Scene-aware Activity Program Generation with Language Guidance

ZEJIA SU, Shenzhen University, China
QINGNAN FAN, Vivo, China
XUELIN CHEN, Tencent AI Lab, China
OLIVER VAN KAICK, Carleton University, Canada
HUI HUANG, Shenzhen University, China
RUIZHEN HU*, Shenzhen University, China

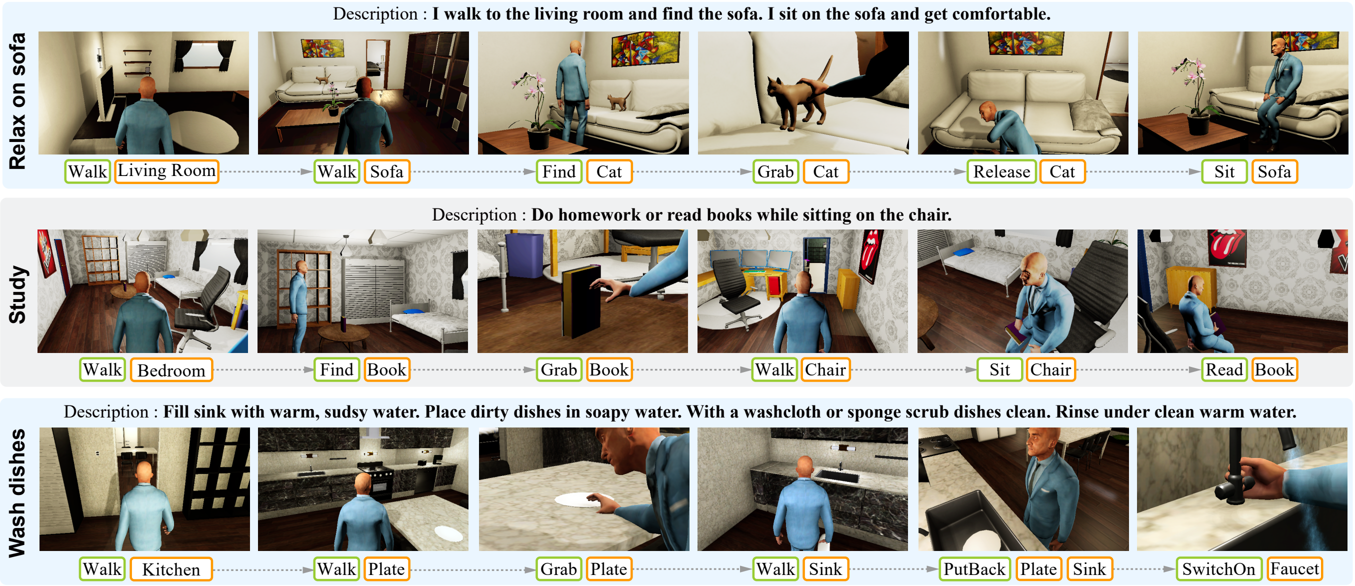

Description : I walk to the living room and find the sofa. I sit on the sofa and get comfortable.
Relax on sofa
Walk
Living Room
Walk
Sofa
Find
Cat
Grab
Cat
Release
Cat
Sit
Sofa
Description : Do homework or read books while sitting on the chair.
Study
Walk
Bedroom
Find
Book
Grab
Book
Walk
Chair
Sit
Chair
Read
Book
Description : Fill sink with warm, sudsy water. Place dirty dishes in soapy water. With a washcloth or sponge scrub dishes clean. Rinse under clean warm water.
Wash dishes
Walk
Kitchen
Walk
Plate
Grab
Plate
Walk
Sink
PutBack
Plate
Sink
SwitchOn
Faucet

Fig. 1. Our method generates scene-aware activity programs that are highly rational and executable. Here, we show three programs generated by our method, where a virtual avatar is instructed to perform various human activities in different scenes according to the input descriptions.

We address the problem of scene-aware activity program generation, which requires decomposing a given activity task into instructions that can be sequentially performed within a target scene to complete the activity. While existing methods have shown the ability to generate rational *or* executable programs, generating programs with both high rationality and executability still remains a challenge. Hence, we propose a novel method where the key idea is to explicitly combine the language rationality of a powerful language model with dynamic perception of the target scene where instructions are executed, to generate programs with high rationality *and* executability. Our method iteratively generates instructions for the activity program. Specifically, a two-branch feature encoder operates on a language-based and graph-based representation of the current generation progress to extract language features and scene graph features, respectively. These features are then used by a predictor to generate the next instruction in the program. Subsequently, another module performs the predicted action and updates the scene for perception in the next iteration. Extensive evaluations are conducted on the VirtualHome-Env dataset, showing the advantages of our method over previous work. Key algorithmic designs are validated through ablation studies, and results on other types of inputs are also presented to show the generalizability of our method.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Artificial intelligence**.

Additional Key Words and Phrases: activity program generation, language guidance, dynamic scene graph

ACM Reference Format:
Zejia Su, Qingnan Fan, Xuelin Chen, Oliver van Kaick, Hui Huang, and Ruizhen Hu. 2023. Scene-aware Activity Program Generation with Language Guidance. *ACM Trans. Graph.* 42, 6 (December 2023), 16 pages. https://doi.org/10.1145/3618338

*Corresponding author: Ruizhen Hu (ruizhen.hu@gmail.com).

Authors' addresses: Zejia Su, zejiasu.36@gmail.com, Shenzhen University, China; Qingnan Fan, fqnchina@gmail.com, Vivo, China; Xuelin Chen, xuelin.chen.3d@gmail.com, Tencent AI Lab, China; Oliver van Kaick, ovankaic@gmail.com, Carleton University, Canada; Hui Huang, hhzhiyan@gmail.com, Shenzhen University, China; Ruizhen Hu, ruizhen.hu@gmail.com, Shenzhen University, China.

0730-0301/2023/12-ART $15.00
https://doi.org/10.1145/3618338

1 INTRODUCTION

Digital agents are an important ingredient of virtual universes. A high-quality AAA game, for example “The Witcher 3: Wild Hunt”, typically includes dozens or even hundreds of digital agents, also known as Non-Player Characters (NPC), with each executing a pre-defined sequence of instructions to perform a designed activity. These instructions, along with the corresponding animations, usually require a significant amount of effort to handcraft. In addition, with advancements in robotics, there have been high hopes for empowering embodied agents, also known as humanoid robots, with the intelligence to undertake household duties. For these reasons, the automatic generation of instruction programs for common human activities has become an important research problem.

In this work, we address the specific problem of *scene-aware* activity program generation, which seeks to decompose an activity task into a sequence of instructions that can be carried out in a target scene. For example, for the “Wash dishes” activity shown in Figure 1, instructions are generated to guide the agent to walk to the kitchen, grab the plate, and then put it back in the sink for washing.

The key challenge of scene-aware activity program generation is to guarantee the *rationality* and *executability* of the generated programs. Specifically, a *rational* program is comprised of categorically reasonable instructions that accomplish the desired task, while an *executable* program can be successfully executed on the target scene. Thus, rationality is related to task completion at the semantic level while executability is related to the application of the program to a scene instance with a specific set of objects.

In order to achieve rationality or executability in activity program generation, the past solutions to this problem fall into two groups. One group of methods resort to large language models (LLMs) trained on open-source text corpora [Huang et al. 2022; Lu et al. 2022]. LLMs are usually parameterized with billions of weights and trained with millions of linguistic data. Leveraging these strong priors learned over large text corpora in activity program generation confers good rationality on the generated programs. However, common LLMs do not take the target scenes into explicit consideration, hence the generated programs are often at odds with the objects within the target scene, resulting in failure when executing the instructions. The other group of methods learn from pairs of activities and programs conditioned on the target scene, training neural networks in a supervised manner to predict program instructions for a desired activity, so that the instructions are consistent with the target scene [Liao et al. 2019; Tuli et al. 2021]. While instructions generated from these models have better executability compared to the LLM-based methods, with the absence of linguistics priors, these methods tend to fail on complex tasks due to the lack of rationality of the programs.

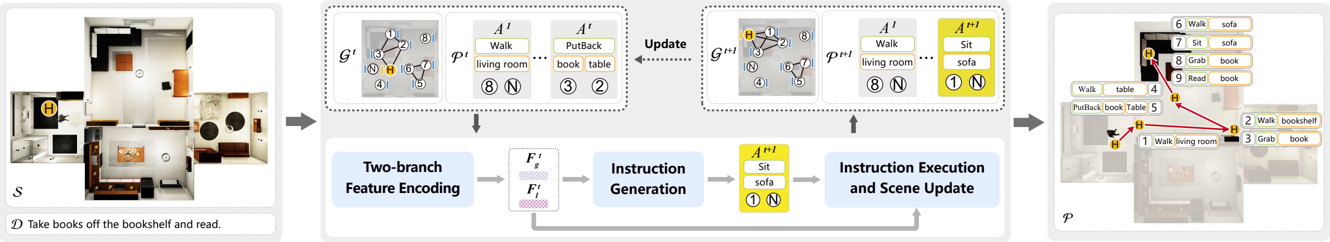
Our key idea to address these limitations is to explicitly leverage the language rationality of pretrained language models and combine it with dynamic perception of the target scene where instructions are executed, to obtain the best of both worlds, generating programs with high rationality and executability. In particular, we incorporate *language guidance* with a pre-trained language model, which leverages object semantics and hence increases the category-level inference ability for better rationality. On the other hand, we enable *scene-aware* program generation by explicitly representing the scene as a graph that is iteratively updated during the instruction prediction and execution. This greatly improves the perception of the dynamically-changing scene and leads to higher executability.

Since LLMs and scene perception cannot be trivially combined, we introduce a new method composed of novel modules that enable the combination of language- and graph-based features for instruction generation. More specifically, our method consists of three functional modules (Figure 2) that synthesize the instructions of a program step by step to accomplish the desired activity. At each iteration, given an activity description, a scene graph and a partial program, our *two-branch feature encoding* module encodes the description and partial program into: (1) A language feature that combines the semantic information of nodes in the scene graph via a pre-trained language model, and (2) A graph feature that embeds the latent features of nodes in the scene graph. Then, our *instruction generation* module derives the next instruction in an object-centric manner. This is accomplished by formulating object selection as a classification problem over all the object instances in the scene graph. This module fuses three instance probabilities, namely graph-guided probability, language-guided probability, and human-centric probability, to guide the instruction generation. In the end, the *instruction execution and scene update* module performs the actions on the corresponding objects, and updates the scene topology and properties accordingly for the generation of the next instruction.

We conduct experiments on the VirtualHome-Env dataset to demonstrate the effectiveness of our algorithm. The results show that our method significantly improves the rationality and executability of the generated programs. Specifically, we improve executability from 0.577 to 0.746 (29% improvement) over the best SOTA zero-shot language-based method, and rationality from 0.348 to 0.438 (26% improvement) over the best SOTA graph-based method. We also improve significantly the completeness of the generated programs, which measures whether the generated programs complete the specific activities in the given scene, increasing completeness from 0.442 to 0.584 (32% improvement) when compared to the SOTA method with best completeness results. In addition, we analyze qualitative examples to demonstrate the improved rationality and executability, and provide ablation studies and an analysis of the generalizability of the method to other data.

In summary, our contributions are several-fold:

- We propose a novel scene-aware activity program generation approach with language guidance, which confers rationality and executability on the generated programs.
- We devise three functional modules, namely, two-branch feature encoding, instruction generation, and instruction execution and scene update, that collectively contribute to an effective program generation method.
- Experiments demonstrate the excellent executability and rationality of the programs generated by our algorithm compared to existing methods.



(a) Input

(b) Iterative Program Generation and Scene Update

(c) Output

Fig. 2. Overview of our activity program generation method, which takes as input (a) an activity description and a scene with a human agent indicated by the yellow H node. (b) At each iteration, the *two-branch feature encoding* module first encodes the current state information, including the scene state, partial program, and description of the desired activity, to guide the *instruction generation* module in predicting the next instruction and target object. Finally, the *instruction execution and scene update* module updates the scene topology and properties based on the execution of the predicted instruction. The method iterates until (c) the final result with the full program and corresponding motion sequence is generated.

2 RELATED WORK

Activity program generation. Early work on activity program generation uses Markov Random Fields (MRF) or probabilistic grammars to parse high-level instructions or demonstrations into action plans [Nyga and Beetz 2012; Yang et al. 2014, 2015]. This line of work is limited to a small set of activities due to the complexity of the methods. Recent efforts in activity program generation fall mainly into two categories. The first category of methods relies on a pre-trained language model to extract high-level semantic features of the text description for task decomposition. Jansen [2020] modeled the translation problem of converting natural language directives into detailed multi-step sequences of actions. Huang et al. [2022] proposed a zero-shot planner that conditions existing demonstrations and semantically translates the plans into admissible actions. Lu et al. [2022] introduced a neuro-symbolic procedural PLANner (PLAN) that elicits procedural planning knowledge from a large language model (LLM) with commonsense-infused prompting. The aforementioned approaches do not have the ability of explicit 3D perception of the target environment and excel at improving the rationality of the program, but not the executability.

The second category of methods condition the activity program generation problem on the target scene. Puig et al. [2018] created the VirtualHome dataset by crowd-sourcing programs for a variety of activities that happen in people's homes, and provided a number of 3D indoor scenes for executing the programs in a virtual environment. Based on this dataset, Liao et al. [2019] addressed the problem of environment-aware program generation. They proposed ResActGraph, a network that generates a program from an environment graph, where a node representing an object is selected for interaction at each step. TANGO [Tuli et al. 2021] enhances ResActGraph [Liao et al. 2019] with commonsense knowledge from ConceptNet [Speer et al. 2017] and a goal-conditioned attention mechanism to help decode the programs. However, since TANGO takes as input the goal state of the scene, it is difficult to extend the method to derive activity programs from input descriptions. In general, due to the lack of a more sophisticated language prior to facilitate the program generation, all the approaches discussed in this paragraph are good at improving the executability of a program in a target 3D scene, but not the rationality.

Language grounding in 3D Scenes. Prior work has also investigated how to ground natural language in 3D scenes for many other tasks. Early methods [Artzi and Zettlemoyer 2013; Misra et al. 2015, 2016; Tenorth et al. 2010] relied on ruled-based lexical analysis or semantic parsing to resolve linguistic ambiguities in language instructions. These methods exhibit limited generalization ability on complex tasks and scenes. Due to the significant development of natural language modeling, pre-trained language models have performed well on various tasks related to 3D object/scene perception, such as object manipulation [Lynch and Sermanet 2020a,b], navigation [Fried et al. 2018; Majumdar et al. 2020; Wang et al. 2019], and 3D question answering [Lei et al. 2021; Ye et al. 2022]. Benefiting from the significantly increased network parameters and text data, recent work directly leverages large language models [Brown et al. 2020; Chen et al. 2021; Wei et al. 2021] on high-level task planning for embodied agents without any further fine-tuning [Ahn et al. 2022; Huang et al. 2022]. This class of work has performed well in large-scale simulated environments for embodied AI such as procedurally-generated [Deitke et al. 2022; Kolve et al. 2017; Li et al. 2023] and real scanned environments [Ramakrishnan et al. 2021].

Dynamic graph learning. Dynamic graph learning [Rossi et al. 2020] updates the features of nodes in a graph according to instant graph changes and can be separated into two categories: discrete-time dynamic graph learning (DTDGL) and continuous-time dynamic graph learning (CTDGL). DTDGL learns the node embedding by aggregating information of graph snapshots from different time steps [Lu et al. 2019; Manessi et al. 2020; Sankar et al. 2020]. On the other hand, CTDGL aims to capture the temporal evolution pattern of the graph and dynamically updates the node features in the continuous time domain [Rossi et al. 2020; Trivedi et al. 2019; Zhang et al. 2020]. Inspired by these approaches, we propose to update the topology and node features of the scene graph according to instantaneous instruction execution in the target 3D scene.

3 OVERVIEW

Given an activity description $\mathcal{D}$, a target scene $\mathcal{S}$, and a human agent $\mathcal{H}$, our goal is to generate a scene-specific program $\mathcal{P}$ that can be used to instruct the human agent to perform a sequence of actions on the object instances in the given scene to accomplish

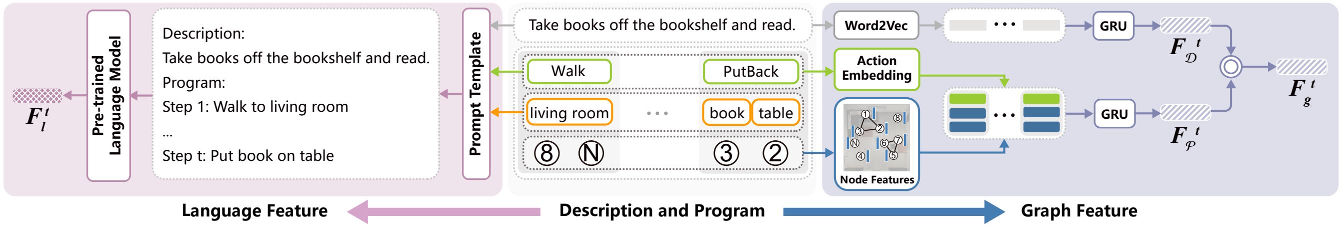


Fig. 3. Two-branch feature encoding: An activity description and partial program (center column) are encoded into a language feature F_l^t (left) through a pre-trained language model, and into a graph feature F_g^t (right) through action embedding and node features extraction.

the desired activity. As the states of both the human agent and the objects that the agent interacts with will get updated during program execution, and in turn affect the decision-making process, we adopt an online program generation approach to synthesize the instructions. Figure 2 provides an overview of our method.

Our method operates in an iterative manner and synthesizes one instruction per iteration based on the dynamic changes of the scene and agent states (Figure 2 (b)). Specifically, after t iterations of the method, we will have generated a partial program $\mathcal{P}^t$ consisting of t instructions, each of which is composed of one atomic action and at most two object instances in the scene that the agent interacts with. The current scene and human agent are encoded into a scene graph $\mathcal{G}^t$, with each node associated with an object or human instance along with its corresponding properties, and connectivity determined by spatial adjacency. The activity description $\mathcal{D}$ is kept fixed during the whole process.

Then, the *two-branch feature encoding* module encodes the partial program $\mathcal{P}^t$, the current scene graph $\mathcal{G}^t$, and the description $\mathcal{D}$ into a *language feature* F_l^t and *a graph feature* F_g^t. These two features are used by the *instruction generation* module to predict the next instruction and target object(s), so that we are able to incorporate guidance from a pre-trained language model and the current scene graph into the instruction generation. Finally, the *instruction execution and scene update* module executes the predicted instruction and updates the scene graph in preparation for the next iteration. The method iterates until a *stop* instruction is generated.

The output of the method is the complete, scene-aware program for carrying out the specified activity in the input scene (Figure 2 (c)). The sequence of instructions in the program can also be interpreted as a motion sequence of the agent in the scene.

4 ITERATIVE INSTRUCTION GENERATION

In this section, we discuss our method in more detail.

Input and output of each iteration. Each iteration of our method takes as input the current scene graph $\mathcal{G}^t$, the activity description $\mathcal{D}$, and the partial program $\mathcal{P}^t$, and synthesizes one instruction which is appended to $\mathcal{P}^t$. The scene graph at step t is notated as $\mathcal{G}^t = (V^t, E^t)$, where $V^t = \{v_i^t\}_{i=1}^{N_V}$ denotes a set of N_V nodes and $E^t = \{e_{ij}^t | d(v_i^t, v_j^t) \leq \delta\}$ denotes the edges connecting nodes whose physical distance $d(v_i^t, v_j^t)$ is less than a certain threshold δ. Each node v_i^t is represented by $v_i^t = (c_i, f_i, s_i^t, z_i^t)$, where c_i denotes the category label of the corresponding object, f_i denotes the affordance, such as "grabbable" or "sittable", s_i^t denotes the instance state, such as "on" or "off" for a TV, and z_i^t is a latent feature. Note that both the category label c_i and affordance f_i of each instance are fixed during the whole process, while the state s_i^t is updated if executable actions are performed on the instance and the feature z_i^t is updated at each iteration. Each edge has a property $e_{ij}^t \in \{$"on", "inside", "close to", "face at", "between"$\}$ that represents the spatial relation between two nodes.

The current program $\mathcal{P}^t$ consists of a sequence of instructions $\mathcal{P}^t = \{A^k\}_{k=1}^t$, where each instruction is further denoted as $A^k = (a^k, v_{i\_1}^k, v_{i\_2}^k)$, with a^k representing the atomic action selected from a predefined action set, and $v_{i\_1}^k$ and $v_{i\_2}^k$ representing the two objects selected from V^t. Thus, the scene graph information $\mathcal{G}^t$ is incorporated as part of the program $\mathcal{P}^t$ for later feature encoding. At the conclusion of one iteration, a new instruction A^{t+1} is generated, which is appended to $\mathcal{P}^t$.

Termination of iterations. The scene graph also includes an extra "stop" node and the atomic action set includes a "stop" action. We terminate the program generation if a stop node or action is selected. We also set the maximal number of generated instructions to 25.

4.1 Two-branch feature encoding

At each iteration, to enable better rationality and executability of the next instruction generated, we extract two types of features from the description $\mathcal{D}$ and partial program $\mathcal{P}^t$: a *language feature* F_l^t focusing on the semantic information extracted from a pre-trained language model, and a *graph feature* F_g^t focusing on the learned node embedding extracted from the scene graph (Figure 3).

Language feature. To aid the program generation with semantic information, we transform the partial program $\mathcal{P}^t$ together with the description $\mathcal{D}$ into a natural language paragraph, and obtain a program feature using the pre-trained GPT-2 language model [Radford et al. 2019], as shown in Figure 3 (left). In detail, we first translate each instruction A^k, in the form of the action label a^k and the category labels of the involved objects $(c_{i\_1}^k, c_{i\_2}^k)$, into a template sentence A_c^k following [Huang et al. 2022]. Then we gather all the translated instruction sentences $\mathcal{P}_c^t = \{A_c^k\}_{k=1}^t$ and the description $\mathcal{D}$ together, and feed them into a prompt template, to generate a category-aware long prompt. By feeding the prompt into GPT-2, we obtain a sequence of contextualized representations for each word in the prompt. Thanks to the causal attention mechanism in

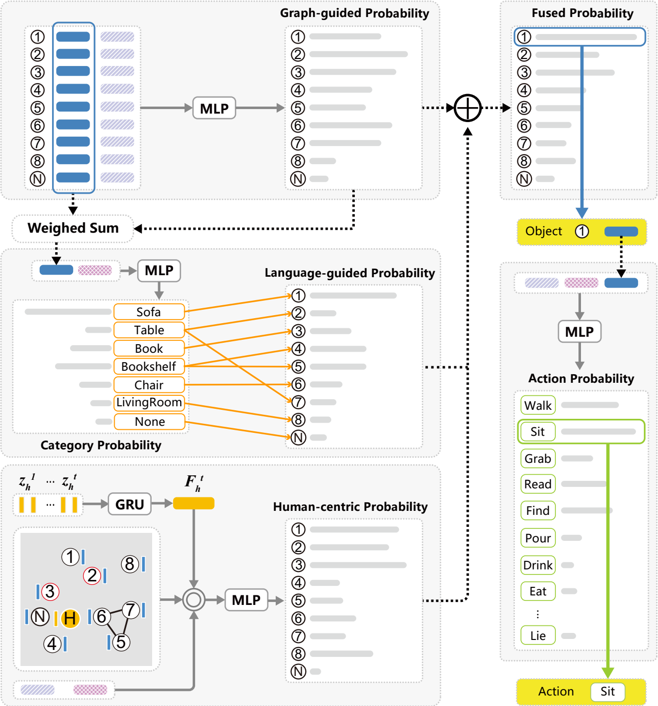


Fig. 4. Instruction generation: a *graph-guided probability* (top-left), *language-guided probability* (middle-left), and *human-centric probability* (bottom-left) are fused together to guide the selection of an object (top-right) and action (bottom-right) that compose a program instruction.

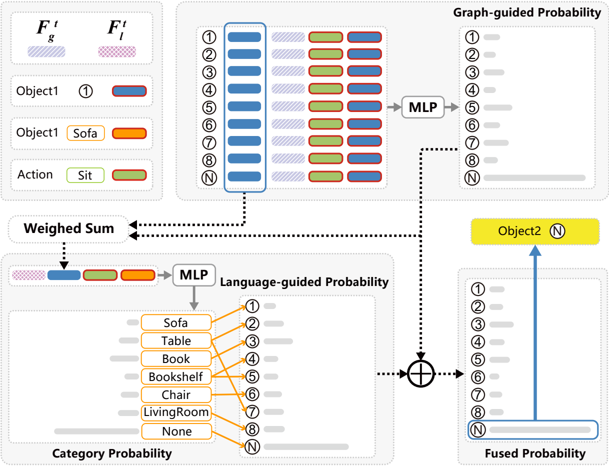


Fig. 5. Prediction of the second object for interaction based on global features and conditioned on the action and object already selected.

GPT-2, the latent representation of the last word incorporates the information of all the sentences, and hence is treated as the language feature F_l^t.

Graph feature. To make the program generation scene-aware, we further extract scene-related information from the partial program $\mathcal{P}^t$ and the description $\mathcal{D}$, as shown in Figure 3 (right). In detail, for each instruction A^k in the program, we obtain its feature encoding F_A^k by feeding the concatenation of the action embedding $\mathcal{E}_a(a^k)$ and two instance features $(z^k_{i\_1}, z^k_{i\_2})$ from the scene graph into a Multi-Layer Perceptron (MLP) network. The embedding of each action is randomly initialized and then optimized during the training, while the instance features are the node features extracted from the current graph $\mathcal{G}^t$. All the instruction features are further passed to a gated recurrent unit (GRU) [Cho et al. 2014] to obtain the instance-aware program feature $F_{\mathcal{P}}^t$. Likewise, for the description $\mathcal{D}$, we first convert it into a sequence of word embeddings [Mikolov et al. 2013], and then feed them to another GRU to obtain the description feature $F_{\mathcal{D}}^t$. Note that the word embeddings also get updated during the training. Finally, $F_{\mathcal{P}}^t$ and $F_{\mathcal{D}}^t$ are concatenated and passed through an MLP network to obtain the desired graph feature F_g^t.

4.2 Instruction generation

With the aforementioned features F_l^t and F_g^t as input, we utilize an instruction generation network to derive the next instruction $A^{t+1} = (a^{t+1}, v^{t+1}_{i\_1}, v^{t+1}_{i\_2})$. Actions to be performed usually have a strong dependency on the state of the objects involved in the action, for example, to "Watch TV", we need to identify the TV and "turn it on" first if its current state is "off". Hence, we opt to predict the instruction in an object-centric manner, where we select the object instance involved in the action first, and then the appropriate action as shown in Figure 4.

We formulate object selection as a classification problem over all the object instances in the current scene graph, where the probability of an instance is obtained by fusing three distinct probabilities: a *graph-guided probability* P_g^t based only on the graph features, a *language-guided probability* P_l^t considering semantic inference based on the language features, and a *human-centric probability* P_h^t driven by the link probability to the human agent. We first introduce the three types of instance probabilities for object selection and then elaborate more on how to generate the next instruction with the estimated probabilities.

Graph-guided probability. The graph-guided probability is predicted using all the graph features, including the instance-wise latent feature z^t and the global graph feature F_g^t. For each instance i, its graph-guided instance probability $P_{g,i}^t$ is obtained by feeding the concatenation of z_i^t and F_g^t to a shared MLP network.

Language-guided probability. To further guide the object selection using semantic information, we first infer a category probability P_c^t, which represents the probability of each category to be interacted with at the semantic level. Then, we covert it into the desired language-guided instance probability P_l^t by mapping the category probability to each object instance in the scene according to their category label.

In order to estimate the category probability, a straightforward solution is to directly leverage the aforementioned language feature F_l^t, which however lacks key information with respect to the scene and hence may hinder the performance of object selection. Thus, we propose to extract the global scene information from the

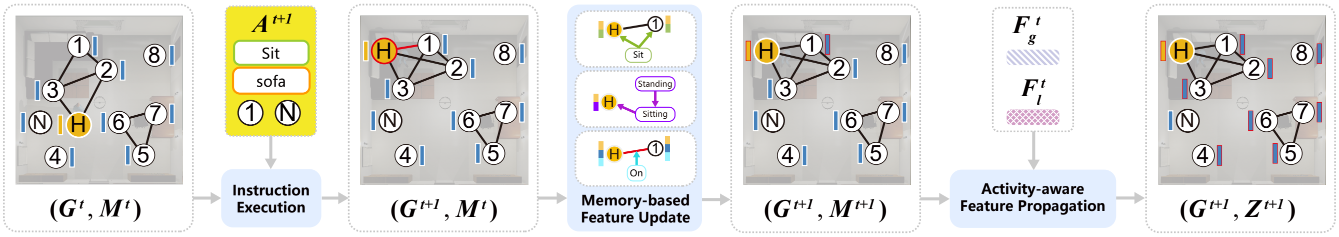


Fig. 6. Instruction execution and scene update: given the current scene graph, memory, and instruction, the instruction is executed to modify the graph, and the node features are updated in the *memory-based feature update* and then in the *activity-aware feature propagation*.

graph-guided probability first, and then fuse it with the language feature to obtain an estimate for the category probability. To be specific, we conduct a weighted sum of the latent node representation $Z^t = \{z_i^t\}_{i=1}^{N_V}$ based on the node-wise graph-guided probability P_g^t to produce the global scene feature F_S^t. Then, we concatenate F_S^t and F_l^t to predict the category probability $P_c^t = \{P_{c,i}^t\}_{i=1}^{N_C}$ via an MLP network, where N_C denotes the number of categories.

Human-centric probability. To constrain the object selection in terms of the adjacency of the human agent node, we further predict the human-centric probability, which indicates the link probability of the human agent node to each object node in the next time step. We first obtain a historical human feature F_h^t by feeding all of the historic node features $\{z_h^k\}_{k=1}^t$ into a GRU. Then F_h^t is concatenated with the graph feature F_g^t, language feature F_l^t, as well as the instance-wise latent feature z_i^t to predict the corresponding human-centric instance probability P_h^t through an MLP network.

Instruction generation. We mix the three instance probabilities to obtain the fused instance probability P_f^t, and select the instance with the highest probability as the object $v_{i\_1}^{t+1}$ for interaction. Specifically, we sum up the distribution of the three instance probabilities which are not normalized, and then normalize the resulting distribution to produce the fused instance probability P_f^t. The reason for this design is that the network can adaptively learn how these three probabilities should be combined by adjusting the scale of values in their predicted distributions.

To select the appropriate action, we estimate the action probability P_a^t from the feature obtained by concatenating the global graph and language feature (F_g^t, F_l^t) along with the latent feature of the selected object $z_{i\_1}^{t+1}$. Then, the action candidate with the highest probability is chosen as action a^{t+1}.

Some instructions involve a second object $v_{i\_2}^{t+1}$ for interaction (*e.g.*, "make coffee with milk"), which is predicted in a similar way as the first object $v_{i\_1}^{t+1}$. As shown in Figure 5, the estimation is conditioned on the selected action a^{t+1} and object $v_{i\_1}^{t+1}$, but ignores the effect of the human-centric probability which mainly guides the selection of the first object. The final object $v_{i\_2}^{t+1}$ is selected based on the maximum fused probability. Note that in most of cases, only one object is needed to perform the selected action. Then, the second object is assigned with a special "none" node for padding the instruction.

4.3 Instruction execution and scene update

After predicting the instruction $A^{t+1} = (a^{t+1}, v_{i\_1}^{t+1}, v_{i\_2}^{t+1})$, we need to let the human agent execute this instruction and update the scene accordingly for the next instruction generation, as shown in Figure 6. During the instruction execution, the properties of the selected object instances $(v_{i\_1}^{t+1}, v_{i\_2}^{t+1})$ and the corresponding topology in the scene graph are directly affected and should have an effect in the generation of the next instruction. Experimentally, we also observed that propagating such direct effects to the entire graph benefits the global understanding of human activities.

Thus, we maintain two variants of node features, similar to the work of TGN [Rossi et al. 2020]. One feature records all the direct effects of instruction execution and is called the memory $M^t = \{m_i^t\}_{i=1}^{N_V}$. The other feature, obtained after propagation over the entire graph, is the node feature Z^t we used in previous sections for instruction prediction. We refer to the process of updating from M^t to M^{t+1} as *memory-based feature update* and that of propagating from M^{t+1} to Z^{t+1} as *activity-aware feature propagation*, illustrated in Figure 6.

Instruction execution. The instruction execution is performed using the rule-based simulator proposed in VirtualHome [Puig et al. 2018], which checks the executability of the instruction and updates the node connectivity E^{t+1} as well as the state $S^{t+1} = \{s_i^{t+1}\}_{i=1}^{N_V}$ according to the execution of A^{t+1}. For example, in order to execute the instruction "sit chair", the current state of the human agent should be "standing" and the chair should not be occupied by other objects, *i.e.*, there is no connecting edge from other objects with type "on". If any of the constraints is not satisfied, the instruction will not be executed and M^t will simply be transferred to M^{t+1}. Otherwise, memory M^{t+1} will be updated based on the graph changes caused by the instruction execution.

Memory-based feature update. To update the node features in memory M^t, we consider three types of node feature updates, including action-driven, state-driven, and connectivity-driven updates, which refer to the feature updates caused by the action performance, node state changes, and node connectivity changes, respectively. The action-driven update concatenates the action embedding $\mathcal{E}_a(a^{t+1})$ with the node feature m_j^t, where j indicates the index of either the human agent or objects involved in the action, and then obtains the update message $q_{a,j}^{t+1}$ via an MLP network. Similarly, the state-driven update concatenates the state embedding $\mathcal{E}_s(s_j^{t+1})$

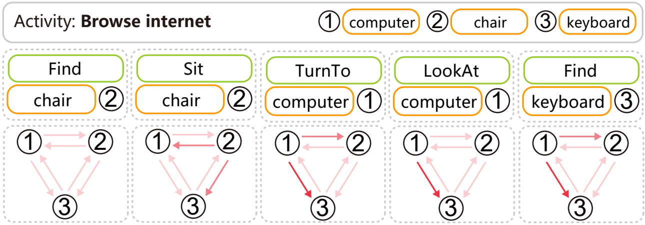


Fig. 7. Attention weights among neighboring objects in the *activity-aware feature propagation*. Darker colors indicate higher weights.

to the node feature m_j^t, where j indicates the index of instances whose states are updated due to the instruction execution, and then obtains the update message $q_{s,j}^{t+1}$ via an MLP network. For a new edge e_{ij}^{t+1}, the connectivity-driven update message is computed for both connecting instances, where we concatenate both the node features (m_i^t, m_j^t) and the relation embedding $\mathcal{E}_r(e_{ij}^{t+1})$ to predict the update message $q_{e,j}^{t+1}$ for node j via an MLP network. Note that one object may receive different types of update messages. We simply add these update messages to the corresponding node feature m_j^t to obtain the updated node state m_j^{t+1} in the final memory M^{t+1}.

Activity-aware feature propagation. With the updated memory M^{t+1}, we then propagate the features in the whole scene graph using a GNN to obtain the final node features Z^{t+1}. One thing to note here is that the scene connectivity is determined by the adjacency between objects and the information each edge encodes is only the spatial relationship, which makes the scene structure task-agnostic. However, the importance of each object to its adjacent objects will be different for different activities, and thus we need to make the feature propagation activity-aware. To achieve this goal, we develop an activity-aware heterogeneous graph attention network (aHGN) based on Simple-HGN [Lv et al. 2021], where the attention weight w_{ij} between two objects is conditioned on the global features F_g^t and F_l^t with the accumulated program, activity, and scene information.

We obtain the final node features Z^{t+1} for the next instruction generation by an $\mathbf{aHGN_1}$ with one layer : $Z^{t+1} = \mathbf{aHGN_1}(M^{t+1})$. Note that $Z^0 = M^0$ is obtained by feeding the initialized node features to another $\mathbf{aHGN_2}$ with two layers where the only global feature is the activity description $F_{\mathcal{D}}^t$. Details on the feature propagation in the aHGN are provided in the supplementary material. For each node j, the category embedding $\mathcal{E}_c(c_j)$ and state embedding $\mathcal{E}_s(s_j^0)$ are first concatenated and then passed to an MLP network to provide the initial node feature.

In Figure 7, for the activity "Browse Internet", we visualize the attention weights in five consecutive steps among the three neighboring objects "computer", "chair", and "keyboard". We find that the attention weight between the current object being interacted with and the next object to be interacted with is relatively higher, indicating that more information is transferred from the current object of interaction to the next object of interaction. This adaptive feature propagation updates the node features more accurately and benefits the selection of the objects of interaction.

4.4 Loss Function

Our network is trained end-to-end with the loss function defined for each iteration of instruction generation as follows:

$$L = \omega_a L_a + \omega_f L_f + \omega_c L_c + \omega_h L_h, \tag{1}$$

where L_a is the action loss, $L_f = L_f^1 + L_f^2$ is the fused instance loss, $L_c = L_c^1 + L_c^2$ is the category loss, and L_h is the human link loss. In our experiments, we set $\omega_a = \omega_f = 1$, $\omega_c = 0.1$, and $\omega_h = 0.01$. We use cross-entropy loss for all the four losses and compare the prediction of action probability P_a^t, fused instance probability P_f^t, category probability P_c^t, and human-centric probability P_h^t (as shown in Figure 4) to the GT distributions. More details on the loss function are provided in the supplementary material.

5 RESULTS AND EXPERIMENTS

5.1 Experiment setup

Dataset. We evaluate our method on the VirtualHome-Env dataset [Liao et al. 2019], which consists of around 30,000 programs, each paired with a corresponding scene. We use the same training/testing splits as Liao et al. [2019] in all our experiments.

Evaluation metrics. Following Liao et al. [2019], we evaluate the performance of program generation with well-established metrics that compare a result to the ground truth program associated with the given scene, including *LCS*, *Exectuability*, and *Completeness*.

LCS computes the normalized longest common subsequence between the generated and ground truth sequences of instructions, which is sensitive to the order of the sequences. *Executability* refers to the correctness in terms of execution, meaning that each instruction can be executed given the current scene state. *Completeness* measures if the generated programs complete the specific activities in the given scene, which is obtained by computing the difference between the final scene graph after the execution of the predicted program and the scene graph obtained by executing the ground truth program. The difference is measured with the F1 score. In particular, only the sub-graphs containing the object instances mentioned in the predicted program and the ground truth program are compared. Note that the *Completeness* is computed only if the generated program is executable, otherwise, it is set to zero. Thus, *Completeness* can also be seen as a metric considering both rationality and executability of the program in the given scene.

We also compute a *Rationality* metric to evaluate purely the semantic rationality of the program regarding the given task [Puig et al. 2018], by considering the semantic category of objects involved in the generated program instead of the object instances mapped in the given scene. Specifically, we first build a reference program by extracting the necessary steps from the human annotated activity description collected in [Puig et al. 2018], and then compute the normalized longest common subsequence between this reference program and the generated program at the semantic level.

5.2 Baseline methods and comparisons

As the two key contributions of our method include using language guidance to improve the *rationality* of the activity programs and

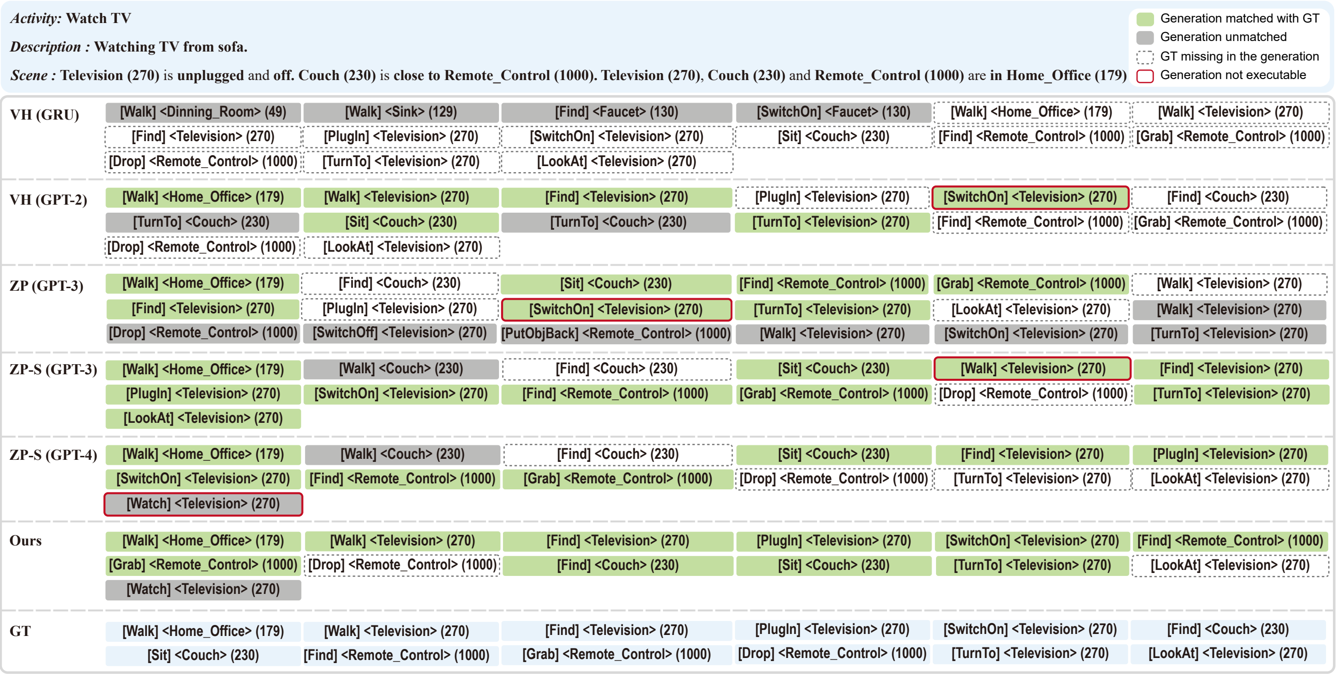

Activity: Watch TV
Description : Watching TV from sofa.
Scene : Television (270) is unplugged and off. Couch (230) is close to Remote_Control (1000). Television (270), Couch (230) and Remote_Control (1000) are in Home_Office (179)
Generation matched with GT
Generation unmatched
GT missing in the generation
Generation not executable
VH (GRU)
[Walk] <Dinning_Room> (49)
[Walk] <Sink> (129)
[Find] <Faucet> (130)
[SwitchOn] <Faucet> (130)
[Walk] <Home_Office> (179)
[Walk] <Television> (270)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[Sit] <Couch> (230)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Drop] <Remote_Control> (1000)
[TurnTo] <Television> (270)
[LookAt] <Television> (270)
VH (GPT-2)
[Walk] <Home_Office> (179)
[Walk] <Television> (270)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[Find] <Couch> (230)
[TurnTo] <Couch> (230)
[Sit] <Couch> (230)
[TurnTo] <Couch> (230)
[TurnTo] <Television> (270)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Drop] <Remote_Control> (1000)
[LookAt] <Television> (270)
ZP (GPT-3)
[Walk] <Home_Office> (179)
[Find] <Couch> (230)
[Sit] <Couch> (230)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Walk] <Television> (270)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[TurnTo] <Television> (270)
[LookAt] <Television> (270)
[Walk] <Television> (270)
[Drop] <Remote_Control> (1000)
[SwitchOff] <Television> (270)
[PutObjBack] <Remote_Control> (1000)
[Walk] <Television> (270)
[SwitchOn] <Television> (270)
[TurnTo] <Television> (270)
ZP-S (GPT-3)
[Walk] <Home_Office> (179)
[Walk] <Couch> (230)
[Find] <Couch> (230)
[Sit] <Couch> (230)
[Walk] <Television> (270)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Drop] <Remote_Control> (1000)
[TurnTo] <Television> (270)
[LookAt] <Television> (270)
ZP-S (GPT-4)
[Walk] <Home_Office> (179)
[Walk] <Couch> (230)
[Find] <Couch> (230)
[Sit] <Couch> (230)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Drop] <Remote_Control> (1000)
[TurnTo] <Television> (270)
[LookAt] <Television> (270)
[Watch] <Television> (270)
Ours
[Walk] <Home_Office> (179)
[Walk] <Television> (270)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Drop] <Remote_Control> (1000)
[Find] <Couch> (230)
[Sit] <Couch> (230)
[TurnTo] <Television> (270)
[LookAt] <Television> (270)
[Watch] <Television> (270)
GT
[Walk] <Home_Office> (179)
[Walk] <Television> (270)
[Find] <Television> (270)
[PlugIn] <Television> (270)
[SwitchOn] <Television> (270)
[Find] <Couch> (230)
[Sit] <Couch> (230)
[Find] <Remote_Control> (1000)
[Grab] <Remote_Control> (1000)
[Drop] <Remote_Control> (1000)
[TurnTo] <Television> (270)
[LookAt] <Television> (270)

Fig. 8. Qualitative comparison with rationality-oriented program generation methods.

encoding dynamic changes of the scene to improve the *executability* of the programs, we categorize related works into two types: *rationality-oriented approaches* that generate programs in a textual sequence-to-sequence manner to improve rationality but lack consideration of the target scene where the activity should be executed, and *executability-oriented approaches* that condition the activity program generation on the target scene to improve executability.

Comparison to rationality-oriented approaches. We first compare to a set of baselines that consider activity program generation purely as a natural language processing problem and generate the activity program using supervised learning or help from a pre-trained language model. Once the program is generated, we need to create a mapping from the object category in each instruction to an object instance inside the scene. We use the same simple mapping strategy as VirtualHome [Puig et al. 2018], which randomly maps an object category to the object instance in the scene with the same category. The program is not executable if there is no object instance that can be mapped to in the scene.

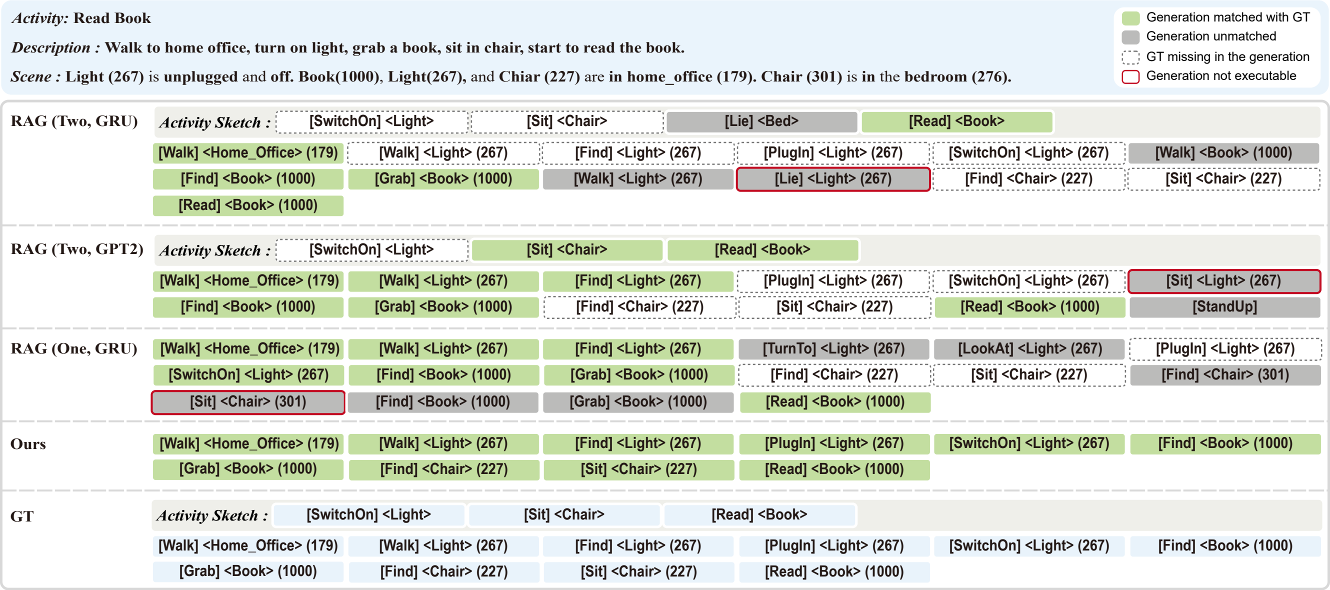
We compare to the following two baselines of this type:

- *VirtualHome (GRU)* [Puig et al. 2018], VH (GRU) for short, which encodes the input description with an RNN encoder and uses another RNN as the decoder to generate one instruction at a time. To adapt VH to our problem, we transform the scene graph into text and place it before the description in the input. Then, we use the GRU in our implementation and train this baseline with our dataset. To improve the rationality, we also enhance the method by replacing the GRU with GPT-2 [Radford et al. 2019] as in our method, and fine-tune on the dataset. This new variant is denoted as *VH (GPT-2)*.

Table 1. Quantitative comparisons with rationality-oriented approaches.

	Rationality	LCS	Executability	Completeness
VH (GRU)	0.123	0.096	0.332	0.020
VH (GPT-2)	0.334	0.324	0.385	0.257
ZP (GPT-3)	0.421	0.391	0.307	0.128
ZP-S (GPT-3)	0.435	0.419	0.334	0.226
ZP-S (GPT-4)	0.436	0.389	0.577	0.442
Ours	**0.438**	**0.515**	**0.746**	**0.584**

- *Zero-shot planner (GPT-3)* [Huang et al. 2022], ZP (GPT-3) for short, with the input adapted to our problem setting, which uses a pre-trained causal large language model (LLM) GPT-3 [Brown et al. 2020] to generate the program by taking an example program corresponding to a similar activity description as input. Note that the direct output of the LLM is a set of natural language sentences, which are then converted into instructions based on heuristics. This baseline adopts zero-shot learning and there is no supervision provided by our training set. To make this baseline more scene-aware, we also construct a new prompt with the specific scene information inspired by [Singh et al. 2022], and denote the enhanced variant as *ZP-S (GPT-3)*. The prompt construction details can be found in the supplementary material. Since the prompt with scene information is quite long, to be able to better utilize the contextual scene information for program generation, we test another baseline where we further enhance the scene-aware baseline with the latest released LLM GPT-4 [OpenAI 2023], denoted as *ZP-S (GPT-4)*.



Activity: Read Book

Description : Walk to home office, turn on light, grab a book, sit in chair, start to read the book.

Scene : Light (267) is unplugged and off. Book(1000), Light(267), and Chiar (227) are in home_office (179). Chair (301) is in the bedroom (276).

🟩 Generation matched with GT
▨ Generation unmatched
⬚ GT missing in the generation
🟥 Generation not executable

RAG (Two, GRU)
Activity Sketch : [SwitchOn] <Light> [Sit] <Chair> [Lie] <Bed> [Read] <Book>
[Walk] <Home_Office> (179) [Walk] <Light> (267) [Find] <Light> (267) [PlugIn] <Light> (267) [SwitchOn] <Light> (267) [Walk] <Book> (1000)
[Find] <Book> (1000) [Grab] <Book> (1000) [Walk] <Light> (267) [Lie] <Light> (267) [Find] <Chair> (227) [Sit] <Chair> (227)
[Read] <Book> (1000)

RAG (Two, GPT2)
Activity Sketch : [SwitchOn] <Light> [Sit] <Chair> [Read] <Book>
[Walk] <Home_Office> (179) [Walk] <Light> (267) [Find] <Light> (267) [PlugIn] <Light> (267) [SwitchOn] <Light> (267) [Sit] <Light> (267)
[Find] <Book> (1000) [Grab] <Book> (1000) [Find] <Chair> (227) [Sit] <Chair> (227) [Read] <Book> (1000) [StandUp]

RAG (One, GRU)
[Walk] <Home_Office> (179) [Walk] <Light> (267) [Find] <Light> (267) [TurnTo] <Light> (267) [LookAt] <Light> (267) [PlugIn] <Light> (267)
[SwitchOn] <Light> (267) [Find] <Book> (1000) [Grab] <Book> (1000) [Find] <Chair> (227) [Sit] <Chair> (227) [Find] <Chair> (301)
[Sit] <Chair> (301) [Find] <Book> (1000) [Grab] <Book> (1000) [Read] <Book> (1000)

Ours
[Walk] <Home_Office> (179) [Walk] <Light> (267) [Find] <Light> (267) [PlugIn] <Light> (267) [SwitchOn] <Light> (267) [Find] <Book> (1000)
[Grab] <Book> (1000) [Find] <Chair> (227) [Sit] <Chair> (227) [Read] <Book> (1000)

GT
Activity Sketch : [SwitchOn] <Light> [Sit] <Chair> [Read] <Book>
[Walk] <Home_Office> (179) [Walk] <Light> (267) [Find] <Light> (267) [PlugIn] <Light> (267) [SwitchOn] <Light> (267) [Find] <Book> (1000)
[Grab] <Book> (1000) [Find] <Chair> (227) [Sit] <Chair> (227) [Read] <Book> (1000)

Fig. 9. Qualitative comparison with executability-oriented program generation methods.

Table 1 shows the quantitative comparison results. We see that the two variants of the ZP baseline have high *Rationality*, indicating that they can generate rational programs that contain the necessary steps to complete the tasks at the semantic level. However, they cannot achieve good Executability and Completeness, mainly because these methods do not effectively utilize the scene information presented in the form of text. On the other hand, our method provides the highest Rationality, LCS, Executability, and Completeness among all the baselines, given its scene-aware program generation.

An example of a qualitative comparison is shown in Figure 8. For the results generated by each method, instructions matched with the GT are put in green boxes, while instructions in the GT but not predicted by the method are put in white boxes with dashed borders. All the remaining unmatched instructions are put in gray boxes. Moreover, a red border is added to each instruction that is not executable in the current scene, no matter whether it is matched with the GT or not, as the executability of an instruction also depends on the current states of the agent and the involved objects. For the activity “Watch TV”, an instruction “plug in television” is required to adapt to the scene where the television is unplugged. The VH (GRU) method generates many irrelevant instructions mainly due to overfitting. The VH (GPT-2) method alleviates the overfitting problem with a pre-trained model, but still does not generate the instruction for adapting to the scene. The ZP (GPT-3) method directly translates the description into a program and also does not generate the instruction “plug in television”. Moreover, ZP (GPT-3) duplicates several instructions at the end, as the language model has a preference to repeat the previous sentences [Xu et al. 2022]. While the ZP-S (GPT-3) and ZP-S (GPT-4) methods successfully predict the “plug in television” step by making use of the scene information provided in text form, they fail to generate an executable program. For ZP-S (GPT-3), an instruction “walk television” is predicted when the state of the human agent is “sitting” after the execution of the instruction “sit chair”, as the dynamic change of agent and scene states are not captured by the model. Similarly, for ZP-S (GPT-4), an instruction “turn to television” should be predicted first to make the human agent “face at” the television before the execution of the instruction “watch television”. Compared to ZP-S (GPT-3) and ZP-S (GPT-4), our method successfully generates an executable program to complete the required activity, with the final state of the scene after the program execution being the same as when executing the GT program. This example also shows that there are multiple ways to complete a given activity in a given scene, which explains why our method has high executability and completeness but relatively low LCS in this example.

Table 2. Quantitative comparisons with executability-oriented baselines, including three variants of RAG [Liao et al. 2019].

	Rationality	LCS	Executability	Completeness
RAG (Two, GRU)	0.305	0.377	0.529	0.350
RAG (Two, GPT-2)	0.327	0.343	0.472	0.322
RAG (One, GRU)	0.348	0.400	0.481	0.339
Ours	**0.438**	**0.515**	**0.746**	**0.584**

Comparison to executability-oriented approaches. We also compare to the other set of baselines that specifically take the scene into consideration and directly generate activity programs associated to scene instances. All the baselines are variants of the most related work [Liao et al. 2019], which we refer to as *RAG*. RAG [Liao et al. 2019] adopts a two-stage framework, which first transforms the input description into an activity sketch with a GRU, and then generates the complete program in the given scene with another

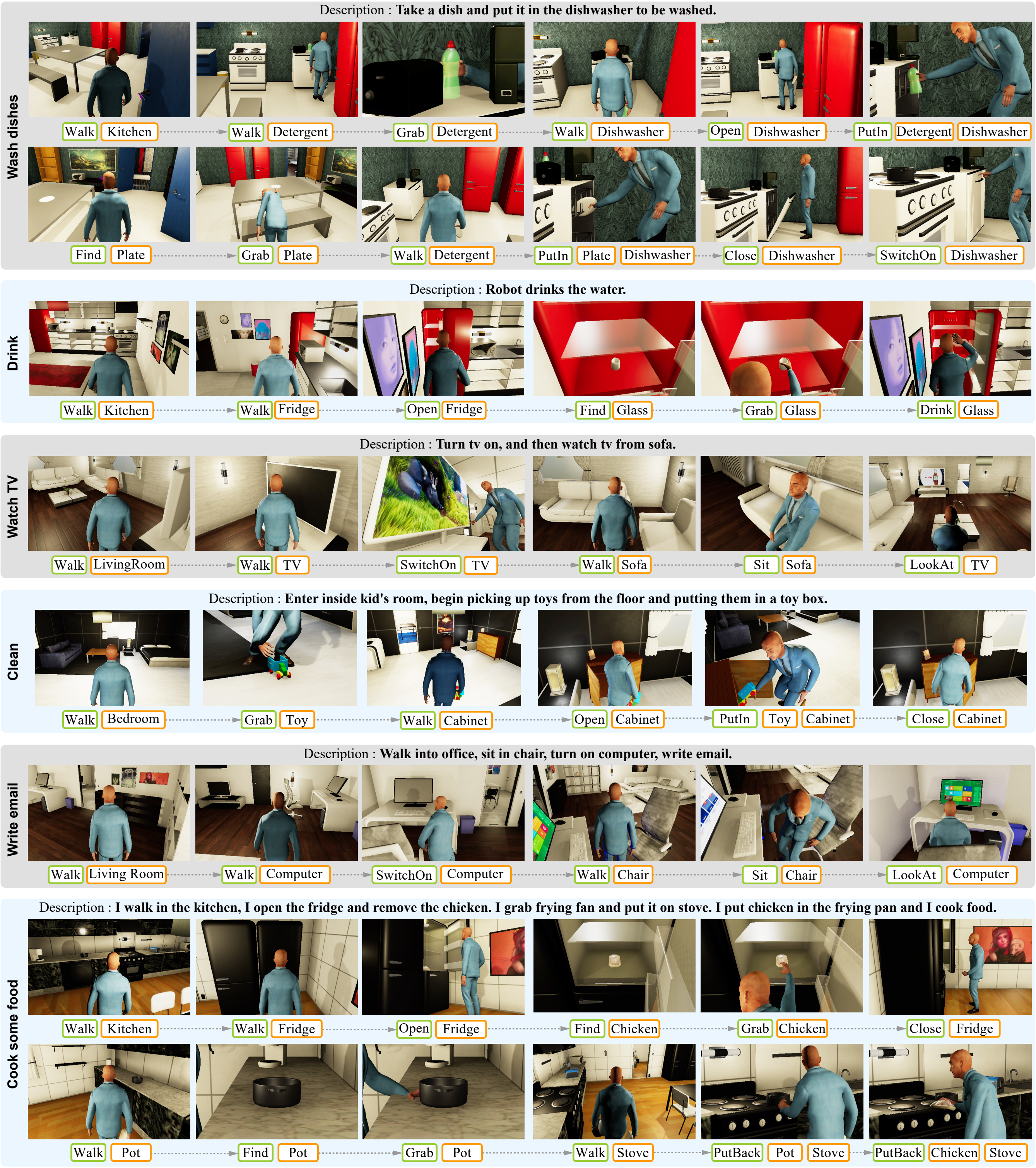


Fig. 10. Keyframes of several example animation sequences automatically generated from our results using the simulator of VirtualHome [Puig et al. 2018].

GRU based on the sketch. The key limitation of this method are the inconsistencies generated by the two-stage framework, where imperfect sketches are predicted at the first stage and then fed to the program generation stage trained on ground truth sketches, generating programs with errors. To alleviate this problem, we implement an improved version of this baseline by replacing the GRU network with GPT-2. We denote the two baselines as *RAG (Two, GRU)* and *RAG (Two, GPT-2)* according to the backbone used in the first stage, and implement another baseline that directly transforms the description into a program with a GRU without an activity sketch, and denote it as *RAG (One, GRU)*. Note that the setting with a one-stage framework based on GPT-2 is similar to our framework, and thus we also include it in the more detailed ablation study on language guidance in the supplementary material.

Table 2 shows the quantitative comparison results. We can see that these methods have relatively low Rationality, and our method again performs consistently better than the three baselines in terms of all metrics. The inconsistency problem is alleviated by the improved *RAG (Two, GPT-2)* and *RAG (One, GRU)*. Thus, these baselines perform better than the original *RAG (Two, GRU)* baseline.

One example of a qualitative comparison is shown in Figure 9. An action "sit chair" should be predicted to complete the task expressed by the activity description, and the "chair" object specified for sitting should be the one in the home office. Due to the limited ability of the GRU, the *RAG (Two, GRU)* baseline fails to predict the sitting step in the sketch, and thus does not add it to the complete program. By introducing a pre-trained language model, the *RAG (Two, GPT-2)* baseline successfully predicts this step in the sketch, but still fails to add it to the complete program. Although the *RAG (One, GRU)* baseline successfully predicts the step of sitting on a chair, it selects an inaccessible chair in the bedroom to sit on, as the human agent has entered the home office in the first instruction. Since we generate the program with language guidance and adjacency constraints of the human agent, our method successfully predicts the "sit chair" instruction in terms of semantics, and chooses the right chair instance inside the home office to sit on.

5.3 Activity demonstration

Once the complete activity program is constructed, we can further generate an animation of the human agent executing the activity program in the given scene with the simulator provided by VirtualHome [Puig et al. 2018]. Specifically, the animations of the 12 most frequent atomic actions, such as walk, grab, and open, are predefined in the simulator, and Unity's NavMesh framework is used for path planning and navigation. Figure 10 shows some keyframes of several automatically-generated animation sequences. It is interesting to note how the activity is decomposed into actions of the human agent that are executable in the given scene, although there are some visual artifacts in the pre-defined atomic actions. More demonstrations can be found in the supplementary video.

5.4 Ablation Study

To examine the effectiveness of our design, we conduct a detailed ablation study on the key components of our method, including using category information and a pre-trained language model (PLM) for language guidance (LG), considering the human agent's position for adjacency constraints (AC), and adopting the memory-driven dynamic graph update (DU). The baseline model only utilizes the instance-based graph feature F_t^g to predict the graph-guided probability P_t^g, without any help from our key components. The results are summarized in Table 3.

Table 3. Ablation study on key components of our method.

AC	DU	LG	Rationality	LCS	Executability	Completeness
			0.348	0.400	0.481	0.339
		✓	0.391	0.451	0.600	0.455
	✓		0.360	0.425	0.691	0.498
✓			0.346	0.423	0.604	0.423
✓	✓		0.352	0.422	0.680	0.505
✓	✓	✓	**0.438**	**0.515**	**0.746**	**0.584**

We can see that all the key components solely improve the performance, and the performance consistently increases by stacking the key components. We find that LG mainly improves the Rationality and LCS metrics, while DU and AC mainly improve the Executability and thus Completeness metrics.

We show qualitative ablation results in Figure 11. The first, second, and third rows are the qualitative ablation results obtained when adding the different key components to our method, i.e., AC, LG, and DU, respectively. The first row shows an example of qualitative ablation on the adjacency constraint (AC). As described in the activity description, the target program is related to the object "sofa" in the living room. Due to the lack of consideration of adjacency of the human agent node, the baseline predicts the instructions that involve an irrelevant and inaccessible object "book" in the bedroom, as the human agent has entered the home office in the first instruction, while our method focuses on the relevant objects in the home office when using the adjacency constraint. The second row shows an example of qualitative ablation on language guidance (LG). As described in the activity description, instructions related to the object "keyboard" should be predicted in the program to complete the task described in the description. Due to the absence of semantic information, the baseline fails to predict these instructions in the program, while our method with LG successfully predicts these steps and generates a more rational program with the help of language guidance. The third row shows an example of qualitative ablation on the dynamic graph update (DU), where the ground truth program of the activity is composed of a series of instructions involving three nearby objects. The baseline predicts many unmatched instructions with irrelevant objects. In contrast, the method with DU is able to generate a more rational program, since the activity-aware spatial relationship among the nodes is explicitly captured in the dynamic graph update. We also present a more detailed analysis on the impact of each module on executability in the supplementary material.

5.5 Generality to different inputs

Generality to different activity specification formats. The activity description in the dataset is a scene-associated description that is implicitly conditioned on the environment. To validate the robustness

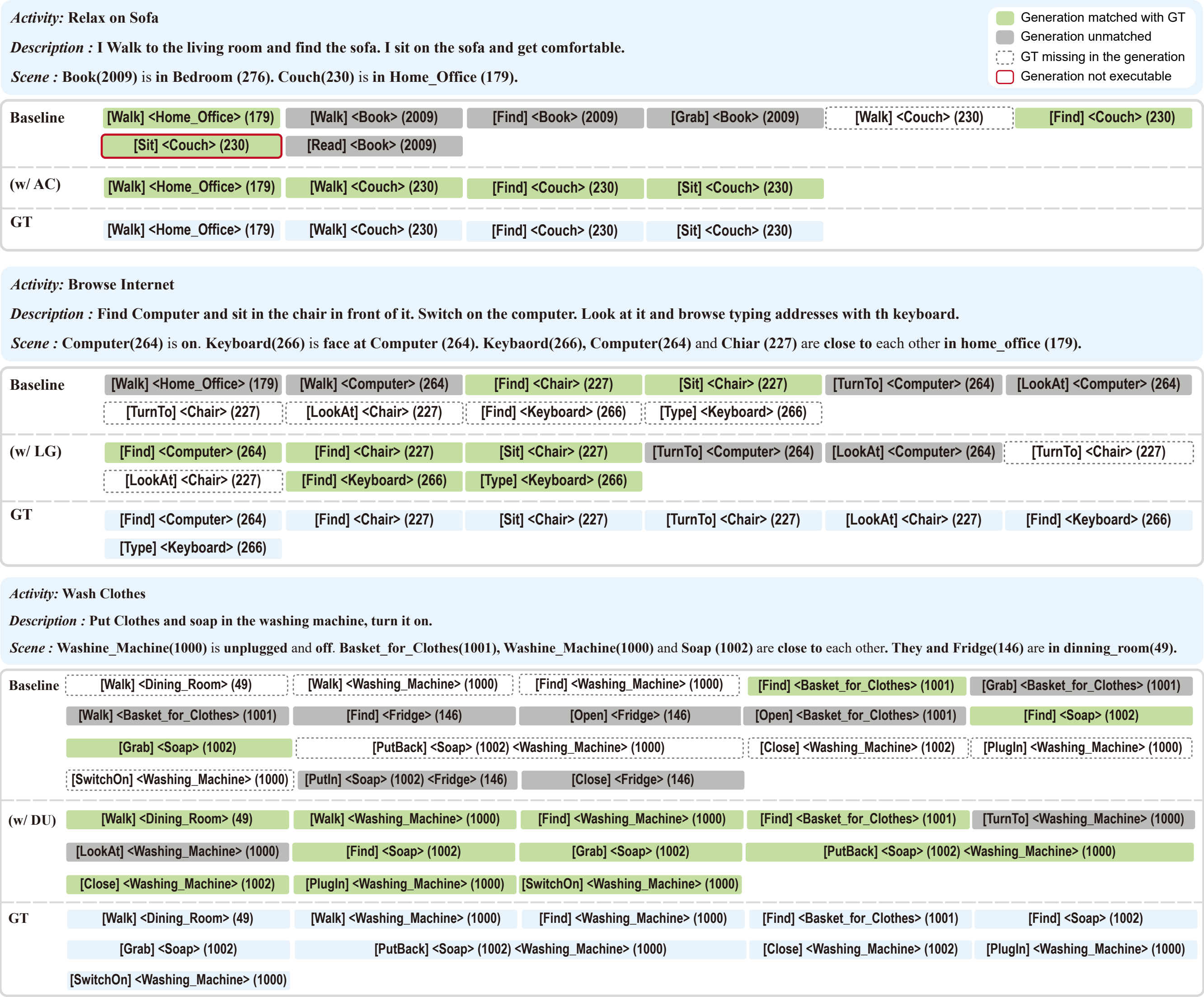

Activity: Relax on Sofa
Description : I Walk to the living room and find the sofa. I sit on the sofa and get comfortable.
Scene : Book(2009) is in Bedroom (276). Couch(230) is in Home_Office (179).
Generation matched with GT
Generation unmatched
GT missing in the generation
Generation not executable
Baseline
[Walk] <Home_Office> (179)
[Walk] <Book> (2009)
[Find] <Book> (2009)
[Grab] <Book> (2009)
[Walk] <Couch> (230)
[Find] <Couch> (230)
[Sit] <Couch> (230)
[Read] <Book> (2009)
(w/ AC)
[Walk] <Home_Office> (179)
[Walk] <Couch> (230)
[Find] <Couch> (230)
[Sit] <Couch> (230)
GT
[Walk] <Home_Office> (179)
[Walk] <Couch> (230)
[Find] <Couch> (230)
[Sit] <Couch> (230)
Activity: Browse Internet
Description : Find Computer and sit in the chair in front of it. Switch on the computer. Look at it and browse typing addresses with th keyboard.
Scene : Computer(264) is on. Keyboard(266) is face at Computer (264). Keybaord(266), Computer(264) and Chiar (227) are close to each other in home_office (179).
Baseline
[Walk] <Home_Office> (179)
[Walk] <Computer> (264)
[Find] <Chair> (227)
[Sit] <Chair> (227)
[TurnTo] <Computer> (264)
[LookAt] <Computer> (264)
[TurnTo] <Chair> (227)
[LookAt] <Chair> (227)
[Find] <Keyboard> (266)
[Type] <Keyboard> (266)
(w/ LG)
[Find] <Computer> (264)
[Find] <Chair> (227)
[Sit] <Chair> (227)
[TurnTo] <Computer> (264)
[LookAt] <Computer> (264)
[TurnTo] <Chair> (227)
[LookAt] <Chair> (227)
[Find] <Keyboard> (266)
[Type] <Keyboard> (266)
GT
[Find] <Computer> (264)
[Find] <Chair> (227)
[Sit] <Chair> (227)
[TurnTo] <Chair> (227)
[LookAt] <Chair> (227)
[Find] <Keyboard> (266)
[Type] <Keyboard> (266)
Activity: Wash Clothes
Description : Put Clothes and soap in the washing machine, turn it on.
Scene : Washine_Machine(1000) is unplugged and off. Basket_for_Clothes(1001), Washine_Machine(1000) and Soap (1002) are close to each other. They and Fridge(146) are in dinning_room(49).
Baseline
[Walk] <Dining_Room> (49)
[Walk] <Washing_Machine> (1000)
[Find] <Washing_Machine> (1000)
[Find] <Basket_for_Clothes> (1001)
[Grab] <Basket_for_Clothes> (1001)
[Walk] <Basket_for_Clothes> (1001)
[Find] <Fridge> (146)
[Open] <Fridge> (146)
[Open] <Basket_for_Clothes> (1001)
[Find] <Soap> (1002)
[Grab] <Soap> (1002)
[PutBack] <Soap> (1002) <Washing_Machine> (1000)
[Close] <Washing_Machine> (1002)
[PlugIn] <Washing_Machine> (1000)
[SwitchOn] <Washing_Machine> (1000)
[PutIn] <Soap> (1002) <Fridge> (146)
[Close] <Fridge> (146)
(w/ DU)
[Walk] <Dining_Room> (49)
[Walk] <Washing_Machine> (1000)
[Find] <Washing_Machine> (1000)
[Find] <Basket_for_Clothes> (1001)
[TurnTo] <Washing_Machine> (1000)
[LookAt] <Washing_Machine> (1000)
[Find] <Soap> (1002)
[Grab] <Soap> (1002)
[PutBack] <Soap> (1002) <Washing_Machine> (1000)
[Close] <Washing_Machine> (1002)
[PlugIn] <Washing_Machine> (1000)
[SwitchOn] <Washing_Machine> (1000)
GT
[Walk] <Dining_Room> (49)
[Walk] <Washing_Machine> (1000)
[Find] <Washing_Machine> (1000)
[Find] <Basket_for_Clothes> (1001)
[Find] <Soap> (1002)
[Grab] <Soap> (1002)
[PutBack] <Soap> (1002) <Washing_Machine> (1000)
[Close] <Washing_Machine> (1002)
[PlugIn] <Washing_Machine> (1000)
[SwitchOn] <Washing_Machine> (1000)

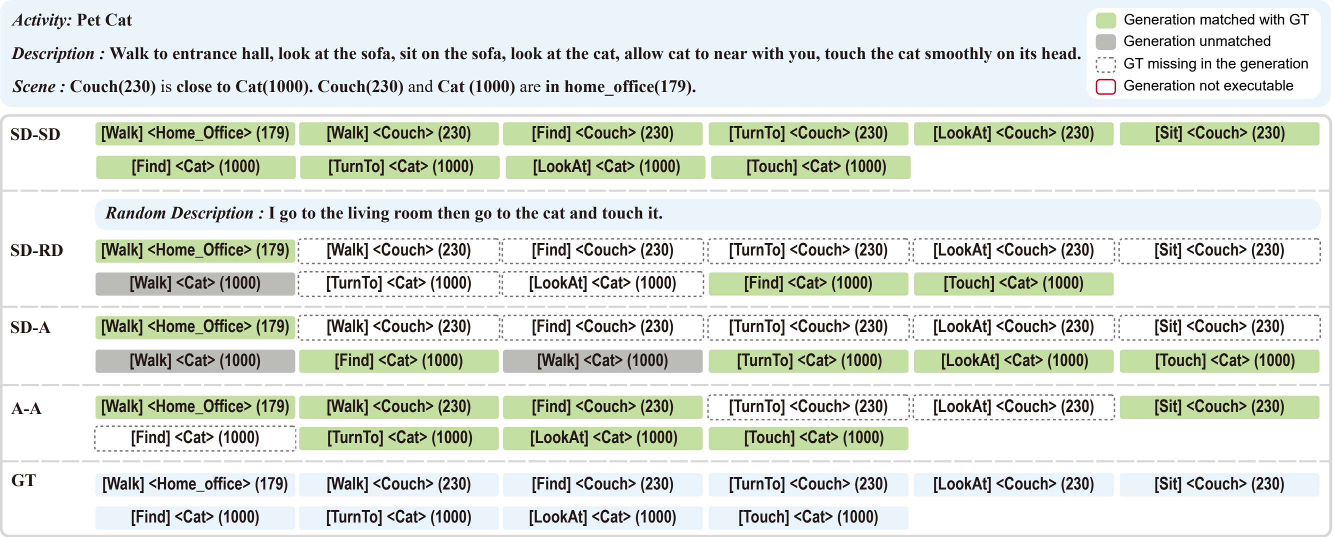
Fig. 11. Qualitative results of ablation on key components of our method: adjacency constraint (AG), language guidance (LG), and dynamic graph update (DU).

Table 4. Generality to different activity specification formats.

	Rationality	LCS	Executability	Completeness
SD-SD	**0.438**	**0.515**	**0.746**	**0.584**
SD-RD	0.359	0.404	0.668	0.459
SD-A	0.387	0.483	0.699	0.524
A-A	0.418	0.484	0.725	0.550

of our method to the format used for specifying the input activity, we test our method with three different representations of the activity specification: the scene-associated description (SD), a random description (RD), and an activity label (A). SD refers to the description associated with the given input scene in the dataset, which we used in the previous experiments, RD refers to a description with the same activity label as the scene description but associated to another randomly-sampled scene in the dataset, and A refers to a high-level activity label like “Watch TV”.

Table 4 shows the results of our method trained and then tested on different specification formats. On the left column, the acronym before the hyphen indicates the format of the training data, while the acronym after the hyphen indicates the format of the testing data. We can see that our method trained on the scene-associated description (SD) is robust even when the input description of the test data is replaced by a short activity label (SD-A). The performance is also close to the method retrained on the dataset with activity labels as input (A-A). These results demonstrate that even though there is no scene-associated information specified by the



Activity: Pet Cat

Description : Walk to entrance hall, look at the sofa, sit on the sofa, look at the cat, allow cat to near with you, touch the cat smoothly on its head.

Scene : Couch(230) is close to Cat(1000). Couch(230) and Cat (1000) are in home_office(179).

🟩 Generation matched with GT
▨ Generation unmatched
⬚ GT missing in the generation
🟥 Generation not executable

SD-SD
🟩 [Walk] <Home_Office> (179) 🟩 [Walk] <Couch> (230) 🟩 [Find] <Couch> (230) 🟩 [TurnTo] <Couch> (230) 🟩 [LookAt] <Couch> (230) 🟩 [Sit] <Couch> (230)
🟩 [Find] <Cat> (1000) 🟩 [TurnTo] <Cat> (1000) 🟩 [LookAt] <Cat> (1000) 🟩 [Touch] <Cat> (1000)

Random Description : I go to the living room then go to the cat and touch it.

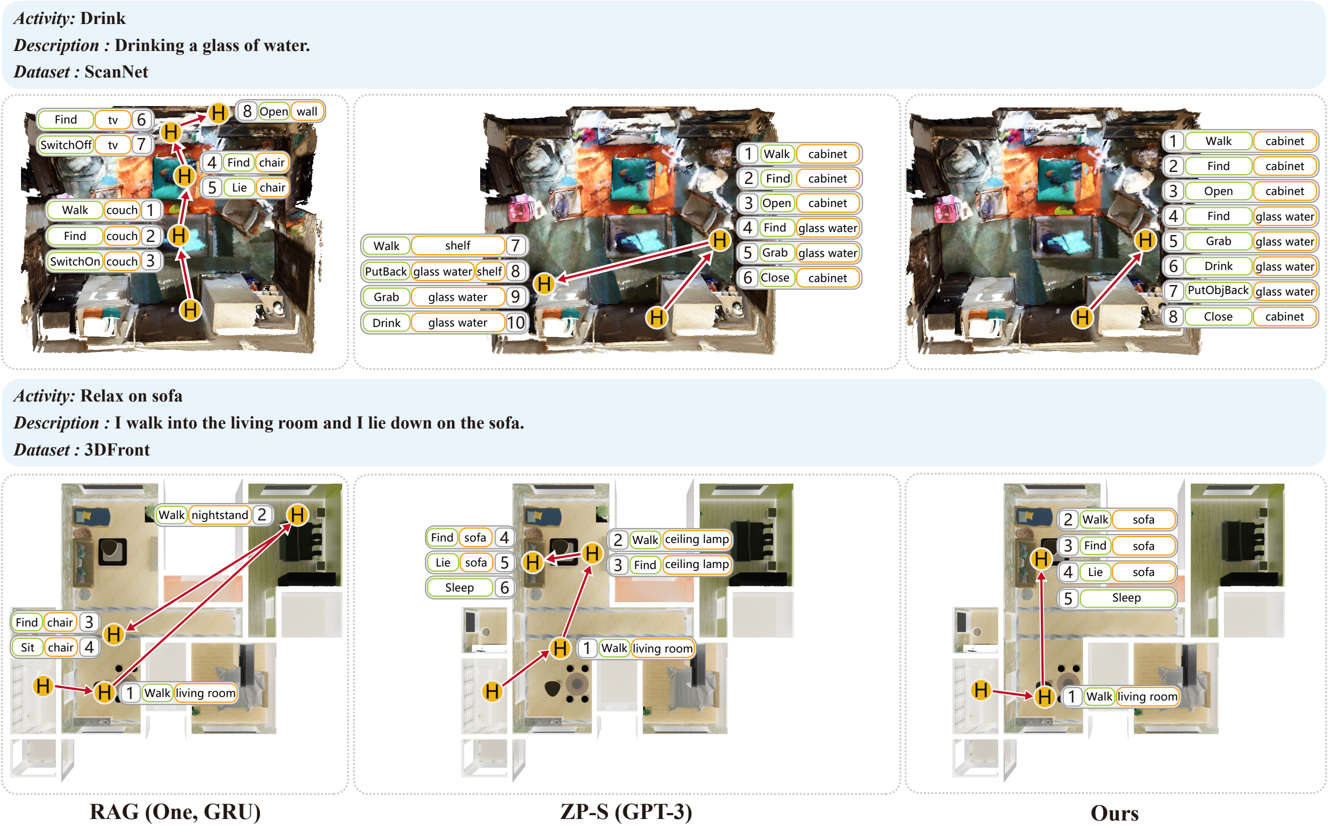
SD-RD
🟩 [Walk] <Home_Office> (179) ⬚ [Walk] <Couch> (230) ⬚ [Find] <Couch> (230) ⬚ [TurnTo] <Couch> (230) ⬚ [LookAt] <Couch> (230) ⬚ [Sit] <Couch> (230)
▨ [Walk] <Cat> (1000) ⬚ [TurnTo] <Cat> (1000) ⬚ [LookAt] <Cat> (1000) 🟩 [Find] <Cat> (1000) 🟩 [Touch] <Cat> (1000)

SD-A
🟩 [Walk] <Home_Office> (179) ⬚ [Walk] <Couch> (230) ⬚ [Find] <Couch> (230) ⬚ [TurnTo] <Couch> (230) ⬚ [LookAt] <Couch> (230) ⬚ [Sit] <Couch> (230)
▨ [Walk] <Cat> (1000) 🟩 [Find] <Cat> (1000) ▨ [Walk] <Cat> (1000) 🟩 [TurnTo] <Cat> (1000) 🟩 [LookAt] <Cat> (1000) 🟩 [Touch] <Cat> (1000)

A-A
🟩 [Walk] <Home_Office> (179) 🟩 [Walk] <Couch> (230) 🟩 [Find] <Couch> (230) ⬚ [TurnTo] <Couch> (230) ⬚ [LookAt] <Couch> (230) 🟩 [Sit] <Couch> (230)
⬚ [Find] <Cat> (1000) 🟩 [TurnTo] <Cat> (1000) 🟩 [LookAt] <Cat> (1000) 🟩 [Touch] <Cat> (1000)

GT
[Walk] <Home_office> (179) [Walk] <Couch> (230) [Find] <Couch> (230) [TurnTo] <Couch> (230) [LookAt] <Couch> (230) [Sit] <Couch> (230)
[Find] <Cat> (1000) [TurnTo] <Cat> (1000) [LookAt] <Cat> (1000) [Touch] <Cat> (1000)

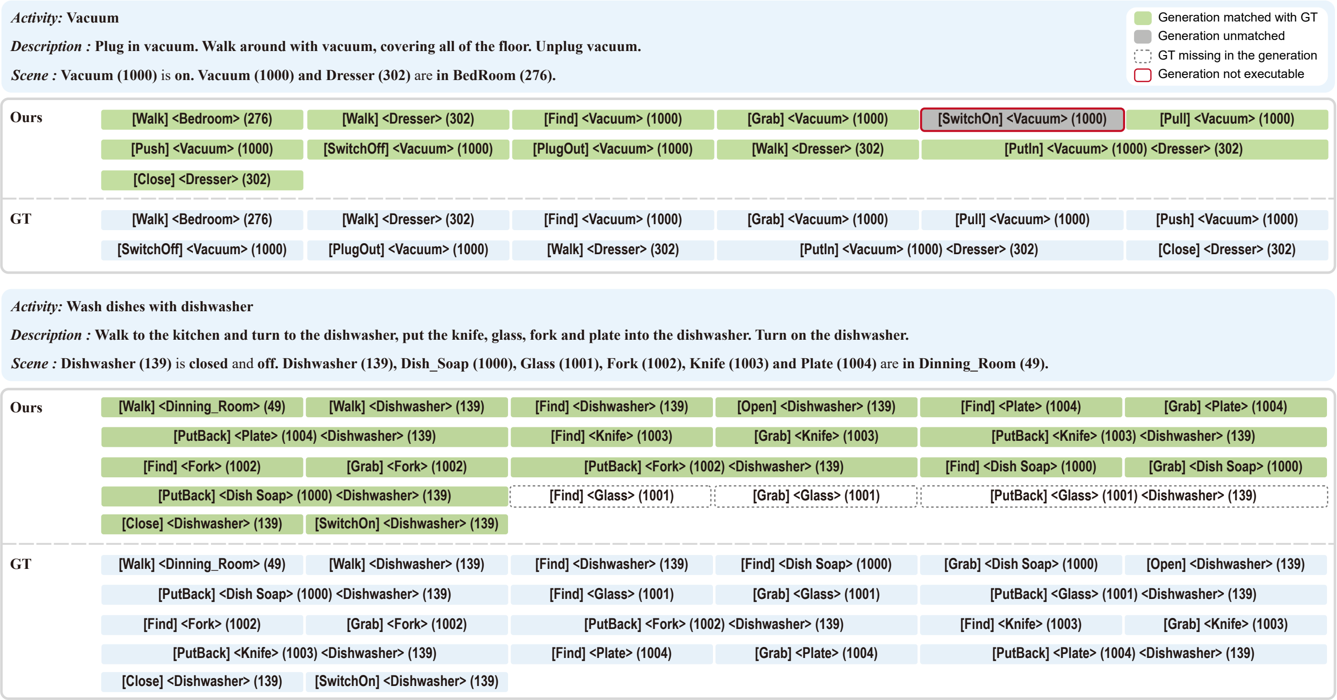
Fig. 12. Qualitative results showing the generality of our method to different activity specification formats.



Activity: Drink
Description : Drinking a glass of water.
Dataset : ScanNet

Activity: Relax on sofa
Description : I walk into the living room and I lie down on the sofa.
Dataset : 3DFront

Fig. 13. Qualitative examples showing the generality of our method to scene inputs from different datasets. The sequences of arrows laid over the top views of the scenes show the steps of the activity programs and the locations in the scenes where the activities should take place.



Activity: Vacuum

Description : Plug in vacuum. Walk around with vacuum, covering all of the floor. Unplug vacuum.

Scene : Vacuum (1000) is on. Vacuum (1000) and Dresser (302) are in BedRoom (276).

Activity: Wash dishes with dishwasher

Description : Walk to the kitchen and turn to the dishwasher, put the knife, glass, fork and plate into the dishwasher. Turn on the dishwasher.

Scene : Dishwasher (139) is closed and off. Dishwasher (139), Dish_Soap (1000), Glass (1001), Fork (1002), Knife (1003) and Plate (1004) are in Dinning_Room (49).

Ours
[Walk] <Dinning_Room> (49)
[Walk] <Dishwasher> (139)
[Find] <Dishwasher> (139)
[Open] <Dishwasher> (139)
[Find] <Plate> (1004)
[Grab] <Plate> (1004)
[PutBack] <Plate> (1004) <Dishwasher> (139)
[Find] <Knife> (1003)
[Grab] <Knife> (1003)
[PutBack] <Knife> (1003) <Dishwasher> (139)
[Find] <Fork> (1002)
[Grab] <Fork> (1002)
[PutBack] <Fork> (1002) <Dishwasher> (139)
[Find] <Dish Soap> (1000)
[Grab] <Dish Soap> (1000)
[PutBack] <Dish Soap> (1000) <Dishwasher> (139)
[Find] <Glass> (1001)
[Grab] <Glass> (1001)
[PutBack] <Glass> (1001) <Dishwasher> (139)
[Close] <Dishwasher> (139)
[SwitchOn] <Dishwasher> (139)
GT
[Walk] <Dinning_Room> (49)
[Walk] <Dishwasher> (139)
[Find] <Dishwasher> (139)
[Find] <Dish Soap> (1000)
[Grab] <Dish Soap> (1000)
[Open] <Dishwasher> (139)
[PutBack] <Dish Soap> (1000) <Dishwasher> (139)
[Find] <Glass> (1001)
[Grab] <Glass> (1001)
[PutBack] <Glass> (1001) <Dishwasher> (139)
[Find] <Fork> (1002)
[Grab] <Fork> (1002)
[PutBack] <Fork> (1002) <Dishwasher> (139)
[Find] <Knife> (1003)
[Grab] <Knife> (1003)
[PutBack] <Knife> (1003) <Dishwasher> (139)
[Find] <Plate> (1004)
[Grab] <Plate> (1004)
[PutBack] <Plate> (1004) <Dishwasher> (139)
[Close] <Dishwasher> (139)
[SwitchOn] <Dishwasher> (139)

Fig. 14. Failure cases of our method. Top row: the state of the vacuum is not taken into account. Bottom row: not all objects are added to the dishwasher.

activity description, our model is still able to adapt to a new scene and generate a program based only on an activity label. The testing results on the random description (RD) are the worst, since the random description provides a specific way of performing the required activity which may contain steps inconsistent with the given scene.

One example of a qualitative comparison is shown in Figure 12. Instructions related to the object "couch" should be predicted so that the task can be completed. Since the random description does not mention "couch", our model with a random description as input fails to predict the couch-related instruction steps. Similarly, our model with an activity label also fails to predict these steps, but successfully predicts other correct steps based only on the activity label. Our model retrained on the dataset with activity label as input can predict most of the steps, which is close to the performance of the model with a scene-associated description as input.

Generality to different scene inputs. To evaluate the generality of our approach to different scene datasets, we tested our method on the ScanNet [Dai et al. 2017] and 3DFront [Fu et al. 2021] datasets. To be able to test our method on these datasets, we constructed scene graphs for the scenes in these datasets in the VirtualHome format. Specifically, since our method is trained on fixed object categories from VirtualHome, which cover most of the household object categories, we first map the objects of other datasets to VirtualHome's nodes according to the semantic similarity of the object categories. Then, we compute the spatial relationships between objects based on their bounding boxes. Qualitative results are shown in Figure 13.

Since the scenes from these two datasets are relatively different from those of VirtualHome, the purely graph-based approach *RAG (One, GRU)* exhibits poor generalization and fails to generate reasonable programs even for simple activities. ZP-S (GPT-3) based on a large language model cannot properly consider the scenes and thus tends to generate irrelevant intermediate steps in the programs. Our approach can accurately generate programs for these new scenes, exhibiting better generalizability.

6 CONCLUSION, LIMITATIONS, AND FUTURE WORK

In this work, we propose a novel algorithm to achieve automatic scene-aware activity program generation. Our algorithm outperforms the past approaches in both rationality and executability, thanks to the design of the method which extracts semantic information from a pre-trained language model and deploys an explicit perception of the target scene via a scene graph.

Since our method is trained end-to-end with data entries in the form <activity, program>, it works well on the seen activities, and also generalizes well to test cases that include the same activities with different descriptions, which correspond to the test set in the VirtualHome-Env dataset. However, our method is not guaranteed to generate rational programs for activities never seen before. Although the language model has good generalization capability to unseen text, it is deployed in our method more as a feature extractor which does not directly determine the generated instructions.

Moreover, Figure 14 shows example results that represent the main failure modes of our method, where our method either generates redundant instructions that are not matched to an object state or misses some key instructions when too many objects are involved in the activity.

For the activity "Vacuum" shown in the first row of the figure, the vacuum is given in "on" state and thus does not need to be "switched on" in the GT program. However, since we use all kinds of global and local information to guide the program generation and most of the GT programs in the dataset contain the "switch on the vacuum" step, our method sometimes fails to generate the appropriate program relative to the current object state and generates the redundant "switch on" step in this case, which leads to the the program not being executable. It would be interesting to explore ways to make the program generation process give more importance to the object state to further improve the executability of the programs.

For the activity "Wash dishes with dishwasher" shown in the second row, as many objects need to be grabbed and put into the dishwasher, our method successfully completes the task for all the objects other than the "Glass". One future direction is to find a better encoding of the activity description to ensure that our method generates programs that cover all the objects that are mentioned in the description or should be involved when completing an activity.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. This work was supported in parts by NSFC (62322207, U21B2023, U2001206), GD Natural Science Foundation (2021B1515020085), Shenzhen Science and Technology Program (RCYX20210609103121030), Tencent AI Lab Rhino-Bird Focused Research Program (RBFR2022013), NSERC Canada through a Discovery Grant and Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ).

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691* (2022).

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics* 1 (2013), 49–62.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5828–5839.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, et al. 2022. Procthor: Large-scale embodied ai using procedural generation. *arXiv preprint arXiv:2206.06994* (2022).

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. 2018. Speaker-follower models for vision-and-language navigation. *Advances in Neural Information Processing Systems* 31 (2018).

Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 2021. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10933–10942.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207* (2022).

Peter A Jansen. 2020. Visually-grounded planning without vision: Language models infer detailed plans from high-level instructions. *arXiv preprint arXiv:2009.14259* (2020).

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474* (2017).

Jie Lei, Linjie Li, Luowei Zhou, Zhe Gan, Tamara L Berg, Mohit Bansal, and Jingjing Liu. 2021. Less is more: Clipbert for video-and-language learning via sparse sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7331–7341.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. 2023. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*. PMLR, 80–93.

Yuan-Hong Liao, Xavier Puig, Marko Boben, Antonio Torralba, and Sanja Fidler. 2019. Synthesizing environment-aware activities via activity sketches. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6291–6299.

Yujie Lu, Weixi Feng, Wanrong Zhu, Wenda Xu, Xin Eric Wang, Miguel Eckstein, and William Yang Wang. 2022. Neuro-Symbolic Causal Language Planning with Commonsense Prompting. *arXiv preprint arXiv:2206.02928* (2022).

Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S Yu, and Yanfang Ye. 2019. Temporal network embedding with micro-and macro-dynamics. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 469–478.

Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1150–1160.

Corey Lynch and Pierre Sermanet. 2020a. Grounding language in play. (2020).

Corey Lynch and Pierre Sermanet. 2020b. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648* (2020).

Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. 2020. Improving vision-and-language navigation with image-text pairs from the web. In *European Conference on Computer Vision*. Springer, 259–274.

Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition* 97 (2020), 107000.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

Dipendra Misra, Kejia Tao, Percy Liang, and Ashutosh Saxena. 2015. Environment-driven lexicon induction for high-level instructions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 992–1002.

Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. 2016. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research* 35, 1-3 (2016), 281–300.

Daniel Nyga and Michael Beetz. 2012. Everything robots always wanted to know about housework (but were afraid to ask). In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 243–250.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8494–8502.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

Santhosh K Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, et al. 2021. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *arXiv preprint arXiv:2109.08238* (2021).

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637* (2020).

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*. 519–527.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302* (2022).

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Thirty-first AAAI conference on artificial* *intelligence*.

Moritz Tenorth, Daniel Nyga, and Michael Beetz. 2010. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *2010 ieee international conference on robotics and automation*. IEEE, 1486–1491.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*.

Shreshth Tuli, Rajas Bansal, Rohan Paul, et al. 2021. Tango: commonsense generalization in predicting tool interactions for mobile manipulators. *arXiv preprint arXiv:2105.04556* (2021).

Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 2019. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6629–6638.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652* (2021).

Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. 2022. Learning to Break the Loop: Analyzing and Mitigating Repetitions for Neural Text Generation. *arXiv preprint arXiv:2206.02369* (2022).

Yezhou Yang, Anupam Guha, Cornelia Fermüller, and Yiannis Aloimonos. 2014. Manipulation action tree bank: A knowledge resource for humanoids. In *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 987–992.

Yezhou Yang, Yi Li, Cornelia Fermuller, and Yiannis Aloimonos. 2015. Robot learning manipulation action plans by" watching" unconstrained videos from the world wide web. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

Shuquan Ye, Dongdong Chen, Songfang Han, and Jing Liao. 2022. 3D question answering. *IEEE Transactions on Visualization and Computer Graphics* (2022).

Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhao Li, and Can Wang. 2020. Learning temporal interaction graph embedding via coupled memory networks. In *Proceedings of the web conference 2020*. 3049–3055.