

While EQA [1] included a forth question type **prepositions**, we found those questions in MP3D to be relatively few, with strong biases in their answer, thus we do not include them.

While room and object annotations and positions supporting the three question types above are available in MP3D, human names for object colors are not. To rectify this, we collect the dominant color of each object from workers on Amazon Mechanical Turk (AMT). Workers are asked to select one of 24 colors for each object. The color palette was created by starting with Kenneth Kelly's 22 colors of maximum contrast [37] and adding 2 additional colors (off-white and slate-grey) due to their prevalence in indoor scenes. Overall, the most reported color was gray. For each object, we collect 5 annotations and take the majority vote, breaking ties based on object color priors. We include details of the AMT interface in the supplementary.

Following the protocol in [1], we filter out questions that have a low entropy in distribution over answers across environments *i.e*. peaky answer priors – *e.g*. the answer to *'What room is the shower in?'* is nearly always *'bathroom'* – to ensure that questions in our dataset require the agent to navigate and perceive to answer accurately. We remove rooms or objects that are ambiguous (*e.g*. "misc" rooms) or structural (*e.g*. "wall" objects). Below are the objects and rooms that appear in our generated questions:

Objects: *shelving, picture, sink, clothes, appliances, door, plant, furniture, fireplace, chest of drawers, seating, sofa, table, curtain, shower, towel, cushion, blinds, counter, stool, bed, chair, bathtub, toilet, cabinet*

	Homes	Floors	Total Qns.	Unique Qns.
train	57	102	767	174
val	10	16	130	88
test	16	28	239	112

Table 1: Statistics of splits for EQA in Matterport3D

Rooms: *family room, closet, spa, dinning room, lounge, gym, living room, office, laundry room, bedroom, foyer, bathroom, kitchen, garage, rec room, meeting room, hallway, tv room*

In total, we generate ∼1100 questions across 83 home environments (7 environments resulted in no questions after filtering). Note that this amounts to ∼13 question-answer pairs per environment compared to ∼12 per scene in [1]. Color_room questions make up the majority of questions. These questions require searching the environment to find the specified object in the specified room. Whereas [1] requires both the object and the room to be unique within the environment, we only require the (object, room) pair to be unique, thereby giving the navigator significantly less information about the location of the object.

We use the same train/val/test split of environments as in MINOS [23]. Note that in [1], the test environments differ from train only in the layout of the objects; the objects themselves have been seen during training. In MP3D-EQA, the agents are tested on entirely new homes, thus may come across entirely new objects – testing semantic *and* perceptual generalization. Tab. 1 shows the distribution of homes, floors, and questions across these splits. We restrict agent start locations to lie on the same floor as question targets and limit episodes to single floors.

4. Perception for Embodied Agents

Agents for EmbodiedQA must understand the given question, perceive and navigate their surroundings collecting information, and answer correctly in order to succeed. Consider an EmbodiedQA agent that navigates by predicting an action a_t at each time step t based on its trajectory of past observations and actions $\sigma_{t-1} = (s_1, a_1, s_2, a_2, \ldots, s_{t-1}, a_{t-1})$, the current state s_t, and the question Q. There are many important design decisions for such a model – action selection policy, question representation, trajectory encoding, and observation representation. In this work, we focus on the observation representation – *i.e.* perception – in isolation and follow the architectural pattern in [1] for the remaining components. In this section, we describe our approach and recap existing model details.

4.1. Learning Point Cloud Representations

Consider a point cloud $P \in \mathcal{P}$ which is an unordered set of points in 3D space with associated colors, *i.e.* $P = \{(x_m, y_m, z_m, R_m, G_m, B_m)\}_{m=1}^{M}$. To enable a neural agent to perceive the world using point clouds, we must learn a function $f : \mathcal{P} \rightarrow \mathbb{R}^d$ that maps a point cloud to an observation representation. To do this, we leverage a widely used 3D architecture, PointNet++ [32].

PointNet++. At a high-level, PointNet++ alternates between spatial clustering and feature summarization – resulting in a hierarchy of increasingly coarse point clusters with associated feature representations summarizing their members. This approach draws an analogy to convolution and pooling layers in standard convolutional neural networks.

More concretely, let $\{p_1^i, ..., p_{N_i}^i\}$ be the set of N_i points at the i^{th} level of a PointNet++ architecture and $\{h_1^i, ..., h_{N_i}^i\}$ be their associated feature representations (*e.g.* RGB values for the input level). To construct the $i+1^{\text{th}}$ level, N_{i+1} centroids $\{p_1^{i+1}, ..., p_{N_{i+1}}^{i+1}\}$ are sampled from level i via iterative farthest point sampling (FPS) – ensuring even representation of the previous layer. These centroids will make up the points in level $i+1$ and represent their local areas. For each centroid p_k^{i+1}, the K closest points within a max radius are found and a symmetric learnable neural architecture [31], composed of a series of per-point operations (essentially 1-by-1 convolutions) and a terminal max-pool operation, is applied to this set of associated points to produce the summary representation h_k^{i+1}. These clustering and summarization steps (referred to as Set Abstractions in [32]) can be repeated arbitrarily many times. In this work we use a 3 level architecture with $N^1 = 1045$, $N^2 = 256$, and $N^3 = 64$. We compute a final feature with a set of 1-by-1 convolutions and a max-pool over the 3rd level point features and denote this network as $f(\cdot)$.

Given an input point cloud P_t from an agent's view at time t, we produce a representation $s_t = f(P_t)$ where $s_t \in \mathbb{R}^{1024}$. However, point clouds have an interesting property – as an agent navigates an environment the number of points it perceives can vary. This may be due to sensor limitations (*e.g.* being too close or too far from objects) or properties of the observed surfaces (*e.g.* specularity). While the encoder f is invariant to the number of input points, representations drawn from few supporting points are not likely to be good representations of a scene. For a navigation or question-answering agent, this means there is no way to discern between confident and unconfident or erroneous observation. To address this, we divide the range spanning the possible number of points in any given point cloud – $[0, 2^{14}]$ – into 5 equal sized bins and represent these bins as 32-d feature vectors that encode the sparsity of a point cloud. Now, given a point cloud P_t with $|P_t|$ points, we retrieve its corresponding sparsity embedding c_t and produce a final encoding $[s_t, c_t] \in \mathbb{R}^{1056}$ that is used by the agent for navigation and question-answering.

Visual Pretraining Tasks. To train the encoder architecture to extract semantically and spatially meaningful representations of agent views, we introduce three pretraining tasks based on the annotations provided in Matterport3D. Specifically, these tasks are:

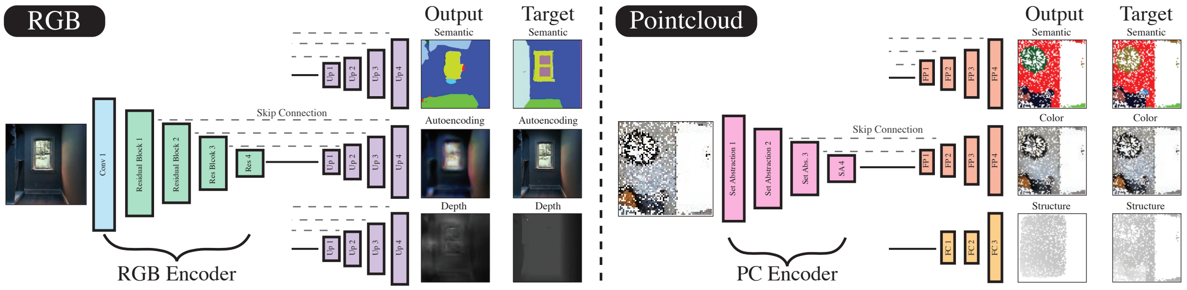


Figure 3: The visual encoders a trained using three pertaining tasks to imbue their scene representations with information about semantics (segmentation), color (autoencoding), and structure (depth). All decoder heads share the same encoder. Upsampling for RGB (Up #) is done with bi-linear interpolation. Upsampling for pointclouds (FP #), is achieved with Feature Propagation layers [32]. After pretraining, the decoders are discarded, and the encoder is treated as a static feature extractor.

- **Semantic Segmentation** in which the model predicts the object annotation for each point, y_i^s, from the summarized representation $s_i = f(P)$. We train a PointNet++ feature propagation network $g_{\mathrm{s}}(\cdot)$ to minimize the cross-entropy between y_i^s and $\hat{y}_i^s = g_{\mathrm{s}}(f(P))$ [32]. This encourages the encoder, $f(\cdot)$, to include information about which objects are in the frame.
- **Color Autoencoding** mirrors the semantic segmentation task. However, the network $g_{\mathrm{c}}(\cdot)$ is now trained to minimize the smooth-L1 loss between y_i^c and $\hat{y}_i^c = g_{\mathrm{c}}(f(P))$. This task encourages the encoder $f(\cdot)$ to capture holistic information about the colors in the scene.
- **Structure Autoencoding** where point coordinates must be recovered from the summarized representation, *i.e.* $\{(x_i, y_i, z_i, R_i, G_i, B_i)\}_{i=1}^N \rightarrow \{(x_i, y_i, z_i)\}_{i=1}^N$. We implement this decoder as a multi-layer perceptron that regresses to the $N \times 3$ spatial coordinates. As in [38], we use the earth-movers distance as the loss function.

We demonstrate these tasks in Fig. 3. These tasks encourage the model features to represent colors, objects, and spatial information including free-space and depth that are essential to navigation. We collect ∼100,000 frames from Matterport3D using our simulator and train the point cloud encoder for these tasks. We discard the decoder networks after training, and use the encoder f as a fixed feature extractor.

RGB Image representations. We utilize ResNet50 [39] trained using an analogous set of tasks (semantic segmentation, autoencoding, and depth prediction) to learn a representation for egocentric 224×224 RGB images as in [1]. We find that ResNet50 is better able to handle the increased visual complexity of the Matterport3D environments than the shallow CNN model used in Das *et al*. We provide further details about perception model and decoder architectures in the supplement.

4.2. Navigation and Question Answering

We now provide an overview of the navigation and question answering models we use in this work.

Question Encoding. In order to succeed at navigation and question answering, it is important for an embodied agent to understand the queries it is being tasked with answering. We use two layer LSTMs with 128-d hidden states to encode questions. The question encoding for navigation and question answering are learned separately.

Question Answering Models. We experimented with three classes of question answering models:

- **Question-only** We examine the question-only baselines proposed in [1] – a small classification network that predicts the answer using just the question encoding. We also examine the recently proposed question-only baselines in [40] – a simple nearest neighbors approach and a bag-of-words with a softmax classifier.
- **Attention** This is the best performing VQA model from [1]. It computes question-guided attention over the features of the last five frames observed by the agent before stopping, followed by element-wise product between the attended feature and question encoding to answer; and
- **Spatial Attention** utilizes the bag-of-words encoder proposed in [40] to compute spatial attention over the last-frame. We use scaled dot-product attention [41] over the feature map, perform an element-wise product between attended features and the question feature, and predict an answer. This model only uses RGB inputs.

Navigation Models. We consider two baseline navigators:

- **Forward-only (Fwd)** which always predicts forward.
- **Random** which uniformly chooses one of `forward`, `turn-left`, and `turn-right` at every time step.

We consider two navigation architectures:

- **Reactive (R)** which is a simple feed-forward network that takes a concatenation of the embedding of the five most recent visual observations as input to predict an action. As such, this is a memory-less navigator.
- **Memory (M)** which is a two-layer GRU-RNN that takes the encoding(s) of the current observation and previous action as inputs to predict the current action.

For each navigation architecture, we examine the combination of our 4 different perception variations, `None` (*i.e.* a blind model as suggested by Thomason *et al.* [42]), PC, RGB, and PC+RGB, with the 2 different language variations, None and Question. For reactive models that utilize the question, we incorporate the question embedding by concatenation with the visual embedding. For memory models, the question embedding is an additional input to the GRU. Due to the highly correlated observations our agents see during training, we utilize Group Normalization layers [43] in our navigation models. The action space for all our navigation models is forward, turn-left, turn-right, and stop.

4.3. Imitation Learning from Expert Trajectories

To train our models, we first create a static dataset of agent trajectories by generating training episodes based on shortest-paths from agent spawn locations to the best view of the object of interest. For example, if a question asks *'What color is the sofa in the living room?'*, we spawn an agent randomly in the environment in the same floor as the target object – the sofa – and compute the shortest navigable path to the best view of the sofa. The best view of the sofa is determined by exhausting all possible view positions within a reasonable radius of the target object. The quality of a view is determined by the intersection over union of a pre-determined bounding box and the segmentation mask of the target. In normalized image coordinates, the bounding box's top left corner is at (0.25, 0.25) and it has a height of 0.6 and a width of 0.5. We use this metric instead of simply maximizing the number of visible pixels in the segmentation mask to maintain context of the object's surroundings.

To provide enough data to overcome the complexity of Matterport3D environments, we generate ∼11,796 such paths in total (corresponding to approximately ∼15 episodes per question-environment pair, each for a different spawn location of the agent). For computational efficiency in the large Matterport3D environments, we compute shortest paths in continuous space using LazyTheta* [44] and greedily generate agent actions to follow that path, rather than directly searching in the agent's action space.

Perception. We use the frozen pre-trained perception models as described in Section 4.1. For PC+RGB models we simply concatenate both visual features.

Question Answering. The question answering models are trained to predict the ground truth answer from a list of 53 answers using Cross Entropy loss. The models with vision use the ground-truth navigator during training.

4.4. Imitating Long Trajectories Effectively

All navigation models are trained with behavior cloning where they are made to mimic the ground truth, shortest path agent trajectories. That is to say the agents are walked through the ground truth trajectory observing the corresponding frames (though reactive models retain only the last five) up until a given time step and then make an action prediction. Regardless of the decision, the agent will be stepped along the ground truth trajectory and repeat this process. One challenge with this approach is that relatively unintelligent policies can achieve promising validation loss without really learning anything useful – one such strategy simply repeats the previous ground truth action. Insidiously, these models achieve very high validation accuracy for action prediction but miss every transition between actions!

Inflection Weighting. To combat this problem and encourage agents to focus on important decisions along the trajectory, we introduce a novel weighting scheme we call Inflection Weighting. Conceptually, we weight predictions at time steps more heavily if the ground truth action differs from the previous one – that is if the time step is an inflection point in the trajectory. More formally, we define a per-time step weight

$$w_t = \begin{cases} \frac{N}{n_I} & a_{t-1} \neq a_t \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

where N/n_I is the inverse frequency of inflection points (approximately 5.7 in our dataset). We can then write an inflection weighted loss between a sequence of predictions $\hat{Y}$ and a ground truth trajectory A over as:

$$\ell_{IW}(\hat{Y}, A) = \frac{1}{\sum_{t=1}^{T} w_t} \sum_{t=1}^{T} w_t \ell\left(\hat{y}_t, a_t\right) \tag{2}$$

where $\ell(\cdot, \cdot)$ is the task loss – cross-entropy in our setting. We define the first action, $t = 1$, to be an inflection. In practice, we find inflection weighting leads to significant gains in performance for recurrent models.

Inflection weighting may be viewed as a generalization of the class-balanced loss methods that are commonly used in supervised learning under heavily imbalanced class distributions (*e.g.* in semantic segmentation [45]) for a particular definition of a 'class' (inflection or not).

5. Experiments and Analysis

We closely follow the experimental protocol of Das *et al.* [1]. All results here are reported on novel test environments. Agents are evaluated on their performance 10, 30, or 50 primitive actions away from the question target, corresponding to distances of 0.35, 1.89, and 3.54 meters respectively. One subtle but important point is that to achieve these distances the agent is first randomly spawned within the environment, and then the agent is *walked along the shortest path to the target* until it is the desired distance from the target (10, 30, or 50 steps).

We perform an exhaustive evaluation of design decisions, training a total of 16 navigation models (2 architectures, 2 language variations, and 4 perception variations), 3 visual question answering models, and 2 perception models.

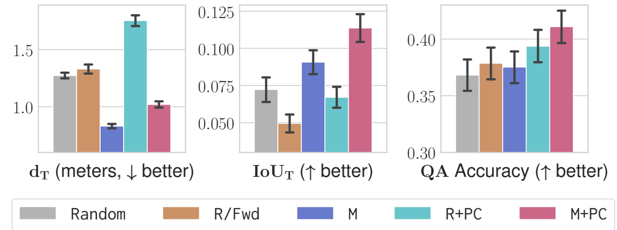


Figure 4: Models with memory significantly outperform their memory-less counterparts. Surprisingly, the baselines, random and forward-only, and a vision-less navigator with memory perform very well.

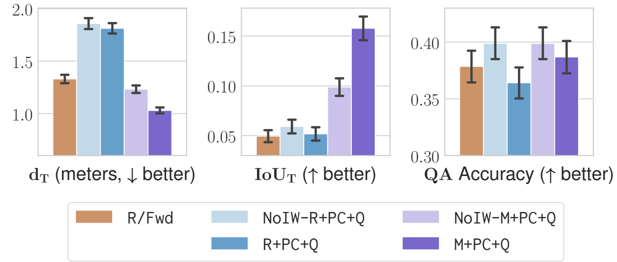


Figure 5: Models trained with inflection-weighted cross-entropy loss significantly outperform their unweighted cross-entropy counterparts and the baselines.

5.1. Metrics

Question Answering. For measuring question answering performance, we report the top-1 accuracy, *i.e.* did the agent's predicted answer match the ground truth or not.

Navigation. For navigation, we report the distance to the target object from where the agent is spawned ($\mathbf{d_0}$) for reference, measure distance to the target object upon navigation completion $\mathbf{d_T}$ (lower is better), and the percentage of actions that result in a collision with an obstacle $\%_{\mathbf{collision}}$ (lower is better). All the distances are geodesic, *i.e.* measured along the shortest path.

We propose a new metric, $\mathbf{IoU_T}$ (higher is better), to evaluate the quality of the view of the target the agent obtains at the end of navigation. We compute the intersection-over-union (IoU) score between the ground-truth target segmentation and the same centered bounding box used to select views during dataset generation (see Section 4.3). To compensate for object size, we divide by the best attainable IoU for the target object. We define $\mathbf{IoU_T}$ as the maximum of the last N IoU scores. We set N to 5 as the VQA model receives the last 5 frames.

5.2. Results and Analysis

Question Answering. The top-1 accuracy for different answering modules on the validation set using the ground-truth navigator is shown below.

	Top-1 (%)
`spatial+RGB+Q`	**46.2**
`attention+RGB+Q`	40.0
`attention+PC+RGB+Q`	38.4
`attention+PC+Q`	36.1
`lstm-question-only`	32.8
`nn-question-only` [40]	35.4
`bow-question-only` [40]	38.3

In-order to compare **QA** performance between navigators, we report all **QA** results with the best-performing module – `spatial+RGB+Q` – regardless of the navigator.

Navigation. We use the following notation to specify our models: For the base architecture, `R` denotes reactive models and `M` denotes memory models. The base architectures are then augmented with their input types, `+PC`, `+RGB`, and `+Q`. So a memory model that utilizes point clouds (but no question) is denoted as `M+PC`. Unless otherwise specified (by the prefix `NoIW`), models are trained with inflection weighting. We denote the two baseline navigators, forward-only and random, as `Fwd` and `Random`, respectively.

Due to the large volume of results, we present key findings and analysis here (with T_{-30}) and, for the intrepid reader, provide the full table (with 300+ numbers!) in the supplement. We make the following observations:

Forward-only is a strong baseline. One of the side-effects of the evaluation procedure proposed in [1] is that the agent is commonly facing the correct direction when it is handed control. This means the right thing to do to make progress is to go forward. As a result, a forward-only navigator does quite well, see Fig. 4. Forward-only also tends to not overshoot too much due to its 'functional stop': continually running into an obstacle until the max step limit is reached. Our vision-less reactive models (`R/Fwd` and `R+Q/Fwd`) learn to only predict forward, the most frequent action.

Fig. 4 also shows that the random baseline is a deceptively strong baseline. The lack of a `backward` action, and `left` and `right` cancelling each other out in expectation, results in random essentially becoming forward-only.

Inflection weighting improves navigation. We find inflection weighting to be crucial for training navigation models with behavior cloning of a shortest-path expert; see Fig. 5. While we see some improvements with inflection weighting for most models, memory models reap the greatest benefits – improving significantly on both $\mathbf{d_T}$ and $\mathbf{IoU_T}$. Interestingly, these gains do not translate into improved **QA** accuracy. While we have only utilized this loss for behavior cloning, we suspect the improvements seen from inflection weighting will transfer to models that are fine-tuned with reinforcement learning as they begin with better performance.

Memory helps. Fig. 4 shows that models with memory are better navigators than their reactive counter parts. Surpris-

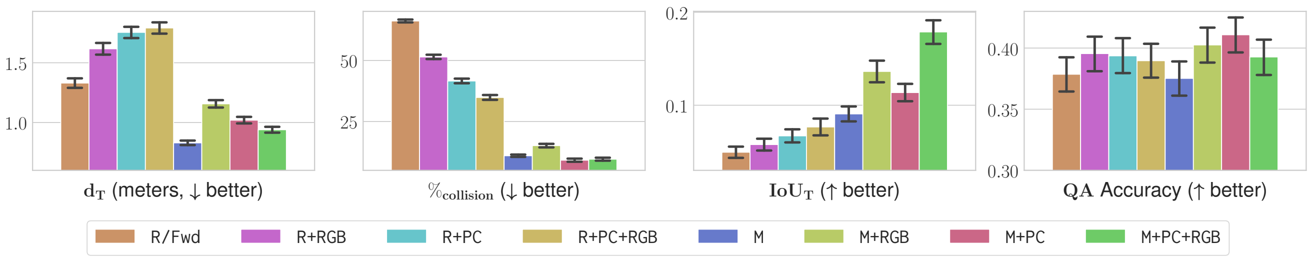


Figure 6: Vision generally hurts distance-based navigation metrics. However metrics that are dependent on the navigators ability to look in a particular direction ($\mathbf{IoU_T}$ and $\mathbf{QA}$) generally improve, and the models collide with the environment less.

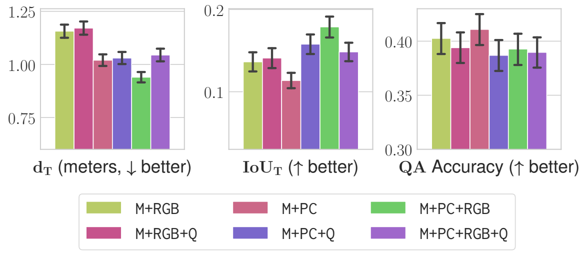


Figure 7: Comparison of memory navigation models with and without the question. Interestingly, adding the question doesn't appear to aid models trained with behavior cloning.

ingly, a vision-less navigator with memory performs very well at distance based navigation metrics. Like a vision-less reactive navigator (forward-only), a vision-less memory navigator is only able to learn priors on how shortest paths in the dataset tend to look, however memory allows the model to count and therefore it is able to stop and turn.

Vision helps gaze direction metrics. Fig. 6 shows the effect of adding vision to both reactive and memory models. The addition of vision leads to improvements on $\mathbf{IoU_T}$ and $\mathbf{QA}$, however, the improvements in $\mathbf{IoU_T}$ do not translate directly improvement on $\mathbf{QA}$. This is likely due to naive VQA models. Models with vision also tend to collide with the environment less often, as can be seen by $\%_{\mathbf{collision}}$ usually being lower.

Vision hurts distance metrics. Surprisingly, adding vision hurts distance based navigation metrics ($\mathbf{d_T}$). For reactive models, adding vision causes the models to collide significantly less frequently, resulting in a loss of the 'functional stop' that forward-only uses, *i.e*. continually colliding until the step limit is reached. For memory models, the story isn't as clear; however, memory models with vision stop less often and thus have a higher average episode length than their vision-less counterpart, which causes them to overshoot more often. We suspect this is because they learn a more complex function for stopping than the simple counting method used by vision-less memory models and this function is less able to handle errors during navigation.

Question somewhat helps. Fig. 7 provides a comparison of M+PC and M+RGB and M+PC+RGB with and without the question (Q). Interestingly, we do not see large improvements when providing models with the question. Given how much **color_room** dominates our dataset, it seems reasonable to expect that telling the navigation models which room to go to would be a large benefit. We suspect that our models are not able to properly utilize this information due to limitations of behavior cloning. Models trained with behavior cloning never see mistakes or exploration and therefore never learn to correct mistakes or explore.

PC+RGB provides the best of both worlds. Fig. 6 also provides a comparison of the three different vision modalities. The general tend is that point clouds provided a richer signal for obstacle avoidance (corresponding to lower $\%_{\mathbf{collision}}$ values), while RGB provides richer semantic information (corresponding to a higher $\mathbf{IoU_T}$ and $\mathbf{QA}$). Combining both point clouds and RGB provides improvements to both obstacle avoidance and leveraging semantic information.

6. Conclusion

We present an extension of the task of EmbodiedQA to photorealistic environments utilizing the Matterport 3D dataset and propose the MP3D-EQA v1 dataset. We then present a thorough study of 2 navigation baselines and 2 different navigation architectures with 8 different input variations. We develop an end-to-end trainable navigation model capable of learning goal-driving navigation policies directly from 3D point clouds. We provide analysis and insight into the factors that affect navigation performance and propose a novel weighting scheme – *Inflection Weighting* – that increases the effectiveness of behavior cloning. We demonstrate that two the navigation baselines, random and forward-only, are quite strong under the evaluation settings presented by [1]. Our work serves as a step towards bridging the gap between ***internet vision***-style problems and the goal of ***vision for embodied perception***.

Acknowledgements. This work was supported in part by NSF (Grant # 1427300), AFRL, DARPA, Siemens, Samsung, Google, Amazon, ONR YIPs and ONR Grants N00014-16-1-{2713,2793}. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

References

[1] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018. 1, 2, 3, 4, 5, 6, 7, 8, 13, 14

[2] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017. 1, 2

[3] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environments. In *ICLR Workshop*, 2018. 1, 2

[4] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 1, 2

[5] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual Semantic Planning using Deep Successor Representations. In *ICCV*, 2017. 1, 2

[6] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*, 2017. 1, 2

[7] 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, 2

[8] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojtek Czarnecki, Max Jaderberg, Denis Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017. 1

[9] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. IQA: Visual question answering in interactive environments. In *CVPR*, 2018. 1, 2

[10] Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural Modular Control for Embodied Question Answering. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2018. 1

[11] 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. 1, 2

[12] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents, 2018. 1

[13] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*. 2017. 2

[14] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois Robert Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, Weber Liu, Thomas Funkhouser, and Alberto Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *ICRA*, 2018. 2

[15] Shiqi Zhang, Yuqian Jiang, Guni Sharon, and Peter Stone. Multirobot symbolic planning under temporal uncertainty. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Sytems (AAMAS)*, May 2017. 2

[16] 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. *International Conference on 3D Vision (3DV)*, 2017. 2, 3, 11, 12

[17] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(1):1334–1373, Jan. 2016. 2

[18] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. *RSS*, 2017. 2

[19] Abhinav Gupta Dhiraj Gandhi, Lerrel Pinto. Learning to fly by crashing. *IROS*, 2017. 2

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollr, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014. 2

[21] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *arXiv*. 2

[22] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *arXiv 1605.02097*, 2016. 2

[23] 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. 2, 3, 4

[24] 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. 2

[25] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 2

[26] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *CVPR*, 2017. 2

[27] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 2

[28] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d

shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 3

[29] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015. 3

[30] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016. 3

[31] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 3, 4

[32] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 3, 4, 5, 11, 12

[33] Saining Xie, Sainan Liu, Zeyu Chen, and Zhuowen Tu. Attentional shapecontextnet for point cloud recognition. In *CVPR*, 2018. 3

[34] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *CVPR*, 2018. 3

[35] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *ICCV*. IEEE, 2017. 3

[36] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 3

[37] Kenneth L Kelly. Twenty-two colors of maximum contrast. *Color Engineering*, 3(26):26–27, 1965. 3

[38] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017. 5

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5, 13

[40] Ankesh Anand, Eugene Belilovsky, Kyle Kastner, Hugo Larochelle, and Aaron Courville. Blindfold Baselines for Embodied QA. *arXiv preprint arXiv:1811.05013*, 2018. 5, 7, 13

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 5

[42] Jesse Thomason, Daniel Gordan, and Yonatan Bisk. Shifting the baseline: Single modality performance on visual navigation & qa. *arXiv preprint arXiv:1811.00613*, 2018. 6

[43] Yuxin Wu and Kaiming He. Group normalization. *arXiv preprint arXiv:1803.08494*, 2018. 6

[44] Alex Nash, Sven Koenig, and Craig Tovey. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Third Annual Symposium on Combinatorial Search*, 2010. 6

[45] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 6

[46] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 12, 13

7. Color Label Collection Interface

Fig. 8 shows the interface we used to collect dominate color annotations from workers on Amazon Mechanical Turk.

8. Point Cloud Rendering

In order to render 2.5D RGB-D frames, we first construct a global point cloud from all of the panoramas provided in an environment in the Matterport3D dataset [16]. Next, we determine what parts of the global point cloud are in the agent's current view. The agent's current position, A_{pos}, camera parameters (field of view, and aspect ratio), and the mesh reconstruction are used to determine which points are within its view pyramid (frustum).

For each point p_i in view, we first check if the point lies within the agents view frustum using the extrinsic and intrinsic camera matrices. After determining which points lie within the view frustum, we check for occlusions. We then draw a line between the agent's camera and p_i, let L_i be that line. We intersect this line with the provided mesh and keep the intersection that is *closest* to the agent, let $\operatorname{argmin}(L_i \cap \mathcal{M})$ be that intersection. If the distance to the *closest* intersection with the mesh, $||A_{pos} - \operatorname{argmin}(L_i \cap \mathcal{M})||$, is less than the distance to the point, $||A_{pos} - p_i||$, indicating that there is a closer (occluding) point, we remove the point. We also perform this check in the other direction and remove the point if $||A_{pos} - p_i|| < ||A_{pos} - \operatorname{argmin}(L_i \cap \mathcal{M})||$, indicating that the point is in the free-space. Points in the free-space are a result of panorama alignment errors and scanning oddities from reflections. In practice, we find that equality is too strict of a criteria due to mesh reconstruction errors, we instead remove a point if the absolute difference of the distances is greater than $\epsilon = 0.25$ cm, $\operatorname{abs}(||A_{pos} - \operatorname{argmin}(L_i \cap \mathcal{M})|| - ||A_{pos} - p_i||) > \epsilon$.

The implementation implied by the description above would be very slow. What we have described above implies something akin to a ray-tracing rendering. As in normal graphics pipelines, we can significantly speed this up by approximation with rasterization. We rasterize the provided mesh at A_{pos} and capture the depth buffer. The depth buffer provides the distance from the agent to the *closest* intersection with the mesh for some finite number of rays. Let R_j be a ray in that set and $||A_{pos} - \operatorname{argmin}(R_j \cap \mathcal{M})||$ be the distance to the closest intersection with the mesh as provided by the depth buffer. We can then approximate $||A_{pos} - \operatorname{argmin}(L_i \cap \mathcal{M})||$ for every point p_i in the point cloud by finding the R_j that is the closest to parallel to L_i and using the value of $||A_{pos} - \operatorname{argmin}(R_j \cap \mathcal{M})||$.

Further, a single global point cloud in Matterport3D environments has hundreds of millions of points and, in theory, we would need to check every single one to determine what points are visible. This would be quite slow and use a prohibitive amount of memory. We alleviate this by creating a significantly sparser point cloud and perform an initial visibility check on that instead. We then recheck the dense point cloud only in areas of the sparse point cloud that passed the initial visibility check.

9. Perception Models

Here we provide full details on the architectures of our perceptual encoders and the decoder heads used for their pre-training tasks.

9.1. PC – PointNet++

We use notation similar to the notation from Qi *et al.* [32] to specify our PointNet++ architecture. Let $P^{(0)}$ be the input point cloud. Then,

$$SA^{(k+1)}(P^{(k)}, N, [r^{(1)}, ..., r^{(m)}],\\ [[l_1^{(1)}, ..., l_d^{(1)}], ..., [l_1^{(m)}, ..., l_d^{(m)}]])$$

is a set abstraction module with multi-scale grouping that takes input the $k\ th$ level point cloud, $P^{(k)}$, and produces the $k+1\ th$ level point cloud, $P^{(k+1)}$, with N points. See section 4.1 in the main paper for a full description of a set abstraction module with single-scale grouping. A set abstraction module with multi-scale grouping is a logical extension. The feature descriptor for each point in $P^{(k+1)}$ is calculated across m different scales (m balls with different radii). $[l_1^{(1)}, ..., l_d^{(1)}]$ specifies the number of output channels for each layer in the shared-weighted multi-layer perceptron (MLP) used a each scale. The feature descriptor for any point is then the concatenation of the feature descriptor calculated at each scale.

As an analogy, a multi-scale convolution would be achieved by by first convolving the input with convolutions of different kernel sizes and then concatenating the outputs from each convolution.

$SA^{(k+1)}(P^{(k)}, [l_1, ..., l_d])$ is a global set abstraction module and produces a l_d dimensional feature vector.

Our encoder is specified by

$$\begin{aligned}
&SA^{(1)}(P^{(0)}, 1024, [0.05, 0.1],\\
&\quad [[32, 32, 64], [32, 64, 128]])\\
&SA^{(2)}(P^{(1)}, 256, [0.1, 0.2, 0.4],\\
&\quad [[64, 128, 128], [128, 128, 256],\\
&\quad [128, 128, 256, 256]])\\
&SA^{(3)}(P^{(2)}, 64, [0.4, 0.8],\\
&\quad [[128, 128, 128, 256, 256],\\
&\quad [128, 128, 256, 256, 256, 512]])\\
&SA^{(4)}(P^{(3)}, [256, 512, 1024])
\end{aligned}$$

$FP(P^{(k+1}, P^{(k)}, use\_skip, [l_1, ...l_d])$ is a feature propagation layer that transfers the features from $P^{(k+1)}$ to $P^{(k)}$

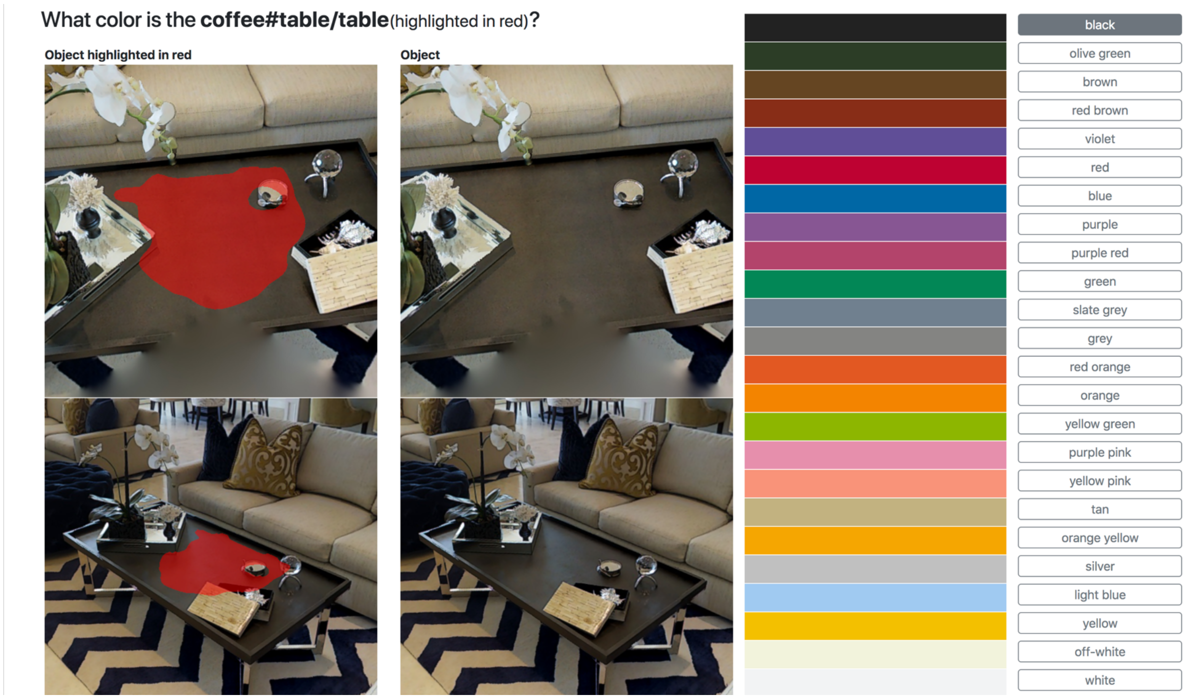

What color is the **coffee#table/table**(highlighted in red)?

Figure 8: Interface shown to AMT workers for collecting the dominate color name for objects in the Matterport3D dataset [16]. Workers were shown up to two good views of the object (some objects only had one good view) and the corresponding instance segmentation mask and asked to selected the dominate color of the object from a predefined list of colors.

[32]. For each point in $P^{(k)}$, its three nearest neighbors in $P^{(k+1)}$ are found and their feature descriptors are combined via inverse-distance weighted interpolation. The interpolated feature descriptor is then processed by shared-weight MLP with output channels specified by $[l_1, ...l_d]$. use_skip is a $True$ or $False$ boolean that determines if any existing features of $P^{(k)}$ are used. If use_skip is true, the the interpolated feature and the existing feature are concatenated together before the shared-weight MLP. If use_skip is false, only the interpolated feature is processed by the shared-weight MLP.

The semantic segmentation head is specified as

$$\begin{aligned} Prop &= FP(P^{(4)}, P^{(3)}, True, [256, 512]) \\ Prop &= FP(Prop, P^{(2)}, True, [256, 256]) \\ Prop &= FP(Prop, P^{(1)}, True, [256, 256]) \\ Output &= FP(Prop, P^{(0)}, True, [256, 128, 128, K]) \end{aligned}$$

where K is the number of class, 40 in our case.

The color prediction head is specified as

$$\begin{aligned} Prop &= FP(P^{(4)}, P^{(3)}, True, [256, 512]) \\ Prop &= FP(Prop, P^{(2)}, True, [256, 256]) \\ Prop &= FP(Prop, P^{(1)}, False, [256, 256]) \\ Output &= FP(Prop, P^{(0)}, False, [256, 128, 128, 3]) \end{aligned}$$

The structure prediction head is specified as

$$\begin{aligned} &(a)\ FC(P^{(4)}, 256) \\ &(b)\ FC((a), 256) \\ &(c)\ FC((b), N * 3) \end{aligned}$$

$FC(in, d_{out})$ is a fully connected layer that takes in and produces a vector of size d_{out}. The output of the structure decoding head, (c), is then reshaped into a $(N, 3)$ point cloud.

Training details. We train with a batch size of 32 and utilize Adam [46]. We use an initial learning rate of 10^{-3} and decay the learning rate by 70% every 6.2×10^{-3} batches. We select the checkpoint by performance on a held-out validation set.

9.2. RGB – ResNet50

We use the initial convolution and 4 residual blocks from ResNet50 [39] as the encoder for RGB images. Each of the three decoders (semantic, depth, and color) are identical and consistent of 1x1 convolutions and bi-linear interpolation.

Let $RB2$, $RB3$, and $RB4$ be the outputs from the second, third, and fourth residual blocks in ResNet50 respectively. Each decoder is then parameterized as

$$\begin{aligned} Up4 &= 1x1Conv(RB4, C) \\ Up3 &= UpSample(Up4) + 1x1Conv(RB3, C) \\ Up2 &= UpSample(Up3) + 1x1Conv(RB2, C) \\ Output &= UpSample(Up2) \end{aligned}$$

$1x1Conv$ is simply a 1-by-1 convectional layer that transforms its input to have C channels. $UpSample$ is a bi-linear interpolation layer that up-samples its input to be the necessary size. Where C is the number of output channels for the given decoder type, K for semantic, 1 for depth, and 3 for color.

Training details. Due to the high prevalence of walls and floors among indoor scenes, we use class weighted cross-entropy loss for semantic segmentation. For depth and autoencoding, we use smooth-ℓ_1 The total loss is then the weighted sum of the individual task losses:

$$L = \lambda_{Seg.} L_{Seg.} + \lambda_{Depth} L_{Depth} + \lambda_{AE} L_{AE}$$

We find $\lambda_{Seg.} = 0.1$, $\lambda_{Depth} = 10$, and $\lambda_{AE} = 10$ balances the magnitudes of the various losses and works well.

Tab. 2 provides a comparison of pre-training tasks when trained for separately vs. jointly and comparison with the Shallow CNN used in Das *et al.* [1]. The experiments showed that training jointly doesn't sacrifice performance on depth and autoencoding.

We train with a batch size of 20 and utilize Adam [46]. We use an initial learning rate of 10^{-5}, maximum epochs considered 300, we sampled every 3-rd frame from shortest paths. The checkpoint was selected by performance on a held-out validation set.

10. Question answering model training

- `lstm-question-only`, we train with a batch size of 40 and utilize Adam with a learning rate of 10^{-3};
- `nn-question-only` and `bow-question-only` we use the code provided in [40];
- `attention+*`, we train with a batch size of 20 and utilize Adam with a learning rate of 10^{-3};
- `spatial+RGB+Q` , we train with a batch size of 32 and utilize Adam with a learning rate of 10^{-3}

For all models, the best checkpoint is selected via performance on a held out validation set.

11. Navigation model training

For models trained with and without inflection weighted loss, we train and select checkpoints with the same procedure.

- `R+*`, we train with a batch size of 20 and utilize Adam with a learning rate of 10^{-3};
- `M+*`, we train with a batch size of 5 full sequences and utilize Adam with a learning rate of 2×10^{-4}. Note that due to the length of sequences, we compute the forward and backward pass for each sequence individually and average the gradients

Due to the difference in end-to-end evaluation and teacher-forcing accuracy, we use the following method to select checkpoints for navigation models: First we run non-minimal suppression with a window size of 5 on teacher forcing validation loss (vanilla cross-entropy for `NoIW-*` models, and inflection weighted cross-entropy otherwise). After NMS, we select the top 5 checkpoints by validation loss and run them through end-to-end evaluation on the validation set. The best checkpoint is the model that has the highest value of $\mathbf{QA} + \mathbf{d_\Delta}$ at T_{-50}. We find that inflection weighted cross-entropy is a significantly better predictor of end-to-end performance than vanilla cross-entropy. Interestingly, we also find that teacher forcing validation accuracy is a good predictor of end-to-end performance when training with inflection weighting and don't see a significant difference in performance if the procedure above is run with teacher forcing validation error instead of loss.

12. Results

We provide the full tables we first analyzed in their entirety and then sliced for the analysis provided in the main paper. Tab. 3 shows our primary results with inflection weighting and Tab. 4 shows the same set of models trained without inflection weighting.

We also provide the full tables with confidence intervals. Confidence intervals are 90% confidence intervals calculated with empirical bootstrapping. See Tab. 3 and Tab. 6.

As a reminder, we use the following notation to specify our models: For the base architecture, `R` denotes reactive models and `M` denotes memory models. The base architectures are then augmented with their input types, `+PC`, `+RGB`, and `+Q`. So a memory model that utilizes point clouds (but no question) is denoted as `M+PC`. Unless otherwise specified (by the prefix `NoIW`), models are trained with inflection weighting. We denote the two baseline navigators, forward-only and random, as `Fwd` and `Random`, respectively.

12.1. Navigation Performance Correlation Analysis

Here we provide some additional insight on the effect the question has on our navigation models. We find that while the question does significantly impact performance on average, it does significantly change a model's behavior.

Model	Total loss	AE loss	Depth loss	Sem. loss	PA	MPA	MIOU
Shallow CNN [1] – All Tasks	0.369	0.0047	0.0077	2.45	0.384	0.14	0.070
ResNet50 – AE only	1.774	**0.0030**	0.0873	8.71	0.008	0.02	0.002
ResNet50 – Depth only	2.311	0.1051	0.0072	11.89	0.005	0.02	0.002
ResNet50 – AE and depth only	1.003	0.0034	**0.0067**	9.02	0.006	0.03	0.002
ResNet50 – All Tasks	**0.356**	0.0040	0.0069	**2.48**	**0.390**	**0.15**	**0.078**

Table 2: Performance on the MP3D-EQA v1 validation set for Shallow CNN [1] and for ResNet50 when trained for autoencdoing only, for depth only, for autoencoding and depth, and for all tasks jointly. For segmentation we report the overall pixel accuracy (PA), mean pixel accuracy (MPA) averaged over all semantic classes and the mean IOU intersection over union (MIOU). For depth and autoencoder, we report the smooth-ℓ_1 on the validation set.

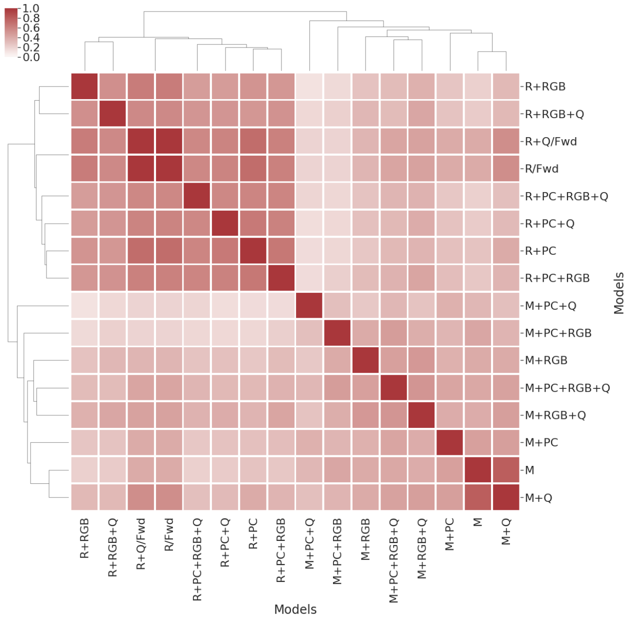


Figure 9: Correlation and clustering of correlation for d_T per episode at T_{-30}. Reactive models build a strong cluster, while memory models are less correlated. Usage of the question changes navigators with vision significantly.

Fig. 9 shows the Pearson correlation correlation coefficient between $\mathbf{d_T}$ at T_{-30} for all models and cluster discovery using the Nearest Point Algorithm. All navigators that use the reactive base architecture build a strong cluster and are highly correlated with each-other. The vision-less memory models are also well correlated with each-other, indicating that the question has little relation to the distribution of actions along a shortest path. Memory models that use vision don't tend to be well correlated with their counterpart that also uses the question in-spite of the question having little effect on overall navigation metrics.

Navigator	**Navigation**																			**QA**		
	$\mathbf{d_0}$ (For reference)			$\mathbf{d_T}$ (Lower is better)			$\mathbf{d_{min}}$ (Lower is better)			$\mathbf{d_\Delta}$ (Higher is better)			$\%_{\mathbf{collision}}$ (Lower is better)			$\mathbf{IoU_T}$ (Higher is better)			$\mathbf{Top-1}$ (Higher is better)		
	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}
`R`	0.354	1.898	3.547	0.933	1.330	2.154	**0.011**	0.346	1.397	−0.579	0.568	1.393	79.554	66.182	62.563	0.062	0.050	0.030	0.390	0.379	0.354
`R+Q`	0.354	1.898	3.547	0.933	1.330	2.154	**0.011**	0.346	1.397	−0.579	0.568	1.393	79.554	66.182	62.563	0.062	0.050	0.030	0.390	0.379	0.354
`R+RGB`	0.354	1.898	3.547	1.194	1.617	2.340	0.040	0.375	1.349	−0.840	0.281	1.207	59.959	51.460	48.425	0.077	0.058	0.031	0.395	0.396	0.372
`R+RGB+Q`	0.354	1.898	3.547	1.407	1.740	2.521	0.034	0.340	1.332	−1.053	0.157	1.026	51.128	44.160	42.692	0.111	0.070	0.054	0.383	0.388	0.375
`R+PC`	0.354	1.898	3.547	1.428	1.754	2.352	0.021	**0.320**	1.164	−1.074	0.144	1.195	50.148	41.612	42.203	0.070	0.067	0.047	0.356	0.394	0.375
`R+PC+Q`	0.354	1.898	3.547	1.514	1.812	2.394	0.033	0.325	**1.160**	−1.160	0.085	1.153	46.910	36.303	39.012	0.059	0.052	0.043	0.364	0.364	0.363
`R+PC+RGB`	0.354	1.898	3.547	1.547	1.791	2.336	0.020	0.322	1.211	−1.193	0.107	1.211	44.941	34.859	37.138	0.084	0.077	0.044	0.374	0.390	0.366
`R+PC+RGB+Q`	0.354	1.898	3.547	1.539	1.843	2.420	0.032	0.323	1.170	−1.185	0.055	1.127	42.018	34.318	37.069	0.067	0.072	0.055	0.370	0.395	0.369
`M`	0.354	1.898	3.547	**0.366**	**0.830**	1.833	0.090	0.505	1.460	**−0.012**	**1.068**	1.714	6.903	10.989	23.250	0.128	0.091	0.081	0.365	0.375	0.363
`M+Q`	0.354	1.898	3.547	0.508	0.933	1.920	0.052	0.426	1.421	−0.154	0.965	1.627	16.268	19.808	32.856	0.147	0.109	0.068	0.391	0.395	0.376
`M+RGB`	0.354	1.898	3.547	0.637	1.157	2.177	0.099	0.538	1.479	−0.283	0.741	1.370	12.582	15.130	26.179	0.188	0.136	0.075	0.397	0.403	0.384
`M+RGB+Q`	0.354	1.898	3.547	0.707	1.171	2.194	0.071	0.423	1.386	−0.353	0.727	1.353	14.212	15.908	25.578	0.189	0.141	0.083	**0.407**	0.394	0.384
`M+PC`	0.354	1.898	3.547	0.494	1.020	1.817	0.098	0.484	1.236	−0.140	0.878	1.730	6.647	9.169	18.319	0.163	0.114	0.083	0.396	**0.411**	**0.390**
`M+PC+Q`	0.354	1.898	3.547	0.502	1.030	1.910	0.081	0.497	1.272	−0.148	0.868	1.637	5.584	**8.833**	**15.783**	0.184	0.158	**0.118**	0.382	0.387	0.374
`M+PC+RGB`	0.354	1.898	3.547	0.461	0.940	**1.791**	0.103	0.513	1.269	−0.107	0.958	**1.756**	**4.957**	9.574	18.890	**0.209**	**0.179**	0.111	0.381	0.393	0.363
`M+PC+RGB+Q`	0.354	1.898	3.547	0.574	1.044	1.898	0.083	0.431	1.203	−0.220	0.854	1.649	8.328	10.674	19.797	**0.209**	0.148	0.112	0.389	0.390	0.373
`Random`	0.354	1.898	3.547	0.912	1.273	2.654	0.048	0.796	2.263	−0.558	0.625	0.893	13.775	10.708	10.677	0.098	0.072	0.041	0.365	0.368	0.364
`ShortestPath`	0.354	1.898	3.547	0.005	0.005	0.005	0.005	0.005	0.005	0.349	1.893	3.542	0.000	0.000	0.000	0.581	0.581	0.581	0.451	0.451	0.451

Table 3: Evaluation of EmbodiedQA agents trained with inflection weighting on navigation and answering metrics for the MP3D-EQA v1 test set. RGB models perceive the world via RGB images and use ResNet50. PC models perceive the world via point clouds and use PointNet++. PC+RGB models use both perception modalities and their respective networks.

Navigator	Navigation																		QA		
	$\mathbf{d_0}$ (For reference)			$\mathbf{d_T}$ (Lower is better)			$\mathbf{d_{min}}$ (Lower is better)			$\mathbf{d_\Delta}$ (Higher is better)			$\%_{\mathbf{collision}}$ (Lower is better)			$\mathbf{IoU_T}$ (Higher is better)			$\mathbf{Top-1}$ (Higher is better)		
	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}
NoIW-R	0.354	1.898	3.547	0.933	1.330	**2.154**	**0.011**	0.346	1.397	−0.579	0.568	**1.393**	79.554	66.182	62.563	0.062	0.050	0.030	0.390	0.379	0.354
NoIW-R+Q	0.354	1.898	3.547	0.933	1.330	**2.154**	**0.011**	0.346	1.397	−0.579	0.568	**1.393**	79.554	66.182	62.563	0.062	0.050	0.030	0.390	0.379	0.354
NoIW-R+RGB	0.354	1.898	3.547	1.419	1.713	2.528	0.041	0.404	1.417	−1.065	0.185	1.019	56.718	50.376	45.866	0.086	0.049	0.035	0.382	0.386	0.360
NoIW-R+RGB+Q	0.354	1.898	3.547	1.405	1.829	2.658	0.051	0.455	1.463	−1.051	0.069	0.889	43.226	38.538	36.172	0.075	0.060	0.054	0.384	0.390	0.373
NoIW-R+PC	0.354	1.898	3.547	1.385	1.662	2.483	0.026	**0.343**	1.294	−1.031	0.236	1.064	43.067	34.100	37.078	0.074	0.061	0.043	0.375	0.398	0.376
NoIW-R+PC+Q	0.354	1.898	3.547	1.515	1.858	2.646	0.038	0.394	1.333	−1.161	0.040	0.901	37.669	31.714	**33.563**	0.078	0.059	0.054	0.372	**0.399**	0.378
NoIW-R+PC+RGB	0.354	1.898	3.547	1.462	1.759	2.523	0.025	0.347	**1.285**	−1.108	0.139	1.024	53.785	41.955	40.293	0.067	0.052	0.042	0.384	0.389	0.369
NoIW-R+PC+RGB+Q	0.354	1.898	3.547	1.297	1.704	2.543	0.030	0.425	1.417	−0.943	0.194	1.004	47.506	39.554	36.242	0.069	0.062	0.046	0.368	0.375	0.353
NoIW-M	0.354	1.898	3.547	0.933	1.330	2.186	**0.011**	0.346	1.430	−0.579	0.568	1.361	79.554	66.182	62.818	0.062	0.050	0.029	0.390	0.379	0.356
NoIW-M+Q	0.354	1.898	3.547	0.933	1.330	2.202	**0.011**	0.346	1.445	−0.579	0.568	1.345	79.554	66.182	62.931	0.062	0.050	0.029	0.390	0.379	0.354
NoIW-M+RGB	0.354	1.898	3.547	0.902	1.396	2.512	0.024	0.400	1.608	−0.548	0.502	1.035	67.347	59.661	59.378	0.080	0.061	0.033	**0.406**	0.384	0.353
NoIW-M+RGB+Q	0.354	1.898	3.547	0.911	1.394	2.573	0.024	0.410	1.644	−0.557	0.504	0.974	66.198	59.317	58.941	0.094	0.064	0.030	0.390	0.380	0.356
NoIW-M+PC	0.354	1.898	3.547	0.811	1.245	2.244	0.034	0.370	1.414	−0.457	0.652	1.303	37.865	30.964	38.521	0.113	0.114	**0.091**	0.386	**0.399**	0.376
NoIW-M+PC+Q	0.354	1.898	3.547	**0.790**	**1.233**	2.213	0.038	0.379	1.416	**−0.436**	**0.665**	1.334	36.215	31.431	38.911	0.115	0.099	0.076	0.393	**0.399**	**0.379**
NoIW-M+PC+RGB	0.354	1.898	3.547	0.929	1.405	2.435	0.016	0.375	1.526	−0.575	0.492	1.112	61.658	51.595	52.093	0.100	0.075	0.041	0.388	0.385	0.353
NoIW-M+PC+RGB+Q	0.354	1.898	3.547	0.871	1.322	2.256	0.046	0.381	1.377	−0.517	0.576	1.291	**32.496**	**27.391**	34.923	**0.145**	**0.121**	0.088	0.383	0.398	0.374
NoIW-Random	0.354	1.898	3.547	0.912	1.273	2.654	0.048	0.796	2.263	−0.558	0.625	0.893	13.775	10.708	10.677	0.098	0.072	0.041	0.365	0.368	0.364
NoIW-ShortestPath	0.354	1.898	3.547	0.005	0.005	0.005	0.005	0.005	0.005	0.349	1.893	3.542	0.000	0.000	0.000	0.581	0.581	0.581	0.451	0.451	0.451

Table 4: Evaluation of EmbodiedQA agents trained **without** inflection weighting on navigation and answering metrics for the MP3D-EQA v1 test set.

Navigator	Navigation																		QA		
	$\mathbf{d_0}$ (For reference)			$\mathbf{d_T}$ (Lower is better)			$\mathbf{d_{min}}$ (Lower is better)			$\mathbf{d_\Delta}$ (Higher is better)			$\%_{\mathbf{collision}}$ (Lower is better)			$\mathbf{IoU_T}$ (Higher is better)			$\mathbf{Top-1}$ (Higher is better)		
	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}
NoIW-R	0.354	1.898	3.547	0.933 (0.896, 0.969)	1.330 (1.290, 1.370)	2.154 (2.109, 2.196)	**0.011** (0.010, 0.013)	0.346 (0.333, 0.359)	1.397 (1.367, 1.426)	−0.579 (−0.615, − 0.542)	0.568 (0.524, 0.611)	1.393 (1.347, 1.443)	79.554 (79.043, 80.065)	66.182 (65.604, 66.768)	62.563 (61.911, 63.244)	0.062 (0.056, 0.069)	0.050 (0.043, 0.056)	0.030 (0.026, 0.035)	0.390 (0.376, 0.404)	0.379 (0.364, 0.392)	0.354 (0.340, 0.367)
NoIW-R+Q	0.354	1.898	3.547	0.933 (0.896, 0.969)	1.330 (1.290, 1.370)	2.154 (2.109, 2.196)	**0.011** (0.010, 0.013)	0.346 (0.333, 0.359)	1.397 (1.367, 1.426)	−0.579 (−0.615, − 0.542)	0.568 (0.524, 0.611)	1.393 (1.347, 1.443)	79.554 (79.043, 80.065)	66.182 (65.604, 66.768)	62.563 (61.911, 63.244)	0.062 (0.056, 0.069)	0.050 (0.043, 0.056)	0.030 (0.026, 0.035)	0.390 (0.376, 0.404)	0.379 (0.364, 0.392)	0.354 (0.340, 0.367)
NoIW-R+RGB	0.354	1.898	3.547	1.194 (1.152, 1.234)	1.617 (1.568, 1.665)	2.340 (2.294, 2.385)	0.040 (0.037, 0.043)	0.375 (0.360, 0.390)	1.349 (1.319, 1.379)	−0.840 (−0.881, − 0.797)	0.281 (0.230, 0.333)	1.207 (1.156, 1.258)	59.959 (58.970, 60.958)	51.460 (50.585, 52.356)	48.425 (47.578, 49.301)	0.077 (0.069, 0.084)	0.058 (0.051, 0.064)	0.031 (0.026, 0.035)	0.395 (0.380, 0.409)	0.396 (0.381, 0.409)	0.372 (0.358, 0.386)
NoIW-R+RGB+Q	0.354	1.898	3.547	1.407 (1.361, 1.453)	1.740 (1.692, 1.789)	2.521 (2.467, 2.573)	0.034 (0.031, 0.037)	0.340 (0.327, 0.354)	1.332 (1.301, 1.363)	−1.053 (−1.101, − 1.005)	0.157 (0.106, 0.209)	1.026 (0.969, 1.085)	51.128 (50.100, 52.154)	44.160 (43.263, 45.066)	42.692 (41.744, 43.618)	0.111 (0.101, 0.121)	0.070 (0.061, 0.078)	0.054 (0.047, 0.061)	0.383 (0.369, 0.397)	0.388 (0.374, 0.403)	0.375 (0.361, 0.389)
NoIW-R+PC	0.354	1.898	3.547	1.428 (1.379, 1.478)	1.754 (1.706, 1.800)	2.352 (2.299, 2.402)	0.021 (0.019, 0.022)	**0.320** (0.308, 0.333)	1.164 (1.135, 1.192)	−1.074 (−1.123, − 1.023)	0.144 (0.092, 0.193)	1.195 (1.140, 1.253)	50.148 (49.084, 51.218)	41.612 (40.672, 42.570)	42.203 (41.276, 43.117)	0.070 (0.063, 0.077)	0.067 (0.060, 0.074)	0.047 (0.042, 0.053)	0.356 (0.342, 0.369)	0.394 (0.380, 0.408)	0.375 (0.361, 0.389)
NoIW-R+PC+Q	0.354	1.898	3.547	1.514 (1.466, 1.563)	1.812 (1.763, 1.862)	2.394 (2.344, 2.445)	0.033 (0.030, 0.035)	0.325 (0.312, 0.338)	**1.160** (1.131, 1.189)	−1.160 (−1.209, − 1.110)	0.085 (0.034, 0.138)	1.153 (1.096, 1.210)	46.910 (45.833, 47.956)	36.303 (35.341, 37.244)	39.012 (38.074, 39.942)	0.059 (0.052, 0.065)	0.052 (0.045, 0.058)	0.043 (0.038, 0.048)	0.364 (0.350, 0.378)	0.364 (0.350, 0.378)	0.363 (0.349, 0.377)
NoIW-R+PC+RGB	0.354	1.898	3.547	1.547 (1.497, 1.594)	1.791 (1.743, 1.838)	2.336 (2.287, 2.385)	0.020 (0.018, 0.022)	0.322 (0.309, 0.335)	1.211 (1.182, 1.241)	−1.193 (−1.241, − 1.143)	0.107 (0.055, 0.159)	1.211 (1.158, 1.265)	44.941 (43.856, 46.040)	34.859 (33.871, 35.853)	37.138 (36.190, 38.091)	0.084 (0.076, 0.093)	0.077 (0.068, 0.086)	0.044 (0.039, 0.049)	0.374 (0.360, 0.388)	0.390 (0.376, 0.404)	0.366 (0.352, 0.380)
NoIW-R+PC+RGB+Q	0.354	1.898	3.547	1.539 (1.490, 1.587)	1.843 (1.795, 1.890)	2.420 (2.369, 2.470)	0.032 (0.029, 0.035)	0.323 (0.311, 0.337)	1.170 (1.141, 1.200)	−1.185 (−1.233, − 1.137)	0.055 (0.004, 0.107)	1.127 (1.072, 1.186)	42.018 (40.933, 43.100)	34.318 (33.370, 35.271)	37.069 (36.127, 38.004)	0.067 (0.060, 0.075)	0.072 (0.064, 0.080)	0.055 (0.048, 0.062)	0.370 (0.356, 0.384)	0.395 (0.381, 0.409)	0.369 (0.355, 0.382)
NoIW-M	0.354	1.898	3.547	**0.366** (0.354, 0.379)	**0.830** (0.811, 0.849)	1.833 (1.800, 1.867)	0.090 (0.085, 0.095)	0.505 (0.490, 0.521)	1.460 (1.432, 1.489)	**−0.012** (−0.026, 0.001)	**1.068** (1.042, 1.092)	1.714 (1.674, 1.754)	6.903 (6.508, 7.289)	10.989 (10.516, 11.464)	23.250 (22.632, 23.882)	0.128 (0.117, 0.138)	0.091 (0.083, 0.099)	0.081 (0.072, 0.089)	0.365 (0.351, 0.379)	0.375 (0.361, 0.389)	0.363 (0.349, 0.376)
NoIW-M+Q	0.354	1.898	3.547	0.508 (0.491, 0.524)	0.933 (0.912, 0.955)	1.920 (1.884, 1.955)	0.052 (0.048, 0.056)	0.426 (0.412, 0.441)	1.421 (1.392, 1.450)	−0.154 (−0.171, − 0.136)	0.965 (0.937, 0.992)	1.627 (1.586, 1.669)	16.268 (15.645, 16.890)	19.808 (19.165, 20.458)	32.856 (32.135, 33.588)	0.147 (0.135, 0.157)	0.109 (0.099, 0.119)	0.068 (0.061, 0.075)	0.391 (0.376, 0.405)	0.395 (0.380, 0.409)	0.376 (0.363, 0.390)
NoIW-M+RGB	0.354	1.898	3.547	0.637 (0.611, 0.663)	1.157 (1.125, 1.187)	2.177 (2.129, 2.224)	0.099 (0.093, 0.105)	0.538 (0.521, 0.556)	1.479 (1.447, 1.510)	−0.283 (−0.310, − 0.257)	0.741 (0.707, 0.775)	1.370 (1.319, 1.422)	12.582 (11.849, 13.341)	15.130 (14.404, 15.853)	26.179 (25.319, 27.038)	0.188 (0.174, 0.201)	0.136 (0.125, 0.148)	0.075 (0.067, 0.082)	0.397 (0.384, 0.411)	0.403 (0.388, 0.417)	0.384 (0.370, 0.397)
NoIW-M+RGB+Q	0.354	1.898	3.547	0.707 (0.681, 0.733)	1.171 (1.140, 1.202)	2.194 (2.146, 2.240)	0.071 (0.066, 0.076)	0.423 (0.408, 0.437)	1.386 (1.355, 1.416)	−0.353 (−0.380, − 0.327)	0.727 (0.692, 0.762)	1.353 (1.301, 1.406)	14.212 (13.490, 14.925)	15.908 (15.219, 16.604)	25.578 (24.771, 26.392)	0.189 (0.175, 0.203)	0.141 (0.129, 0.153)	0.083 (0.075, 0.091)	**0.407** (0.392, 0.421)	0.394 (0.380, 0.408)	0.384 (0.370, 0.398)
NoIW-M+PC	0.354	1.898	3.547	0.494 (0.475, 0.512)	1.020 (0.993, 1.047)	1.817 (1.774, 1.859)	0.098 (0.092, 0.103)	0.484 (0.468, 0.500)	1.236 (1.207, 1.265)	−0.140 (−0.159, − 0.120)	0.878 (0.846, 0.910)	1.730 (1.683, 1.778)	6.647 (6.073, 7.217)	9.169 (8.522, 9.804)	18.319 (17.510, 19.148)	0.163 (0.152, 0.174)	0.114 (0.104, 0.123)	0.083 (0.076, 0.090)	0.396 (0.381, 0.410)	**0.411** (0.396, 0.425)	**0.390** (0.376, 0.404)
NoIW-M+PC+Q	0.354	1.898	3.547	0.502 (0.483, 0.521)	1.030 (1.002, 1.059)	1.910 (1.864, 1.954)	0.081 (0.076, 0.086)	0.497 (0.481, 0.514)	1.272 (1.242, 1.302)	−0.148 (−0.168, − 0.128)	0.868 (0.836, 0.900)	1.637 (1.588, 1.688)	5.584 (5.105, 6.066)	**8.833** (8.203, 9.454)	**15.783** (15.007, 16.548)	0.184 (0.171, 0.196)	0.158 (0.146, 0.170)	**0.118** (0.107, 0.128)	0.382 (0.368, 0.396)	0.387 (0.372, 0.401)	0.374 (0.360, 0.388)
NoIW-M+PC+RGB	0.354	1.898	3.547	0.461 (0.443, 0.478)	0.940 (0.915, 0.964)	**1.791** (1.751, 1.831)	0.103 (0.097, 0.109)	0.513 (0.497, 0.529)	1.269 (1.241, 1.297)	−0.107 (−0.125, − 0.088)	0.958 (0.928, 0.987)	**1.756** (1.709, 1.802)	**4.957** (4.519, 5.391)	9.574 (8.973, 10.184)	18.890 (18.105, 19.660)	**0.209** (0.195, 0.222)	**0.179** (0.166, 0.191)	0.111 (0.102, 0.121)	0.381 (0.367, 0.395)	0.393 (0.378, 0.407)	0.363 (0.350, 0.377)
NoIW-M+PC+RGB+Q	0.354	1.898	3.547	0.574 (0.551, 0.598)	1.044 (1.014, 1.074)	1.898 (1.853, 1.941)	0.083 (0.078, 0.088)	0.431 (0.416, 0.446)	1.203 (1.173, 1.232)	−0.220 (−0.244, − 0.196)	0.854 (0.819, 0.887)	1.649 (1.600, 1.700)	8.328 (7.687, 8.962)	10.674 (10.000, 11.342)	19.797 (18.973, 20.624)	**0.209** (0.195, 0.222)	0.148 (0.137, 0.160)	0.112 (0.102, 0.123)	0.389 (0.375, 0.404)	0.390 (0.376, 0.404)	0.373 (0.359, 0.387)
NoIW-Random	0.354	1.898	3.547	0.912 (0.890, 0.935)	1.273 (1.248, 1.298)	2.654 (2.619, 2.686)	0.048 (0.045, 0.051)	0.796 (0.778, 0.815)	2.263 (2.237, 2.289)	−0.558 (−0.583, − 0.533)	0.625 (0.597, 0.651)	0.893 (0.861, 0.926)	13.775 (13.476, 14.069)	10.708 (10.431, 10.985)	10.677 (10.408, 10.946)	0.098 (0.088, 0.107)	0.072 (0.064, 0.080)	0.041 (0.035, 0.046)	0.365 (0.352, 0.380)	0.368 (0.354, 0.382)	0.364 (0.350, 0.378)
NoIW-ShortestPath	0.354	1.898	3.547	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.349 (0.340, 0.358)	1.893 (1.876, 1.909)	3.542 (3.522, 3.563)	0.000 (0.000, 0.000)	0.000 (0.000, 0.000)	0.000 (0.000, 0.000)	0.581 (0.567, 0.595)	0.581 (0.567, 0.595)	0.581 (0.567, 0.595)	0.451 (0.437, 0.466)	0.451 (0.437, 0.466)	0.451 (0.437, 0.465)

Table 5: Tab. 3 (navigation results with inflection weightings) with 90% bootstrap confidence intervals

Navigator	Navigation																		QA		
	$\mathbf{d_0}$ (For reference)			$\mathbf{d_T}$ (Lower is better)			$\mathbf{d_{min}}$ (Lower is better)			$\mathbf{d_\Delta}$ (Higher is better)			$\%_{\mathbf{collision}}$ (Lower is better)			$\mathbf{IoU_T}$ (Higher is better)			$\mathbf{Top-1}$ (Higher is better)		
	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}	T_{-10}	T_{-30}	T_{-50}
NoIW-R	0.354	1.898	3.547	0.933 (0.896, 0.969)	1.330 (1.290, 1.370)	**2.154** (2.109, 2.196)	**0.011** (0.010, 0.013)	0.346 (0.333, 0.359)	1.397 (1.367, 1.426)	−0.579 (−0.615, −0.542)	0.568 (0.524, 0.611)	**1.393** (1.347, 1.443)	79.554 (79.043, 80.065)	66.182 (65.604, 66.768)	62.563 (61.911, 63.244)	0.062 (0.056, 0.069)	0.050 (0.043, 0.056)	0.030 (0.026, 0.035)	0.390 (0.376, 0.404)	0.379 (0.364, 0.392)	0.354 (0.340, 0.367)
NoIW-R+Q	0.354	1.898	3.547	0.933 (0.896, 0.969)	1.330 (1.290, 1.370)	**2.154** (2.109, 2.196)	**0.011** (0.010, 0.013)	0.346 (0.333, 0.359)	1.397 (1.367, 1.426)	−0.579 (−0.615, −0.542)	0.568 (0.524, 0.611)	**1.393** (1.347, 1.443)	79.554 (79.043, 80.065)	66.182 (65.604, 66.768)	62.563 (61.911, 63.244)	0.062 (0.056, 0.069)	0.050 (0.043, 0.056)	0.030 (0.026, 0.035)	0.390 (0.376, 0.404)	0.379 (0.364, 0.392)	0.354 (0.340, 0.367)
NoIW-R+RGB	0.354	1.898	3.547	1.419 (1.372, 1.468)	1.713 (1.666, 1.759)	2.528 (2.475, 2.580)	0.041 (0.037, 0.044)	0.404 (0.389, 0.419)	1.417 (1.385, 1.449)	−1.065 (−1.114, −1.016)	0.185 (0.135, 0.235)	1.019 (0.963, 1.077)	56.718 (55.722, 57.732)	50.376 (49.506, 51.248)	45.866 (44.964, 46.762)	0.086 (0.077, 0.094)	0.049 (0.043, 0.055)	0.035 (0.030, 0.040)	0.382 (0.368, 0.396)	0.386 (0.372, 0.400)	0.360 (0.346, 0.373)
NoIW-R+RGB+Q	0.354	1.898	3.547	1.405 (1.353, 1.454)	1.829 (1.775, 1.882)	2.658 (2.599, 2.716)	0.051 (0.047, 0.055)	0.455 (0.438, 0.472)	1.463 (1.430, 1.496)	−1.051 (−1.101, −0.999)	0.069 (0.012, 0.125)	0.889 (0.827, 0.953)	43.226 (42.106, 44.334)	38.538 (37.559, 39.510)	36.172 (35.216, 37.109)	0.075 (0.066, 0.083)	0.060 (0.053, 0.067)	0.054 (0.047, 0.061)	0.384 (0.369, 0.398)	0.390 (0.376, 0.404)	0.373 (0.359, 0.387)
NoIW-R+PC	0.354	1.898	3.547	1.385 (1.338, 1.430)	1.662 (1.616, 1.708)	2.483 (2.428, 2.535)	0.026 (0.023, 0.028)	**0.343** (0.329, 0.358)	1.294 (1.263, 1.324)	−1.031 (−1.078, −0.983)	0.236 (0.185, 0.286)	1.064 (1.005, 1.123)	43.067 (41.951, 44.169)	34.100 (33.167, 35.046)	37.078 (36.097, 38.050)	0.074 (0.066, 0.082)	0.061 (0.054, 0.068)	0.043 (0.038, 0.048)	0.375 (0.361, 0.389)	0.398 (0.384, 0.412)	0.376 (0.362, 0.390)
NoIW-R+PC+Q	0.354	1.898	3.547	1.515 (1.464, 1.564)	1.858 (1.806, 1.910)	2.646 (2.589, 2.700)	0.038 (0.035, 0.042)	0.394 (0.378, 0.410)	1.333 (1.302, 1.365)	−1.161 (−1.212, −1.108)	0.040 (−0.016, 0.096)	0.901 (0.842, 0.962)	37.669 (36.572, 38.772)	31.714 (30.788, 32.640)	**33.563** (32.594, 34.548)	0.078 (0.070, 0.085)	0.059 (0.052, 0.066)	0.054 (0.047, 0.060)	0.372 (0.359, 0.387)	**0.399** (0.385, 0.413)	0.378 (0.364, 0.392)
NoIW-R+PC+RGB	0.354	1.898	3.547	1.462 (1.415, 1.508)	1.759 (1.712, 1.806)	2.523 (2.468, 2.576)	0.025 (0.022, 0.027)	0.347 (0.332, 0.362)	**1.285** (1.254, 1.316)	−1.108 (−1.155, −1.060)	0.139 (0.087, 0.189)	1.024 (0.966, 1.084)	53.785 (52.783, 54.794)	41.955 (41.038, 42.882)	40.293 (39.360, 41.224)	0.067 (0.059, 0.075)	0.052 (0.045, 0.058)	0.042 (0.036, 0.047)	0.384 (0.370, 0.398)	0.389 (0.375, 0.403)	0.369 (0.356, 0.383)
NoIW-R+PC+RGB+Q	0.354	1.898	3.547	1.297 (1.251, 1.342)	1.704 (1.654, 1.753)	2.543 (2.489, 2.595)	0.030 (0.028, 0.033)	0.425 (0.408, 0.441)	1.417 (1.385, 1.449)	−0.943 (−0.989, −0.897)	0.194 (0.142, 0.247)	1.004 (0.947, 1.063)	47.506 (46.410, 48.588)	39.554 (38.569, 40.565)	36.242 (35.263, 37.210)	0.069 (0.062, 0.076)	0.062 (0.055, 0.069)	0.046 (0.040, 0.052)	0.368 (0.354, 0.382)	0.375 (0.360, 0.389)	0.353 (0.340, 0.367)
NoIW-M	0.354	1.898	3.547	0.933 (0.896, 0.969)	1.330 (1.290, 1.370)	2.186 (2.141, 2.229)	**0.011** (0.010, 0.013)	0.346 (0.333, 0.359)	1.430 (1.400, 1.460)	−0.579 (−0.615, −0.542)	0.568 (0.524, 0.611)	1.361 (1.314, 1.410)	79.554 (79.043, 80.065)	66.182 (65.604, 66.768)	62.818 (62.155, 63.487)	0.062 (0.056, 0.069)	0.050 (0.043, 0.056)	0.029 (0.025, 0.033)	0.390 (0.376, 0.404)	0.379 (0.364, 0.392)	0.356 (0.341, 0.369)
NoIW-M+Q	0.354	1.898	3.547	0.933 (0.896, 0.969)	1.330 (1.290, 1.370)	2.202 (2.156, 2.244)	**0.011** (0.010, 0.013)	0.346 (0.333, 0.359)	1.445 (1.415, 1.475)	−0.579 (−0.615, −0.542)	0.568 (0.524, 0.611)	1.345 (1.299, 1.395)	79.554 (79.043, 80.065)	66.182 (65.604, 66.768)	62.931 (62.273, 63.603)	0.062 (0.056, 0.069)	0.050 (0.043, 0.056)	0.029 (0.025, 0.033)	0.390 (0.376, 0.404)	0.379 (0.364, 0.392)	0.354 (0.340, 0.368)
NoIW-M+RGB	0.354	1.898	3.547	0.902 (0.866, 0.937)	1.396 (1.355, 1.436)	2.512 (2.456, 2.566)	0.024 (0.021, 0.026)	0.400 (0.385, 0.415)	1.608 (1.574, 1.643)	−0.548 (−0.584, −0.511)	0.502 (0.458, 0.547)	1.035 (0.980, 1.093)	67.347 (66.413, 68.309)	59.661 (58.861, 60.466)	59.378 (58.660, 60.133)	0.080 (0.072, 0.088)	0.061 (0.054, 0.068)	0.033 (0.028, 0.038)	**0.406** (0.392, 0.420)	0.384 (0.370, 0.398)	0.353 (0.339, 0.366)
NoIW-M+RGB+Q	0.354	1.898	3.547	0.911 (0.874, 0.948)	1.394 (1.351, 1.435)	2.573 (2.515, 2.630)	0.024 (0.021, 0.026)	0.410 (0.394, 0.425)	1.644 (1.609, 1.678)	−0.557 (−0.594, −0.519)	0.504 (0.460, 0.549)	0.974 (0.917, 1.034)	66.198 (65.281, 67.131)	59.317 (58.523, 60.122)	58.941 (58.212, 59.682)	0.094 (0.086, 0.103)	0.064 (0.057, 0.071)	0.030 (0.026, 0.035)	0.390 (0.376, 0.404)	0.380 (0.365, 0.393)	0.356 (0.343, 0.370)
NoIW-M+PC	0.354	1.898	3.547	0.811 (0.780, 0.841)	1.245 (1.209, 1.282)	2.244 (2.194, 2.292)	0.034 (0.031, 0.037)	0.370 (0.357, 0.385)	1.414 (1.383, 1.447)	−0.457 (−0.488, −0.425)	0.652 (0.612, 0.693)	1.303 (1.251, 1.357)	37.865 (36.674, 39.002)	30.964 (29.981, 31.969)	38.521 (37.532, 39.517)	0.113 (0.103, 0.122)	0.114 (0.104, 0.123)	**0.091** (0.081, 0.100)	0.386 (0.372, 0.401)	**0.399** (0.385, 0.413)	0.376 (0.362, 0.390)
NoIW-M+PC+Q	0.354	1.898	3.547	**0.790** (0.760, 0.820)	**1.233** (1.197, 1.269)	2.213 (2.163, 2.262)	0.038 (0.034, 0.041)	0.379 (0.366, 0.393)	1.416 (1.385, 1.448)	**−0.436** (−0.466, −0.404)	**0.665** (0.626, 0.704)	1.334 (1.282, 1.387)	36.215 (35.059, 37.351)	31.431 (30.439, 32.446)	38.911 (37.929, 39.887)	0.115 (0.106, 0.124)	0.099 (0.090, 0.108)	0.076 (0.068, 0.083)	0.393 (0.379, 0.407)	**0.399** (0.385, 0.413)	**0.379** (0.364, 0.392)
NoIW-M+PC+RGB	0.354	1.898	3.547	0.929 (0.894, 0.965)	1.405 (1.362, 1.448)	2.435 (2.378, 2.490)	0.016 (0.014, 0.018)	0.375 (0.360, 0.389)	1.526 (1.494, 1.558)	−0.575 (−0.610, −0.539)	0.492 (0.446, 0.539)	1.112 (1.055, 1.171)	61.658 (60.620, 62.668)	51.595 (50.659, 52.541)	52.093 (51.216, 52.969)	0.100 (0.092, 0.109)	0.075 (0.068, 0.082)	0.041 (0.036, 0.046)	0.388 (0.374, 0.402)	0.385 (0.371, 0.400)	0.353 (0.339, 0.366)
NoIW-M+PC+RGB+Q	0.354	1.898	3.547	0.871 (0.837, 0.905)	1.322 (1.283, 1.362)	2.256 (2.203, 2.308)	0.046 (0.043, 0.050)	0.381 (0.367, 0.396)	1.377 (1.345, 1.410)	−0.517 (−0.551, −0.482)	0.576 (0.532, 0.620)	1.291 (1.236, 1.349)	**32.496** (31.347, 33.629)	**27.391** (26.415, 28.397)	34.923 (33.916, 35.928)	**0.145** (0.133, 0.157)	**0.121** (0.111, 0.131)	0.088 (0.079, 0.098)	0.383 (0.369, 0.398)	0.398 (0.384, 0.413)	0.374 (0.360, 0.388)
NoIW-Random	0.354	1.898	3.547	0.912 (0.890, 0.935)	1.273 (1.248, 1.298)	2.654 (2.619, 2.686)	0.048 (0.045, 0.051)	0.796 (0.778, 0.815)	2.263 (2.237, 2.289)	−0.558 (−0.583, −0.533)	0.625 (0.597, 0.651)	0.893 (0.861, 0.926)	13.775 (13.476, 14.069)	10.708 (10.431, 10.985)	10.677 (10.408, 10.946)	0.098 (0.088, 0.107)	0.072 (0.064, 0.080)	0.041 (0.035, 0.046)	0.365 (0.352, 0.380)	0.368 (0.354, 0.382)	0.364 (0.350, 0.378)
NoIW-ShortestPath	0.354	1.898	3.547	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.005 (0.004, 0.006)	0.349 (0.340, 0.358)	1.893 (1.876, 1.909)	3.542 (3.522, 3.563)	0.000 (0.000, 0.000)	0.000 (0.000, 0.000)	0.000 (0.000, 0.000)	0.581 (0.567, 0.595)	0.581 (0.567, 0.595)	0.581 (0.567, 0.595)	0.451 (0.437, 0.466)	0.451 (0.437, 0.466)	0.451 (0.437, 0.465)

Table 6: Tab. 6 (navigation results **without** inflection weightings) with 90% bootstrap confidence intervals