Supplementary Document

1 Local Scene Graph Prediction Network

For scene graph generation, we resort to the semantic point cloud information to build the scene graph. To get 3D semantic point cloud, the RGB frame is fed into the Mask RCNN model to get the instance segmentation of each pixel in the frame. The segmentation result is then aligned with the corresponding depth image and the 3D semantic point cloud can be obtained. For each time step t, the local scene graph prediction network based on current semantic point cloud is proposed to generate a local semantic scene graph ($\mathcal{LSSG}$). Specifically, the network consists of two point nets and a graph convolutional network. The generated $\mathcal{LSSG}$ includes the current in-sight objects and their relations with each other.

Specifically, the obtained semantic point cloud includes the point cloud and category information of the current observed instances. The goal of the local scene graph prediction network (LSGPN) is to generate a scene graph $G = (N, R)$, which describes the object nodes (N) in the scene as well as their relation edges (R). The architecture of the proposed LSGPN is shown in Fig.1. The network consists of three modules, namely embedding module, GCN module and predicting module. The embedding module consists of two PointNet and two MLPs, aiming at extracting features for every node and edge. For a scene s, we use the point cloud (P_i) of every instance i as the input to encode the node, and the merged point cloud (P_{ij}) of instance i and j as the input to encode the edge between instance i and j.

$$P_{ij} = \{p_k | p_k \in (P_i \cup P_j)\}_{k=1\cdots|P|} \tag{1}$$

where P represents the entire point cloud of the scene, $|P|$ represents the size of P. The output vector from the PointNet would be sent into the corresponding MLP to get the features representing the nodes and edges respectively. It is noted that we have the number of relationships between each pair of object categories in advance from the ground truth scene graph, and a 3-dimensional tensor $R \times N \times N$ is obtained, where R is the number of the relation types and N is the number of the object categories. Each tuple (r, i, j) corresponds to the number of type r relationship edge that connects from category i to j. We consider that there might be an edge between two instances of category i and j only if $(r_k, i, j) > 0, \forall k \in \{1, 2, \cdots, R\}$, which demonstrates that the categories of the two instances could have relations in some scenes. These possible edges help to build the proposals of the scene graph and each proposal contains the category information of the two instance, and the merged point cloud of the two instances.

We employ the GCN module to process the instances and proposed edges. We arrange the extracted features in triples as (subject, predicate, object), where ϕ_n occupy subject / object units, and edge features ϕ_r occupy the predicate units. Each convolutional layer consists of two MLPs and propagates the graph features in two steps. The first step projects each triple for information propagation, and at the second step the features of neighbors are averaged as the new feature for each node.

At the predicting module, features for each node and edge are fed into two MLPs respectively to predict the category of the node and the relation type of the edge. Note that since there are always more proposed edges than actual existing edges, we add $none$ into the relation types.

Since the complexity of the detected instances varies with the agent changing its position, we want the GCN to allow flexibility in the input number of instances and proposals. Each convolutional layer l of GCN consists of two MLPs and propagate the graph features in two steps. First, each triple, which contains two object and one proposed edge, is fed into MLP $f1(\cdot)$

$$(\psi_{s,ij}^{(l)}, \phi_{ij}^{(l+1)}, \psi_{o,ij}^{(l)}) = f1(\phi_i^{(l)}, \phi_{ij}^{(l)}, \phi_j^{(l)}) \tag{2}$$

where ψ represents the intermediate variables of the triple, s indicates the subject instance, o indicates the object instance. Second, for each node, we aggregate the signals coming from all the

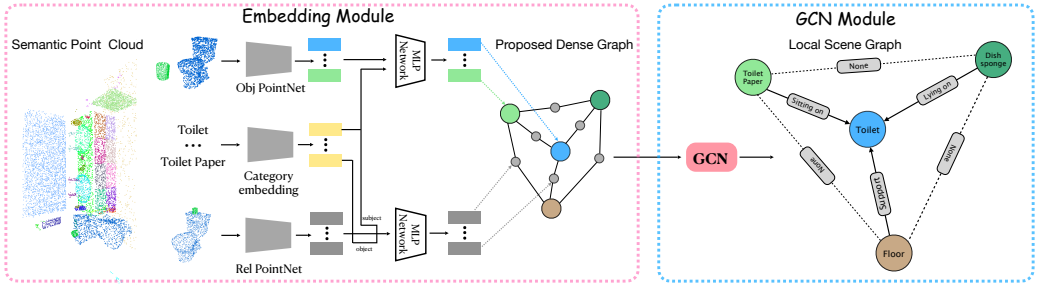

Figure 1: The architecture for the local scene graph prediction network (LSGPN).

connecting edges of the node (either as a subject or an object) together, compute the average value

$$\rho_i^{(l)} = \frac{1}{|R_{i,s}| + |R_{i,o}|}\left(\sum_{j \in R_s} \psi_{s,ij}^{(l)} + \sum_{j \in R_o} \psi_{o,ji}^{(l)}\right) \tag{3}$$

where $|\cdot|$ denotes cardinality and R_s and R_o are the set of the connecting edges in which node i is severed as subject and object respectively. The result is fed into another MLP $f2(\cdot)$ to generate the final representation for node i

$$\phi_i^{(l+1)} = f2(\rho_i^{(l)}) \tag{4}$$

The output features for each node and proposed edge are then propagated by the next convolutional layer $l+1$ in the same fashion. After processed by each layer, the feature of a node propagates to a further neighbour level. Hence, the visibility of the output feature for each node and edge is proportional to the depth of GCN.

At the predicting module, in order to let the network capture the semantic relations between each object and preserve the category and localization information of the objects at the same time, the feature vector for each node is fed into two fully connected layers respectively to predict the category and bounding box of the node, and the feature vector for each edge is fed into a fully connected layer to predict the relation type of the edge. Note that since there are always more proposed edges than actual existing edges, we add $none$ into the relation types.

We train our model in an end-to-end manner, optimizing an object classification loss L_{obj} as well as a relation classification loss L_{rel}

$$\mathcal{L} = \lambda_{obj}\mathcal{L}_{obj} + \lambda_{bbox}\mathcal{L}_{bbox} + \mathcal{L}_{rel} \tag{5}$$

where λ_{obj} and λ_{bbox} are weighting factors. Although the input instance point clouds already have labels and the classification accuracy of Mask RCNN is significantly higher than LSGPN, we believe that the model should still pay attention on objects and relationships to extract more informative features from the point clouds and obtain a better representation of the entire scene graph. Since our relation space mainly focuses on spatial relations, the relation number between two instances is at most 1. To deal with class imbalance, we use the weighted cross entropy loss for object category and relation type classification. The output edges ($none$ type excluded) and the subject and object instances together can compose a local scene graph which contains the observed instances and their relations with each other. In practice, the GCN consists of 5 convolutional layers, λ_{obj} is set to 0.5 and λ_{bbox} is set to 0.02, and we calculate the weights for each object category and relation type by an effective sample number based re-weighting method, where β_{obj} is set to 0.99 and β_{rel} is set to 0.99995. We traverse all the viewpoints in each scene and ask LSGPN to predict the corresponding local semantic scene graph in one epoch, and eventually train 100 epochs. We use SGD optimizer with a learning rate of 10^{-3}.

2 Global Scene Graph Generation

At each time step, the agent takes an action to move, then the equipped camera would capture the RGB and depth frames of scene to construct the local scene graph, which is further used to be merged into the global scene graph. During this period, the alignment between the detected objects

in the local scene graph and the existing objects in the previous global scene graph is extremely important.

Different from the *multi-view consistency* considered in other 3D scene graph construction works that have the multi-view visual information known in advance, in this work, we do not have any prior information about the concerned scene and the information of each object is incrementally collected and updated. Considering this situation, we propose a point cloud based weighted voting mechanism which utilizes the number of the pixels in the point cloud to measure the confidence of the object category. In practice, for each detected object, we maintain a score distribution across the predefined classes, the score is incrementally accumulated by the weighted value of the confidence provided by the detector and the size of the acquired point cloud.

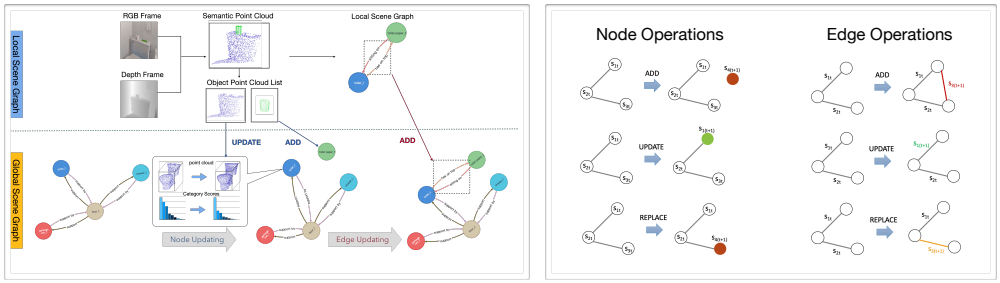

Figure 2: The illustration of updating nodes and edges in the global scene graph. **Left**: the alignments between objects of local scene graph and global scene graph. **Right**: the representative operations on nodes and edges. The change of color indicates the corresponding operation.

Then, we align the object point cloud with the nodes in global scene graph by computing the fraction of point cloud in the object point cloud that is inside the node's 3D bounding box. We calculate this fraction between each object point cloud and the node in the global scene graph. If the highest fraction of object point cloud and the corresponding node is higher than the preset threshold and this fraction is higher than those with other objects, the object and the node is considered aligned. This bidirectional alignment procedure could ensure that each aligned object and the corresponding node are in one-to-one correspondence. Fig.2 demonstrates the alignment process. Based on the aligning result for each object and the weighted voting mechanism, we could align the nodes and edges in the obtained Local Semantic Scene Graph(LSSG) with those in the Global Semantic Scene Graph(GSSG). Based on the aligning result for each object and the weighted voting mechanism, we take a two-step method to merge the obtained Local Semantic Scene Graph(LSSG) into the Global Semantic Scene Graph(GSSG).

Firstly, we update the nodes using the following three operations, as shown in the left panel in right half of Fig.2.

1. **ADD** If the detected object in LSSG has not aligned with any existing node in GSSG, we consider it as a newly appeared object and add it to the GSSG. The attribution of this node is initialized with the detected object.
2. **UPDATE** If the detected object in the LSSG aligns with an existing node in the GSSG, and the predicted category of this object is the same with the node, we update the attribution of this node and merge the object point cloud with the point cloud of the node.
3. **REPLACE** If the detected object in the LSSG aligns with an existing node in the GSSG, but the predicted category of the object is different with this node, we update the category of the node, delete its relation edges with other nodes and merge the point cloud at the viewpoints with the same detected class.

Based on the node updating result, we could perform the following three operations for the corresponding edges, as shown in the right panel in right half of Fig.2.

1. **ADD** If there is no edge between both nodes in the GSSG, we add the corresponding edge to it with the predicted relation type and the corresponding confidence.

2. **UPDATE** If the edge in the LSSG has the same relation type with the aligned edge in the GSSG, we update the confidence value of the edge with the one obtained in the LSSG.
3. **REPLACE** If the edge in the LSSG has different relation type with the aligned edge in the GSSG, we replace the edge in GSSG with one in the LSSG.

The above procedure coarsely describes the updating for the global semantic scene graph.

3 Scene Graph Encoder Implementation Details

We implement an attention-based encoder to encode a local semantic scene graph into a vector. Specifically, denote the embedding vectors of node and edge output from the last graph convolutional layer of LSGPN as $X_o = \{x_{o_1}, x_{o_2}, \cdots, x_{o_M}\}$ and $X_r = \{x_{r_1}, x_{r_2}, \cdots, x_{r_N}\}$, where M and N represents the number of object nodes and relation edges respectively. The attention module for node vectors in local semantic scene graph is defined as:

$$f_o = \sum_{i=1}^{M} \alpha_{o_i} x_{o_i}; \alpha_{o_i} = \frac{exp(\omega_o^T x_{o_i})}{\sum_m exp(\omega_o^T x_{o_m})} \tag{6}$$

where ω_o the learnable weight vector for the node vectors. Similarly, the attention module for edge vectors is:

$$f_r = \sum_{i=1}^{N} \alpha_{r_i} x_{r_i}; \alpha_{r_i} = \frac{exp(\omega_r^T x_{r_i})}{\sum_n exp(\omega_r^T x_{r_n})} \tag{7}$$

The final representation of the local semantic scene graph is the concatenation of the weighted sum vector of nodes and edges: $f_l = [f_o, f_r]$.

For the global semantic scene graph, we calculate the average vectors of the nodes and edges in global semantic scene graph respectively and combine them as the embedding of then global semantic scene graph. Specifically, since the point cloud for each node is becoming more complete with the agent exploring the scene, the vectors of each node and edge in local semantic scene graph generated at the later steps are more accurate, for the aligned node and edges, we maintain its moving average of embedding vector: $gssg\_vec = \alpha \cdot gssg\_vec + (1 - \alpha) \cdot lssg\_vec$, where α is set to 0.1 in practice.

4 Navigation Model Implementation Details

4.1 Imitation Learning

The goal of imitation learning for sequential prediction problem is to train the agent to mimic the expert's behaviors. In our case, to develop the imitation learning algorithm, the essential problem is to generate some demonstration paths for the agent to imitate. In this work, we adopt a two-stage method to deal with this problem. In the first stage, we try to obtain the waypoint set and in the second stage, we perform the interpotation between way-points to get the whole demonstration path. To evaluate the way-points, we first count the visible object at each viewpoint, and select the closest next way-point that has the most number of unseen new objects, which can be expressed as:

$$v^* = \underset{v \in O(v_c, k^*)}{\arg\max}\ new\_object\_num(v)$$

where $O(v_c, k)$ represents the feasible viewpoints that are k steps away from current viewpoint v_c, k^* is the minimal value for k which guarantees

$$\sum_{v \in O(v_c, k)} new\_object\_num(v) > 0,$$

and the way-point set can be updated as $\mathcal{W} = \mathcal{W} \cup \{v^*\}$. The above procedure is repeated until a maximum distance is achieved. In the second step, we implement a beam search over the key point sequence for interpolation aiming at observing the detected objects from as many viewpoints

as possible. With the obtained demonstration paths, we use the negative log likelihood loss to train our agent.

The loss function is defined as follows

$$\mathcal{L}_\theta = -\frac{1}{K}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\log \pi_\theta(\hat{a}_{k,t}|\hat{s}_{k,0}, \hat{a}_{k,0}, \hat{s}_{k,1}, \hat{a}_{k,1}, \cdots, \hat{s}_{k,t}) \tag{8}$$

where K is the number of demonstration paths used for training in one batch, T_k is the length of the k-th path, $\hat{s}_{k,t}$ and $\hat{a}_{k,t}$ are the annotated input visual state and action, and θ denotes the parameter of the exploration policy π. The process of minimizing $\mathcal{L}_\theta$ equals to maximize the probability of the demonstration paths' action sequence based on the annotated inputs.

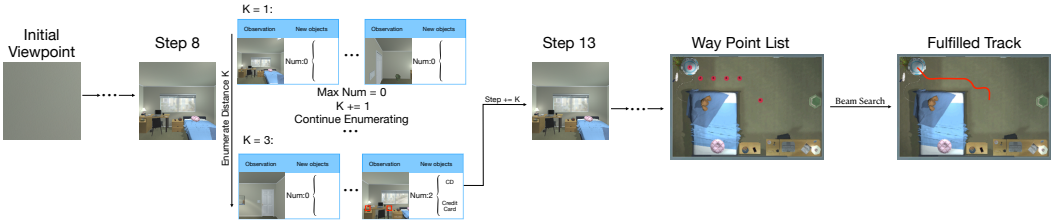

Figure 3: An illustration of the demonstration path generation.

4.2 Reinforcement Learning

After pre-training the exploration model with imitation learning, we then try to further improve its performance using the REINFORCE algorithm. The key to the reinforcement learning is the design of the reward function.

Since our task aims at formulating a scene graph for the entire scene, we directly measure the similarity between the generated scene graph and ground truth one, which can be obtained from the ground truth information about the object and the relationships. Given a global scene graph $\mathcal{GSSG}$ which is constructed from one path, the similarity score can then be calculated as a weighted sum of the precision and recall rate of the nodes and edges:

$$Sim(\mathcal{GSSG}) = \lambda_{node}(R_{node} + \lambda_p P_{node}) + R_{edge} + \lambda_p P_{edge} \tag{9}$$

where P_{node} and P_{edge} are the precision of the nodes and edges, respectively, while R_{node} and R_{edge} are the recall rate of the nodes and edges, respectively.

On the other hand, we hope to encourage the observation diversity when constructing the scene graph. The diversity can be characterized as the number of observation viewpoints of the detected objects. Concretely speaking, given a global scene graph $\mathcal{GSSG}$ which is constructed from one path, we use O to denote the set of the detected objects and calculate the diversity as

$$Div(\mathcal{GSSG}) = \sum_{o \in O} num\_viewpoints(o) \tag{10}$$

where $num\_viewpoints(o)$ is the number of the viewpoints about the detected object instance o.

Therefore we may formulate the score at time instant t as

$$p_t = Sim(\mathcal{GSSG}) + \lambda_d Div(\mathcal{GSSG}) - \rho t. \tag{11}$$

where the third term is used to penally the length of the path, and λ_d, ρ are the corresponding weighting parameters. In practice, we set $\lambda_{node} = 0.1$, $\lambda_p = 0.5$, and $\lambda_d = \rho = 0.001$.

According to the above definition, the immediate reward is designed to be the increment of the score $r(s_t, a_t) = p_t - p_{t-1}$ and the cumulative reward can be computed as

$$R(s_t, a_t) = r(s_t, a_t) + \sum_{t'=t+1}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}), \tag{12}$$

where $R(s_t, a_t)$ represents the expected accumulated reward when agent takes action a_t at state s_t, the discount parameter γ is set to 0.99, and T is the length of the action and state sequence with upper bound of 40 steps, and we use SGD optimizer with a learning rate of 10^{-4}.

5 Dataset Generation Details

To validate the performance of the *Embodied Semantic Scene Graph Generation* framework, we generate a new dataset with the AI2THOR, which contains 120 scenes (including 30 kitchens, 30 living rooms, 30 bedrooms and 30 bathrooms). Utilizing the object attribute information of AI2THOR, we divide the objects into two groups: Group A corresponds to larger objects which are always sitting on the floor or hanging on the wall. Group B corresponds to smaller objects which have to be placed on the object in Group A. We extract 16 semantic relationships (*support, support by, standing on, sitting on, lying on, has on top, above, below, close by, embedded on, hanging on, pasting on, part of, fixed on, connect with, attach on*), which can be clustered into the following seven categories.

1. **Support Relationship**: For the support relationship, we consider Group A, where one object is placed on the other one. For example, we have *floor* supports *desk* denoted as $\langle$*floor, support, desk*$\rangle$ or $\langle$*desk, supported by, floor*$\rangle$ in the scene graph. Since the point cloud of the object is noisy and incomplete, it is very challenging to extract the support relationship automatically. To this end, we calculate the 3D bounding boxes for each instance in the scene and project them into the *XY* plain. And then we look for situation where the projection of one object is completely inside the projection of another object staying below it. Also, the distance between the 3D bounding box of the two objects is less than 5mm. We additionally take a manual verification to eliminate wrong prediction and add missing support relationships. It is noted that we consider *floor*, *ceiling* and *walls* as inherent objects of each scene and they do not need any supporter.
2. **Placement Relationship**: Placement relationship is similar to support relationship. The main difference is that placement relationship defines the relationship between one object in Group A and one in Group B. We use the same rule to find the two objects. When a place relationship is found, we add two-way edges between the two objects (e.g. $\langle$*book, on, desk*$,\rangle$ and $\langle$*desk, has on top, book*$,\rangle$). To enrich the semantic diversity of the relations, we subdivide this *on* relation into three subclass: *standing on*, *sitting on* and *lying on*. We utilize the material of the Group B object and the category of Group A object to determine the subclass of relation *on*. For example, the metal statues and ceramic vases are *standing on* desks and shelves, the sponge pillows and teddy bears are always *lying on* sofas and beds. We design a table to determine the subclass of the placement relationships with prior knowledge.
3. **Hanging Relationship**: Hanging relationship is another kind of support relationship that does not accord with the support relation rule. For example, a mirror hanging on the wall. For hanging relationship, we collect the set of hangable objects and make a priority table to find their supporter. We subdivided *hanging* relation into four subclasses: *hanging on*, *pasting on*, *fixed on* and *embed on*. Based on prior knowledge and object category, we design a mapping table to determine the subclass of the hanging relationships.
4. **Position Relationship**: Since the agent moves and rotates during the exploration, the relative 2D relations like *left*, *right*, *in front* and *behind* may change over the time. Therefore, we preserve the *above*, *below* relations, which are relatively fixed. Meanwhile, we calculate the distances between the 3D bounding boxes of objects with the same supporter and add two-way *close by* edges for the two objects if their distance is short enough.
5. **Connecting Relationship**: Connecting relationship describes the interconnections between the same objects. This relation focuses on illustrating the relation between the adjacent cabinet units or shelf units.
6. **Attachment Relationship**: Attachment relationship refers to the relation between the attachment and the main part. For example, *faucet* is an attachment of *sink*, *stove burner knob* is an attachment of *stove burner*. We use the $\langle$*attachment*, *attach on*, *main part*$\rangle$ triples to illustrate this relationship.
7. **Component Relationship**: Component relationship aims at describing the relation between component parts and their corresponding main parts. For example, *drawer* could be

a component of *counter top* or *desk*. We use a the $\langle drawer, part\ of, counter\ top \rangle$ triple to depict this relationship.

To generate the ground-truth, we divide each scene into rectangle grids with a length of 0.25m and visit 8 angles with 45° interval at each grid vertex to collect the point cloud for each instance to generate the ground truth scene graph. For each scene category, we use 26 scenes for training, 2 for evaluation and 2 for test, which is 104 training scenes, 8 evaluation scenes and 8 testing scenes in total. Specifically, we select about 25 starting viewpoints (one-sixteenth of the total viewpoints) in each scene for evaluation and testing.

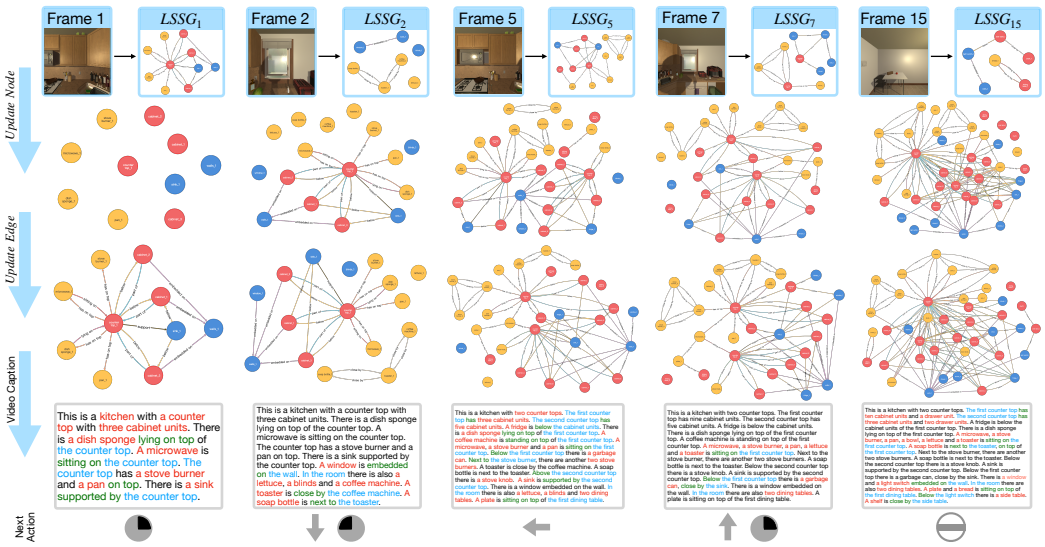

Figure 4: A representative result for our *embodied semantic scene graph generation* framework.

6 Qualitative Analysis on Streaming Video Captioning

We adopt a *Scene Graph based Captioning method* to generate caption from streaming video. Concretely speaking, this method generates an image caption for each key frame using a *seq2seq* model which is widely used in image captioning tasks. Then it utilizes yolo to detect the appeared objects at the frame and aligns them with the object phrases extracted from the image captions. If all the object phrases could be aligned with the detected objects, we consider the caption could depict the semantic relations of the appeared objects well and add it as a subsentence of the video caption and the corresponding nodes are marked as visited.

Since the adopted captions at different key frames are isolated with each other, we resort to the current global semantic scene graph to integrated all the previous frame captions. To this end, we conduct a BFS from the visited object nodes to find the shortest paths to the unvisited nodes. With the found subgraph of the global semantic scene graph, a template is designed to describe the triples of the subgraph and further generate captions that could describe the unvisited nodes. In this way, the captions between each frames are connected. For the last step, the method traverses the global scene graph and generates template-based sentences for all the unvisited nodes which are missed by the image captioning model. And the final video caption is generated.

Fig.4 shows the representative trajectory generated by the final navigation model trained with $LSSG + GSSG$ method. The agent takes 14 steps to explore the scene. From the vertical direction of the figure we can clearly see the updating process of nodes and edges of GSSG. We can see that categories of the nodes in the generated global scene graph are rather accurate, which demonstrates that the proposed weighted voting mechanism can effectively solves the imperfection of Mask RCNN. However, there is one error in this example where the Mask RCNN mistakenly detects an L-shaped counter top as two counter tops. However, this is not a common situation and it is hard to be fixed. The horizontal direction of Fig.4 exhibits the dynamic updating process of the global scene graph, which includes ADD, UPDATE and REPLACE operations over nodes and

From the above comparison between *Random* and our proposed method with the same starting viewpoint, we can intuitively find the performance gap. Since *Random* method chooses to stop at frame 8, we demonstrate the generated captions at frame 2, 3, and 8 of both methods for fairness purpose. *Random* method detects few objects from the scene and generates a semantically poor caption while our proposed framework extracts comprehensive objects and diverse semantic information from the scene, which reveals a more informative and accurate caption.

Figure 5: A qualitative comparison between $Random$ and our $LSSG{+}GSSG$ mehtod. We can see from the constructed global semantic scene graph and generated video caption that our methed captures much informative information from the scene and thus obtain a more comprehensive caption.

edges. With the local scene graphs generated at each step, the global scene graph is incrementally constructed and contains more and more semantic information.

We also find the steps where an node is added and its corresponding relationship is added are different (e.g the $lettuce$ in Fig.4). This phenomenon well illustrates the gap between detecting the object and extracting the semantic relations of the object in the scene. With the learned navigation policy, the agent first explores area where the objects are placed on the counter top and the $lettuce$ is detected at frame 2, while no relationship is detected at this time. And then the navigation policy guides the agent to explore nearby areas, and other small objects placed on the counter top can be detected. For further exploration, the policy guides the agent to move away from current viewpoint and an overall view of the counter top is captured, which corresponds to frame 7. At the same time, the relation between $lettuce$ and $counter\ top$ is finally extracted from the visual input and added to the global scene graph. In this example, we can see that the white dining table is also detected in the first several steps. With the learned policy, the agent is able to explore the nearby areas around the counter top first and then move to the dining table for further exploration. It can be seen that in frame 15, the bread, side table and shelf which are invisible from previous viewpoints are detected. After adding these newly detected objects, the agent considers that it has fully explored the scene and the stop action is triggered.

From the above examples we can find that the navigation policy would guide the agent to detect more objects and extract more semantic relationships at the same time with a relatively short path. Furthermore, the agent leverages its embodiment ability to trigger stop action when it thinks that the scene is fully explored.

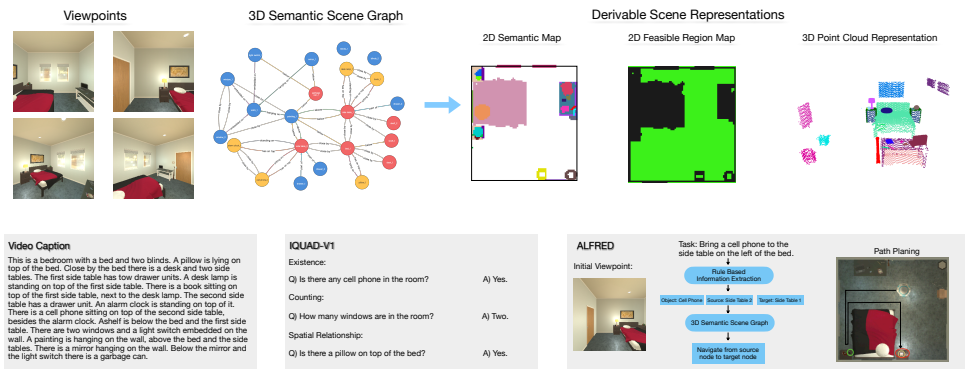

Figure 6: An illustration for more downstream applications with the generated semantic scene graph.

7 Application to Downstream Tasks

As shown in Fig.6, utilizing the point cloud for each node, we can obtain many common spatial representation of the scene, including 2D semantic map, 2D feasible region map and 3D semantic point cloud. Further, the rich semantic information of the global semantic scene graph can help the agent complete task including video captioning, interactive question answering and complicated manipulation commands in ALFRED. Specifically, for the common questions in QA dataset like IQUAD-v1, the agent could traverse the nodes in $\mathcal{GSSG}$ and try to align the phrase to answer the existence and counting type questions. For the spatial relationship question, the agent would ground the subject and objects phrases with the node and compare the edge between the nodes with the queried relation phrase. To solve the more complicated tasks in ALFRED, the agent could first ground the source and target object in the command with the graph nodes, and then utilizing common navigation methods like path planning with the 2D feasible region map of scene to finish the task.