VirtualHome: Building Socially Intelligent Agents via Simulation

by

Xavier Puig Fernández

Grau en Enginerya Informàtica (2016)
Grau en Enginyeria de Telecomunicacions (2016)
SM, Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2023

Author .....
Department of Electrical Engineering and Computer Science
October 14, 2022

Certified by .....
Antonio Torralba
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

VirtualHome: Building Socially Intelligent Agents via Simulation

by

Xavier Puig Fernández

Submitted to the Department of Electrical Engineering and Computer Science
on October 14, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

While remarkable progress has been made in building autonomous agents that can help us at complex tasks, these have typically been studied in isolated environments. To build agents that can be deployed in the real world, we need to study them in more human-centric settings, where they can interact with other humans and agents. Moreover, for agents to assist us effectively, they need to be able to operate in these environments while understanding our intentions and beliefs, and learn to coordinate with us effectively and safely.

This thesis investigates the development of such assistive agents through simulation environments. In the first part of the thesis, we introduce VirtualHome, a multi-agent platform for simulating human activities in household environments, and introduce a knowledge base of daily human activities that can be executed in the simulator. Then, we present agents that can perform different tasks in the environment given human descriptions or demonstrations of the activity. Finally, we study agents that can perform activities together with other humans in the environment. We propose a framework to simulate humans in the environment at scale and propose two challenges, Watch-And-Help and Online-Watch-And-Help, to benchmark the performance of different assistive agents, and test their effectiveness in assisting real humans performing activities in VirtualHome. Together, the methods and tools presented in this thesis provide a way to study assistive agents in simulation, allowing to develop these agents safely and at scale before deploying them into the real world.

Thesis Supervisor: Antonio Torralba
Title: Professor of Electrical Engineering and Computer Science

Acknowledgements

This thesis marks the end of my seven-year journey at MIT. This has been an incredible time, which has given me the chance to grow professionally and personally. I owe this to several people without whom this thesis would not be possible.

First of all, I want to thank my advisor, Antonio Torralba. I met Antonio during my sophomore year in Spain, and he gave me a chance to collaborate with him remotely throughout college, despite having no prior research experience. A few years later, I had the opportunity to do my bachelor thesis in his lab, spend time doing research in Toronto and later, start a PhD in his group. All these experiences made me realize I wanted to pursue a research career in Artificial Intelligence, have allowed me to meet incredible people, and have shaped the last years of my life. Antonio has also profoundly impacted the way I think about research and the kind of work I want to pursue. I want to thank him for giving me the freedom to pursue my interests, for teaching me to focus on the big problems, even when they may be ill-defined, far from being solved, or not yet fully appreciated; to focus on the novel exciting directions, even if these may take longer to reach.

I also want to thank Sanja Fidler, with whom I have had the privilege of working closely throughout my PhD. Sanja welcomed me into her lab before my PhD, giving me a taste of the research I would want to pursue in the following years, and her guidance has helped me become a better scientist and professional. I am also profoundly thankful for her advice, beyond research, throughout the years. Sanja has taught me to identify my strengths and focus on them to provide value to bigger efforts while at the same time learning and expanding my skills.

Furthermore, I would like to thank Phillip Isola, for being part of my committee, and for his insights and guidance, both in this thesis and in the projects I have had the privilege of collaborating with. I really appreciate Phillip's ability to contextualize projects as part of new paradigms, and his vision and originality have inspired a lot of my research. Furthermore, I want to thank him for building a great lab environment with peers from whom I have learned a lot and that I am happy to call my friends.

I also want to thank Josh Tenenbaum for inspiring me to work in human behavior modeling and build more cognitively inspired learning systems and for his guidance throughout the work in this thesis. I am also thankful for his insights on how to think about new problems and how to break big ideas into research experiments. This thesis would not be possible without my collaborators, Tianmin Shu, Shuang Li, Yonglong Tian, Ali Jahanian, Marko Boben, Kevin Ra, Zilin Wang, Clinton Wang, Ekin Akyürek, Yilun Du, Jacob Andreas, Igor Mordach, Yuan-Hong Liao, Jiaman Li, Tingwu Wang, Tete Xiao, Bolei Zhou and Hang Zhao. I want to specially thank Tianmin for greatly influencing my research interests, for the great times thinking about research directions, and for his support throughout this journey. I have also been fortunate to intern two times at Google Research, from which I had the chance to focus on new problems and explore research in industry. Thank you Dilip Krishnan, Li Zhang, Aaron Maschinot, Aaron Sarna, Yukun Zhu, and Max Collins for welcoming me in your groups, giving me freedom to explore but also guidance, and for the inspiring conversations and for your effort connecting me to other researchers and members of the research community at Google.

Over these years, MIT has been like a second home. I owe this to the great people both in the lab, and the wider embodied intelligence group, many of whom have become great friends. Thank you Yusuf Aytar, Hyojin Bahng, David Bau, Manel Baradad, Lucy Chai, Nadiia Chepurko, Brian Cheung, Ching-Yao Chuang, Abhishek Gupta, Minyoung Huh, Aditya Khosla, Agata Lapedriza, Shuang Li, Yunzhu Li, Yen-Chen Lin, Joseph Lim, Wei-Chiu Ma, Javier Marin, Nuria Marzo, Joanna Materzynska, Josep Marc Mingot, Dim Papadopoulos, Amaia Salvador, Swami Sankaranarayanan, Sarah Schwettmann, Pratyusha Sharma, Tianmin Shu, Didac Suris, Yonglong Tian, Carl Vondrick, Tongzhou Wang, Jonas Wulff, Hang Zhao, Hengshuang Zhao and Jun-Yan Zhu. I have really enjoyed our lunch breaks, weekend outings and trips together. Special thanks to Adria, Josep-Marc, Aditya and Bolei, for making my arrival so easy and enjoyable, and for all your support and mentorship. Thank you also to the amazing friends I have made in Boston, you have been a really important part of my PhD. Thank you Junkal, Ximo and Livia for always being there, to my roommates,

Hector, Ines, Guille, Paula and Ranjani for helping me make a home in Boston, and to Erica and Christian for helping me explore new hobbies and passions. Thank you also to the Spanish community at MIT, for always making me feel at home.

Despite many years living away from home, I am fortunate to have had people who always make my feel like I never left. Thank you Miriam for being the person I could always trust and thank you to *La Caracola Pesquera*: Gon, Marc, Vilo and Albert for all these years of friendship and for keeping me grounded but also imaginative and cheerful. To my family for all the unconditional support. Gràcies Mama, Papa, Albert, Guillem i Smoothie. Gràcies Mama per fer-me sentir sempre tan estimat, per inspirar-me a treballar pel que desitjo i a ser millor persona, i gràcies per matenir sempre la familia unida. Gràcies Papa per inspirar-me des de petit, per entendren's i seguir connectat amb nosaltres malgrat la distancia. I gràcies Albert i Guillem, per ser els millors germans que podria demanar, sé que sempre podré comptar amb vosaltres. Us admiro i estimo molt. And finally to Van Anh, for all her love and support, and for being the best companion at the end of this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A long-term goal of AI is to build general-purpose autonomous agents that can assist us with everyday tasks, that is, agents that can understand the goals we want to achieve and interact with us to complete them effectively and safely. This is a task that not only will help us study many core scientific problems in Artificial Intelligence or Robotics, but one that has the potential to transform our lives, from providing accessible childcare or eldercare at a time of labor shortage [117], assisting people with disabilities, to helping us at home or the workplace.

Over recent years, significant progress has been made in building and deploying robots that can perform complex tasks, with notable successes in spaces such as manufacturing or warehouse logistics. However, in most cases, these robots have focused on very specific tasks with minimal human interaction. The challenge in developing more general-purpose assistive agents is that they need to be able to perform a much broader set of tasks and skills in more unconstrained environments. Moreover, they need to interact concurrently with humans who may exhibit very diverse behaviors and must be kept safe. How can we build agents with these properties?

One possibility is to build agents that learn to interact through real-world experience. A variety of techniques in machine learning has allowed the development of agents that can learn tasks from large-scale interactions in real environments [49, 51, 79] or datasets of human demonstrations [9, 115]. The advantage of this approach is that the real world provides a source of rich data that may be valuable

to learning complex tasks, as well as information about how humans behave in these settings. At the same time, it is costly to obtain this source of data, as it requires thousands of hours of interactions in the environment, or human demonstrations. This makes it hard to scale to a large number of tasks. Furthermore, as agents interact in the environment, they could take actions that put humans in danger, posing safety challenges, particularly for tasks involving interactions with humans.

A second approach is to develop world models that agents can leverage to learn to perform tasks more effectively. These can include models of the environment [57, 12], modeling the state changes as agents take actions in it, or models of humans, describing the actions they could take given an environment state [26, 104, 65, 119]. World models allow us to develop agents safely and with fewer data, but the resulting agents will only be useful if the models are sufficiently representative of the real world. Given the difficulty of learning these kinds of models, many of these approaches focus on reduced state and action spaces, ignoring physical or perceptual properties that agents may need to reason with when interacting in the real world. To address this, there has been an increasing interest in developing 3d simulators that can provide realistic representations of the environment [121, 20, 160, 74, 45, 42, 68]. These simulated environments allow us to train and test agents under more realistic conditions at scale and have enabled agents to learn behaviors that can transfer into the real world [150, 76, 8]. Despite these successes, most current simulation approaches have focused on agents interacting *in isolation* in the environment, ignoring the effect of humans in the environment or the agent behavior. In order to build agents that successfully assist humans, we need to test them in more human-centric settings, with models of how these humans behave when interacting with agents, metrics to evaluate cooperation with humans and algorithms that allow agents to coordinate with humans safely.

This thesis investigates how to build socially intelligent agents that can assist humans in household activities via simulation environments. For this, we need to make progress in three areas that will be guiding this work. First, we must consider the kinds of platforms and data necessary to develop and test assistive agents safely.

Second, we need to develop agents that humans can command easily and that can operate on these platforms to do everyday tasks. Finally, we need to study how these agents should interact with humans safely and effectively to be helpful in practice.

In Part I of this thesis, we discuss what properties are required in a simulator to enable the development of socially intelligent agents. To address these, we present in Chapter 2 VirtualHome, a multi-agent platform to simulate household activities. In contrast to current simulators, VirtualHome focuses on representing both complex environments and tasks and the humans interacting in these environments. This is a key property to ensure that agents can be tested and trained to operate in realistic settings with other humans. While VirtualHome allows representing the activities we want agents to perform, we still need information about how these activities should be performed. For this, we introduce, in Chapter 3, VirtualHome ActivityPrograms, a knowledge base of household activities executable in the simulator and natural language descriptions for each activity. With this knowledge base, we can use VirtualHome to generate demonstrations of real-world activities, which can be used to instruct agents how to perform household tasks.

In Part II, we present different approaches to build agents that can perform everyday activities in VirtualHome. We aim to build agents that can not only perform complex tasks in the environment but also easily commanded by humans. We thus introduce, in Chapter 4, agents that perform activities by translating a natural language description or a video demonstration of the task into actions that they can execute in the environment. In some cases, it is impossible to perform the exact instructions in the environment. In Chapter 5, we present agents that infer the implicit goal of the human and adapt their actions to new environments to achieve these goals, rather than perform a literal translation of the given instructions. The work presented in these two chapters allows us to build agents that can perform a wide range of tasks in the environment while providing a natural interface to instruct them, which is crucial if we want agents to assist everyday users.

Under Part III, we study how to build agents that not only operate in isolation in the environment but can assist humans as they are also interacting in it. This is

the setting in which we expect agents to operate in real life. To provide a scalable way to test this setting, we develop agents that serve as a proxy for humans in the environment and propose different agents that can infer their goals and assist these proxy-humans in performing them. We propose two benchmarks to evaluate the effectiveness of assistive agents, which we measure by their capacity to infer the human goal and how efficiently they assist humans in such goal. In the first challenge, *Watch-And-Help*, described in Chapter 6, agents observe a human activity and are asked to assist a human in performing that same activity subsequently. In Chapter 7, we introduce the second challenge, *Online Watch-And-Help*. Here agents are not given a prior demonstration of the task and have to concurrently infer the task and assist the human with what they know about the task so far. We propose an approach that leverages uncertainty over the inferred goal so that the agent can focus on the parts of the task that it is more sure about and correct its behavior as it gathers more evidence about the task they need to perform. We conduct studies with real humans participating in both challenges, assisted by the proposed agents. Our results show that the assistive agents can be helpful to real humans and that testing these agents with proxy humans can be indicative of their performance when assisting real ones.

This thesis covers content from [109, 84, 110, 111]. I want to credit my co-authors for their excellent work; this thesis would not be possible without them.

Part I

A platform to develop assistive agents

Our goal is to build socially intelligent agents that operate in simulation. Thus, the first question we need to address is what kind of platform is needed to develop our agents. We focus on studying agents that can perform daily tasks in a household, so our platform should be able to represent indoor environments, and support everyday household activities.

The study of general-purpose agents that can perform indoor tasks has gained wide popularity over recent years, spawning a wide number of environments [74, 80, 45, 161] and benchmarks [127, 14] to study this problem. While some of these works were posterior to the start of this thesis, why not leveraging these platforms and benchmarks to study our questions? Most of these platforms focus on testing agents that interact isolated in indoor environments, ignoring the effect that other humans in the environment may have in the agent's behavior. In order to study agents that coordinate with and assist humans, we need to be able to represent these humans in the environment.

Part I of this thesis presents the platform we build in order to achieve this. In Chapter 2, we present VirtualHome a multi-agent household activity simulator. Like [80, 74, 145, 43], VirtualHome allows to represent complex indoor environments and a wide range of actions that can be combined to perform long horizon tasks. Unlike these works, VirtualHome includes models of humans that execute these tasks in the environment, allowing to 1) generate demonstrations of human avatars performing these tasks, providing a way to instruct agents and 2) represent humans that can interact with our proposed agents to perform complex activities.

To provide agents with information about how to do everyday tasks, in Chapter 3 we present a knowledge base of household activities, with natural language descriptions of how to perform them, and instructions allowing to represent them in our simulator.

Chapter 2

VirtualHome: A Multi-Agent Household Simulator

2.1 Introduction

The goal of a virtual environment here is to provide a playground for agents to perform tasks before being deployed in the real world. Given that it is impossible to build a simulator that perfectly represents reality, we need to define which kind of tasks and domains should be the focus of our platform. Here, we want a simulator where agents can learn to perform household activities in indoor environments, understand the activities that humans are doing in the environment, and assist humans in doing them. Thus, we aim to design a platform with the following properties:

- **Household Environments:** the simulator should represent indoor household environments, with different rooms, and objects that agents can interact with.
- **Long-Horizon Tasks:** we want to focus on representing long-horizon tasks, including multiple object interactions. Rather than focusing on low-level manipulation problems, the simulator should support mid and high level action primitives, allowing to easily specify complex tasks.
- **Human-Centric:** just like robots do not act in isolation in the real world, the simulator should include human-like agents that can perform tasks in the

environment as other agents are interacting in it.

- **Realistic:** the simulator should represent realistic environments, including the appearance of objects and scenes, as well as the object dynamics as agents interact with them. It should also support realistic human motions, so that agents can infer which actions or activities humans are doing.

While in recent years there have been multiple simulators to represent household environments [121, 20, 160, 74, 45, 42, 68], they generally focus on the interaction between agents and the environment, and not on modeling humans that may be interacting in it. To address this, we propose VirtualHome, a 3D multi-agent household simulator for human-AI collaboration. Like [121, 20, 160, 74, 45, 42, 68], VirtualHome focuses on representing realistic household environments, with interactive objects that change their states as agents interact with them. The simulator supports mid-level (e.g. move forward, turn left) and high-level actions in the environment (e.g. walk to the kitchen, place apple in fridge), allowing to represent long-horizon tasks with few commands. Unlike existing simulators, agents are represented as humans, that are animated to display realistic motions when doing actions in the environment. These humans can interact in the environments as other agents are doing tasks in it, allowing to develop agents that can coordinate with them effectively.

The goal of VirtualHome is to provide a platform that can be used to teach agents how to interact in environments, but also to understand and represent human behaviors and activities, and study how agents should interact or assist these humans. In the rest of this chapter, we describe the main properties of the simulator and compare it with similar platforms.

2.2 Comparison with Existing Platforms

Given the wide use of simulation platforms in many scientific and engineering fields, we focus here in comparing VirtualHome with platforms that involve household environments.

Navigation. There have been many virtual environments designed to develop agents that could interact in a household. Some of these works [121, 120] focus on providing photo-realistic representations of household environments and use them to train agents that can learn to navigate from visual input, showing that agents trained in simulation can transfer these skills in the real world [150, 69]. However, these works do not support interactive objects and therefore are limited to navigation tasks that do not require interaction.

Manipulation. Simulation environments are also widely used for robotic manipulation, and a few works focus on manipulation in the context of household objects or environments [78, 161]. Lee et al. [78] provide an environment for furniture assembly tasks, and [161] provides a suite of rigid, part-based and soft-body everyday objects to learn manipulation tasks. However, these environments do not support long-horizon tasks, or actions and object states that are required to represent everyday activities. In our work, we study long-horizon activities and chose to focus on the task-planning aspects associated to it. Thus, we provide high-level actions that assume access to low-level manipulation primitives and we abstract away low-level physical or material properties of the objects in the environments.

Long-Horizon Tasks. Most related to our work are simulators that focus on long-term activities [26, 42, 74, 145, 80, 20]. These platforms provide higher level commands that allow for long-horizon tasks with a fewer set of actions. They include interactive objects, that change their state when agents take actions on them or due to the effect of time. For instance, [80] includes objects that can be cooked or get dirty over time. Some of these simulators allow for multi-agent interactions, but focus on developing agents that are jointly controlled to perform tasks in the environment. In our work, we focus on modeling human activities as well as humans that can interact with agents in performing these activities. For this reason, we include human agents with actions and motions that allow to represent human behaviors. Posterior to our work, [45] proposed an environments with humanoids that could represent human activities. However their focus is only in kitchen environments, limiting the range of possible tasks. Furthermore they only support a single agent at a time, restrict-

Platform	Action	Views	Realistic	Humanoid	Multi-agent
Overcooked [26]	High/Low	3rd Person	No	No	Yes
TDW [42]	High/Low	3rd Person/Ego	Yes	No	Yes
VRKitchen [45]	High/Low	3rd Person/Ego	Yes	Yes	No
AI2-THOR [74]	High/Low	Ego	Yes	No	Yes
iGibson [80]	Low	Ego	Yes	No	No
Habitat 2.0 [145]	Low	Ego	Yes	No	No
VirtualHome	High/Low	3rd Person/Ego	Yes	Yes	Yes

Table 2.1: We compare VirtualHome with existing embodied single-agent and multi-agent platforms on the following aspects: 1) action space (high-level actions and/or low-level actions), 2) views (3rd person and/or egocentric views), 3) realistic environments, 4) humanoid agents and 5) multi-agent capabilities.

ing from studying Human-AI collaboration scenarios. Table 2.1 summarizes the key features of the proposed VirtualHome in comparison with these virtual platforms. The key features of our environment include i) multiple camera views, allowing to interact in the environment and generate demonstrations of tasks, ii) both high-level and low-level actions, iii) humanoid avatars with realistic motion simulations and iv) multi-agent capacities. Critically, VirtualHome enables collecting and displaying human activities in realistic environments, which is a key function necessarily for social perception and human-AI collaboration. In contrast, existing multi-agent platforms do no offer such functionality.

Humans. There has been an increasing interest in representing humans in simulation. Most of these works focus on representing human motions or low-level actions, collected or learned through human motion capture data [153, 106, 53, 170, 16]. While these works enable a high fidelity representation of human poses and motions, they model humans in static environments or interactions with a limited set of objects [16] and therefore do not allow to represent activities that involve interacting with the environment. Moreover, they are designed to generate images and videos and not as a platform where agents can interact concurrently. Recent work [80] has started focusing in representing humans to be tested with agents but they only focus on navigation tasks. There is a growing interest in representing realistic interactions between humans and robots [28, 38], but they provide very simple representations

Figure 2-1: Agents available in VirtualHome.

of humans, in some cases being passive bodies, or focus very specific tasks, such as object handovers [28]. Humanoids in VirtualHome are driven via animation and inverse kinematics, and therefore exhibit less realistic motions than some of the above works, but they can interact in the environment with other agents, and represent a wide variety of tasks which is key to study human-AI collaboration.

2.3 Main Concepts

We implemented our VirtualHome simulator using the Unity3D game engine which allows us to exploit its kinematic, physics and navigation models, as well as user-contributed 3D models available through Unity's Assets store. The simulator is designed to allow agents to interact in indoor environments, or render videos of human activities. All interactions in VirtualHome work through three components: agents, representing human avatars that will perform actions; environments, representing different apartments with objects that agents can interact with, and programs, that define how agents interact with the environment. We describe these components below.

2.3.1 Agents

Agents are represented as humanoid avatars which are rigged and animated to generate plausible motions when they interact in the environment. Agents are equipped

with navigation models, allowing to navigate towards different objects or rooms in the environment using shortest-path planning. The available agents are shown in Figure 2-1.

2.3.2 Environments

Agents interact in household environments, containing different rooms and interactive objects in them. The simulator contains 7 different apartments, which we obtained from the Unity Assets store. On average, each home contains 357 object instances (86 per room). To support a wider range of activities, we collected objects from additional 30 object classes via the 3D warehouse [1]. To ensure visual diversity, we collected at least 3 different models per class. The apartments are shown in 2-2. The apartments can be modified by adding, removing, and changing the location of the objects in them. Furthermore, VirtualHome allows to generate new apartments procedurally, modifying the number of rooms, their layout and the objects in them, allowing to generate an infinite number of unique environments.

All environments are represented via an *EnvironmentGraph*, a scene graph where every node corresponds an object in the environment and edges represent spatial relationships (e.g. "apple inside fridge", "plate on table", "table near fridge"). Each node is annotated with an identifier number, the object name, its location in the environment and size, and its states (e.g. "microwave is on and closed").

2.3.3 Activities via Programs

Our simulator should allow for both humans and robots to perform complex activities in the environment. We aim to provide a representation that can unify how different agents perform activities, abstracting their differences through action primitives. To this end, we propose to represent these activities via programs, describing the steps to accomplish a given task. A *program* contains a sequence of simple symbolic instructions, each referencing an atomic action (e.g. "sit") or interaction (e.g. "pick-up

[1] https://3dwarehouse.sketchup.com.

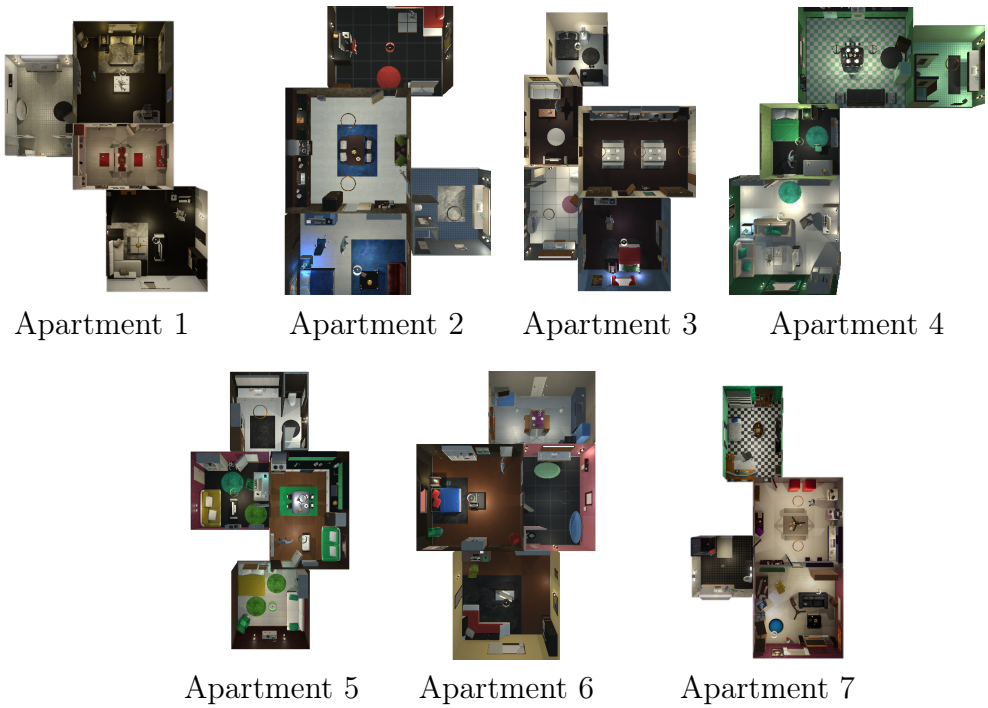


Figure 2-2: 3D households in our VirtualHome. Notice the diversity in room and object layout and appearance. Each home has on average 357 objects.

Table 2.2: List of atomic actions available in VirtualHome.

Action name	Command	Description
Walk	[Walk] $1	Walks to room or object $1
Run	[Run] $1	Runs to room or object $1
Walk Towards	[Walktowards] $1	Walks 1 meter towards room or object $1
Walk Forward	[Walkforward]	Walks 1 meter forward
Turn Left	[Turnleft]	Rotates 30 degrees left
Turn Right	[Turnright]	Rotates 30 degrees right
Sit	[Sit] $1	Sits on object $1
Stand Up	[Standup]	Stands up
Grab	[Grab] $1	Grabs object $1
Open	[Open] $1	Opens object $1
Close	[Close] $1	Closes object $1
Put On	[Putback] $1 $2	Puts object $1 on top of object $2
Put In	[Putin] $1 $2	Puts object $1 inside of object $2
Switch On	[Switchon] $1	Turns on object $1
Switch Off	[Switchoff] $1	Turns off object $1
Drink	[Close] $1	Drinks from object $1, if in hand
Touch	[Touch] $1	Touches object $1
Look At	[Lookat] $1	Turns to object $1

object") and a number of objects that the action refers to (e.g., "put juice on table"). Instructions may also contain modifiers, providing detail about how an action should be performed ("e.g. put juice on table at coordinates (3,2)").

More precisely, a program is represented as a sequence of steps, where step t can be written as:

More precisely, step t in the program can be written as the instruction:

$$\text{step}_t = [action_t] \ \ \langle object_{t,1} \rangle (id_{t,1}) \ ... \ \langle object_{t,n} \rangle (id_{t,n}) \tag{2.1}$$

Here, *id* is a unique identifier of an object and helps in disambiguating different instances of objects that belong to the same class. An example of a program for "watch tv" would be:

$$\begin{aligned}
&\text{step}_1 = [\text{Walk}] \ \ \langle \text{TELEVISION} \rangle (1) \\
&\text{step}_2 = [\text{SwitchOn}] \ \ \langle \text{TELEVISION} \rangle (1) \\
&\text{step}_3 = [\text{Walk}] \ \ \langle \text{SOFA} \rangle (1) \\
&\text{step}_4 = [\text{Sit}] \ \ \langle \text{SOFA} \rangle (1) \\
&\text{step}_5 = [\text{Watch}] \ \ \langle \text{TELEVISION} \rangle (1)
\end{aligned}$$

Here, the program defines that the television in steps 1, 2 and 5 refer to the same object instance. The identifier can represent a counter (as in the example above), or a particular object instance in the environment. In the first case, the program specifies that the same object instance should be used for different instructions, whereas in the second case, the program specifies that only the object with the specified identifier should be used. We describe in Section 2.4.1 how this matching takes place.

This allows agents to perform actions at different levels of detail. VirtualHome currently supports 18 unique atomic actions, involving interactions with objects (e.g. "pick-up apple"), navigation commands (e.g. "run to the kitchen") or animations (e.g. stand up). Table 2.2 shows the list of available actions.

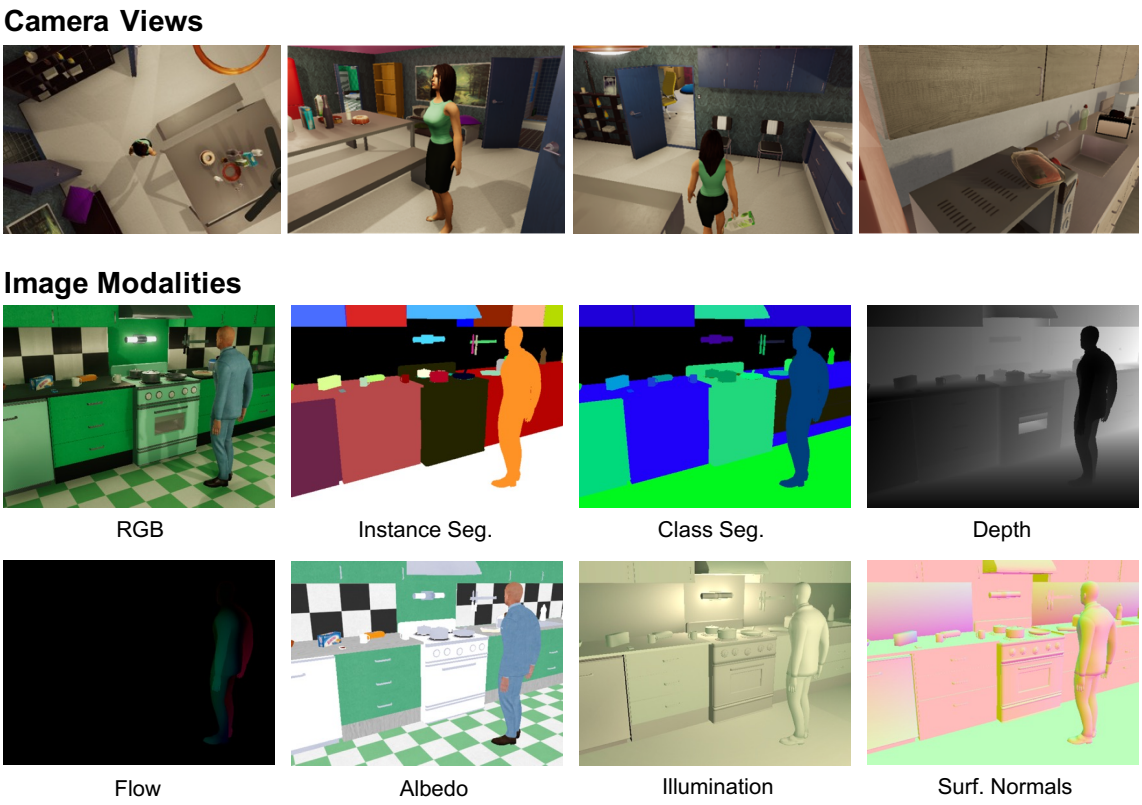


Figure 2-3: Different camera views and input modalities supported in VirtualHome.

2.4 Performing activities in VirtualHome

VirtualHome is designed both to support agents interacting with the environment, allowing to train them to perform tasks, and to generate videos of human activities. Thus, the simulator relies on two modes of operation, which are used depending on the use case: an *interactive mode*, and a *video mode*.

In the *interactive mode*, agents can take an action at each step, and receive an observation in return. Note that multiple agents can take an action at the same step, in which case, each will receive their own observation. The observations can be symbolic (e.g. in the form of a scene graph, as described in 2.3.2, depicting the current state of the environment), include 3D information about the environment and agents' poses, or be visual. Visual observations can come from multiple cameras, either static in the scene or attached to the agent, and different modalities, including depth, optical flow, semantic and instance segmentation, or surface normals. Figure 2-3 shows an

overview of some of the available camera views and input modalities.

The *video mode*, is designed to generate a video of one or multiple agents doing a task in the environment given an activity program, as described in 2.3.3. Here, users can also specify what cameras and input modalities to record during the activity, but instead of obtaining an observation at each step, the simulator generates a video of the full activity, with time-stamps for each step. Thus, we need a way to animate the programs into motions, in order to generate a temporally consistent video.

2.4.1 Animating Programs in VirtualHome

Every step in a program requires us to animate the corresponding (inter)action in our virtual environment. We thus need to both, determine which object in the home (which we refer to as the *game object*) the step requires as well as properly animating the action. We now describe this process in more detail.

Animating atomic actions. We animate the actions in Table 2.2. Note that there is a large variability in how an action is performed depending on to which object it is applied to (e.g., opening a fridge is different than opening a drawer).

We distinguish between actions that require interacting with objects and actions that do not. The later ones correspond to locomotion actions (*walk* and *run*) and *standup*. Locomotion actions are implemented using Unity's NavMesh framework for navigation (path planner to avoid obstacles) and *standup* is implemented through an animation. For the actions requiring objects, we compute the agent's target pose and animate the action using RootMotion FinalIK inverse kinematics package. To allow for more realistic animations, we annotate typical hand poses for humans when interacting with objects, shown in Figure 2-4. We further animate certain objects the agent interacts with, e.g., we shake a coffee maker, animate toast in a toaster, show a (random) photo on a computer or TV screen, light up a burner on a stove, and light up the lamps in the room, when these objects are switched on by the agent. The animation is done by keeping a state over the object, and changing it when the character's hand reaches the object.

Executing a Program. To animate a program we need first to create a mapping

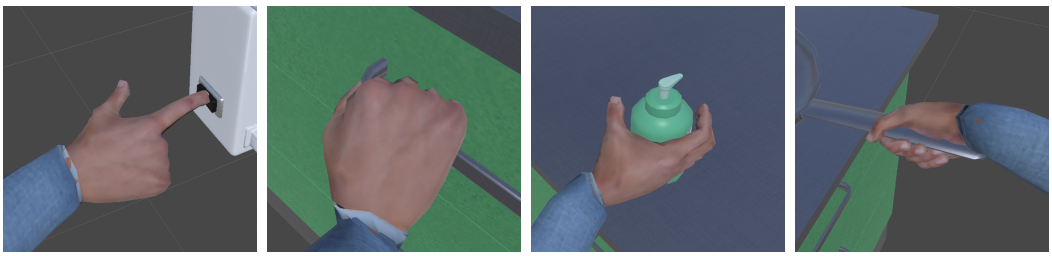


Figure 2-4: Examples of how our hand poses are used while interacting with an object. We support both left and right for each hand pose.

between the objects in the program and the corresponding instances inside the virtual simulator, which we will cal *game objects*. Furthermore, for each step in the program, we also need to compute the interaction position of the agent with respect to an object, and any additional information needed to animate the action (e.g., which hand to use, speed of the action, orientation). As described in Section 2.3.3, the objects in the program contain identifiers, which may correspond to object counters, or to a particular object instance. When the identifier corresponds to an instance, we map the objects that have matching instance identifiers. performing the mapping is straightforward, since it should have a matching identifier. When it corresponds to a counter, we need to solve an optimization problem by taking into account all steps in the program and finding a feasible path. For example, if the program requires the agent to switch on a computer and type on a keyboard, ideally the agent would type on the keyboard next to the chosen computer and not navigate to another keyboard attached to a different computer in possibly a different room. For this, we build a tree of all possibilities of assigning game objects to objects in the program, along with all interaction positions and attributes. To traverse the tree of possible states we use backtracking and stop as soon as a state executing the last step is found. Since the number of possible object mappings for each step is small, and we can prune the number of interaction positions to a few, our optimization runs in a few seconds, on average.

Generating a video. Once the mapping between objects is done and the interaction information for each step is computed, we can generate a video of the activity described in the program. For this, we randomly illuminate the scene to make it

realistic while ensuring visual diversity and record the activity from different cameras within the environment. We allow the user to select which camera to record from, or to let the simulator decide it. In the latter case, we use the static cameras in the environment and switch the recording camera based on agent's visibility. In particular, we randomly select a camera which sees the agent, and keep it until the agent is visible and within allowed distance. For agent-object interaction we also try to select a camera and adjust its field of view to enhance the visibility of the interaction. We further randomize the position, angle and field of view of each camera to ensure diversity in the generated videos.

2.5 Symbolic Simulator

In some cases, rather than rendering videos or obtaining precise 3D information of an environment, we may simply be interested in simulating an activity at a high level, obtaining a symbolic representation of the environment for each action. For instance, we may be interested in knowing at a high level the effects of a given action, or obtain a coarse world-model to embed it into a symbolic planner. For this, one possibility is to use VirtualHome and only obtain symbolic observations for every action. However, the simulator is still storing and updating 3D information in the background and therefore using computation and memory that we may not need. For this, we build *VirtualHome-Symbolic* a very light-weight simulator in Python that can be used as a proxy for VirtualHome when we only care about symbolic states.

The simulator represents states as symbolic graphs and consumes actions that change either the states of nodes (e.g. a fridge becomes *closed* after executing a `Close` action) or edges between nodes (e.g. after grabbing an apple from the table, an `ON` edge is removed between the apple and the table, and a `HOLDING` edge is added between the apple and the agent). Both the available actions and object states are a subset of those supported in VirtualHome. More importantly, the format of the symbolic graphs in this simulator are compatible with those in VirtualHome. This means that when simulating actions in VirtualHome, we can at any time transfer

the state and execution into this simulator, run a few actions there and transfer the execution back into VirtualHome.

We will be using this simulator in Chapters 5, 6, 7 to 1) extend a dataset of human activities and programs and 2) obtain a world-model that can be consumed by a symbolic planner, which will be driving agents in the environment.

2.6 Discussion

In this chapter, we introduced VirtualHome, a household activity simulator to test and develop assistive agents safely and at scale, before deploying them in the real world. VirtualHome focuses on representing both interactive environments, allowing agents to learn to interact with objects and perform complex tasks, and humans, which allows to test these agents in more realistic assistive scenarios. The simulator can be used both to train and test interactive agents, as well as to generate videos depicting human activities with labels, providing a source to train models for activity understanding. These features will allow us to build agents that can perform complex activities given demonstrations, and infer human goals form observing their actions in the environment, as we will discuss in the following chapters. Furthermore, the versatility of the simulator has also allowed different communities to use VirtualHome for a variety of purposes, including language-based planning via large language models [81, 60, 136], scene and object reasoning [17, 102] or smart-home research [19].

While VirtualHome allows to represent and learn a variety of household tasks and behaviors, we still need to define which behaviors we need to focus on in order to build useful assistive agents. Thus, in Chapter 3 we describe our efforts to build a knowledge base of activities that can be represented in the simulator, along with natural language descriptions of these activities, allowing to drive agents, not by a sequence of program commands but via language instructions.

Chapter 3

A Knowledge Base of Household Activities

3.1 Introduction

The ability to simulate household activities in a virtual environment allows us to quickly generate demonstrations of tasks that agents can learn from, as well as ensuring that these agents can learn to interact in the environment in a safe and scalable manner before deploying them into the real world. In the previous chapter, we introduced VirtualHome, a platform to simulate these activities in realistic indoor environments. However, for it to be a useful tool to build assistive agents, we need information about which types of activities need to be represented, and how these activities should be performed.

We aim to build agents that help humans in performing everyday activities. For this, we want agents that can follow human commands, allowing them to perform the task a person may desire. Tasks here are represented as programs, but asking humans to instruct agents through those would require to learn a domain-specific syntax and be time consuming. Instead, we want to teach tasks to agents just like we would teach a human, through a description of the task or a demonstration of how to perform it. Our goal is thus to be able to automatically be able to generate activity programs from natural language descriptions, as well as from video demonstrations, potentially

allowing naive users to teach their robot a wide variety of novel tasks.

Towards this goal, one important missing piece is the lack of a database describing everyday activities, paired with programs specifying each of the steps needed to complete them. Such programs can then be used to instruct a robot or an agent on how to perform the activity or be rendered in the simulator, allowing to generate demonstrations for each task.

For this, we aim to build a large repository of common activities and tasks that we perform in our households in our daily lives. To ensure we can collect a large and diverse dataset, we will crowd-source the knowledge base. We first collect common-sense information about typical activities that happen in people's homes, such as "watching TV" or "make coffee with milk', along with a description of the task, forming the natural language know-how of how these activities are performed. Then, we ask people to write programs that formalize each activity as described in the knowledge base, as defined in section 2.3.3. Note that these programs include all the steps required for the robot to accomplish a task, even those that are not mentioned in the language descriptions and that humans would assume common-sense.

This chapter describes our efforts to build such knowledge base, providing *ActivityPrograms*, a dataset to teach agents how to perform everyday activities.

3.2 Related work

Activity datasets for robots and agents. A common way to teach robots or agents complex tasks is to train them via expert demonstrations [9, 115]. While these demonstrations typically require expert policies or access to the robot's or agent's state, some works learn behaviors from videos of human demonstrations [112, 122, 125, 167], which are easier to collect. However, most of these works focus on navigation or low-level manipulation tasks, and not the long term interactions needed to perform household activities.

A few works have focused on learning more complex activities from demonstrations by representing them as programs. In [165], they teach robots to cook recipes from

cooking videos by detecting objects and actions, and using a probabilistic grammar to generate an action plan. [123, 4] also argued for actions as a sequence of atomic steps. They aligned YouTube how-to videos with their narrations in order to parse videos into such programs. Most of these works were limited to either a small set of activities, or to a narrow domain (cooking). We go beyond this by creating a knowledge base about an exhaustive set of activities and tasks that people do in their homes.

Some works [134, 132, 41] have focused on building datasets of human activities at home, that can be used to teach robots household tasks. [134] crowd-sourced scripts of people's actions at home in the form of natural language. These were mostly comprised of one or two sentences describing a short sequence of actions. While this is valuable information, language is very versatile and thus hard to convert into a usable program on a robot. We show how to do this in our work.

Despite the potential of activity datasets to learn complex robot or agent behaviors, these require grounding videos into agent actions and the environments where they will interact, which is a complex task. For this reason, building activity datasets that are grounded in an environment has been a promising direction to develop agents that can perform complex tasks. A number of works, concurrent with the development of this thesis, have made significant progress towards this goal [139, 127, 14, 157], building upon expert policies or data collection in VR. Like in our work, [127] uses human annotators to describe household activities via language, but generates actions of a given activity using a built-in planner, which does not necessarily represent human behavior. [139] uses a block-like programming language similar to ours to define activity pre and post-conditions, and collect a dataset of human performing those activities in VR.

3.3 Data Collection

In order to collect a diverse set of activities, we use crowd-sourcing to build our Knowledge Base. Describing activities as programs is not a trivial task, and using

crowd-sourcing makes it more challenging, as most annotators have no programming experience. We address this by splitting the data collection into two parts.

In the first part, we asked Amazon Mechanical Turk (AMT) workers to provide verbal descriptions of daily household activities. In particular, each worker was asked to come up with a common activity/task, give it a high level name, eg "make coffee", and describe it in detail. In order to cover a wide spectrum of activities we pre-specified in which scene the activity should start. Scenes were selected randomly from a list of 8 scenes (*living room*, *kitchen*, *dining room*, *bedroom*, *kids bedroom*, *bathroom*, *entrance hall*, and *home office*). An example of a described activity is shown in Figure 3-3. Note that these descriptions are written as if the activity was being described to another human. Because of this, they may likely omit some necessary steps that are commonsense, for example, opening a fridge before grabbing something from it and closing the fridge afterwards.

In the second stage, we showed the collected descriptions to the AMT workers and asked them to translate these descriptions into programs, telling them to produce a program that would "drive" a robot to successfully accomplish the described activity, without missing any step. We designed a programming language to represent each description, together with an interface so that annotators with no programming experience could write them. Our interface builds on top of Scratch [1], a visual programming language developed by the Lifelong Kindergarten Group at MIT Media Lab aimed at introducing young children into programming. In Scratch, people write programs by stacking blocks of instructions as if they were Lego pieces, allowing to create interactive stories, games or animations. We designed our language so that each activity program could also be written by composing blocks representing simple actions or interactions, such as *sit*, *walk*, *grab* or *open*.

Each block contains the name of an action and a list of arguments to be filled with objects. Actions like *stand up* would not need any argument whereas *put something into something* would require two arguments to be filled with objects. Each object is linked to an instance number so that the programs are not ambiguous when they refer to multiple objects belonging the same class: if an activity consists in setting

Figure 3-1: Example block to construct activity programs.

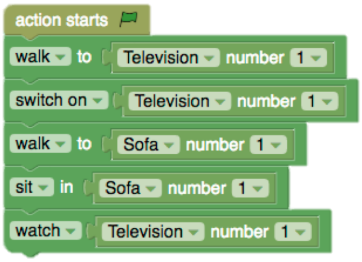


Figure 3-2: Example program for watch tv.

a table for two people, this allowed to disambiguate which plate should be used at every instruction. An example block can be seen in figure 3-1.

The blocks allow to select from a predefined list of 77 actions and 312 objects, compiled by analyzing the frequency of verbs and objects in the collected descriptions. We manually added affordance constraints to the objects, so that it is not possible to create invalid action/object combinations such as *grab shower*. We also allowed annotators to use a "special" block" which allowed to introduce missing actions as free-form text. These blocks were later discarded or replaced, but they allowed to identify blocks that needed to be added into the interface. Figure 3-2 shows an example of a program collected for "watch tv", where the instance number indicates that the television always refers to the same object instance.

Annotators were firstly shown a set of activities and descriptions to annotate, then they were asked to select, for each of them, the rooms and objects needed to perform the activity, and they finally created a program by using those items. Figure 3-3 shows the annotation interface.

We started annotating a small set of programs using Upwork crowd-sourcing platform, which allows to hire freelancers to perform jobs. While this approach was more expensive and slow than using AMT, it allowed to obtain very high quality programs, as well as feedback on how to improve the interface for easier crowd-sourcing. We later asked annotators on AMT to write programs for our collected descriptions,

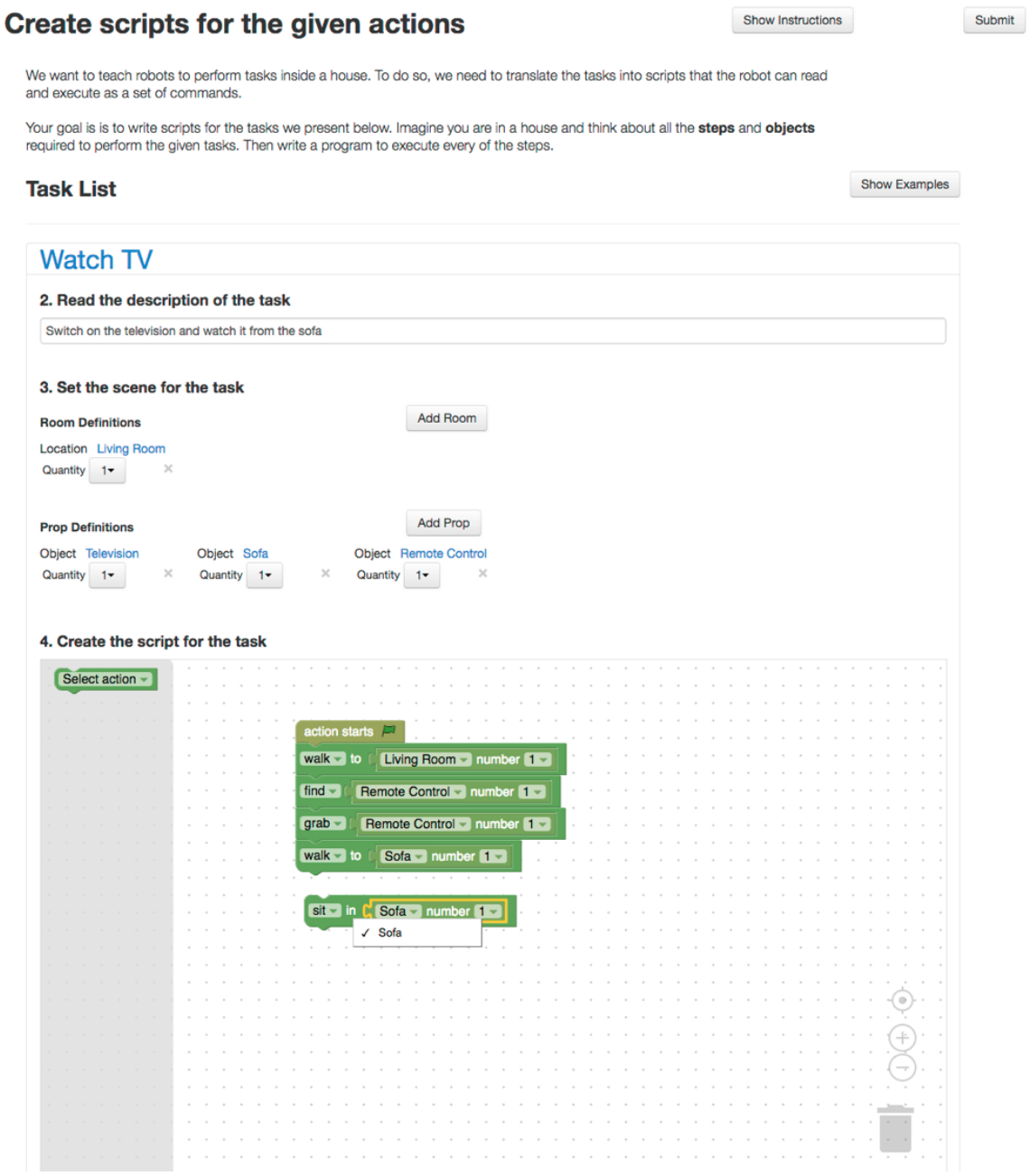


Figure 3-3: Interface for annotating programs from descriptions. Annotators would first read the description of the activity (step 2). They would set the scene (3) by adding the necessary objects and rooms and they would finally write a program by composing blocks (4).

which were finally validated by the annotators in Upwork. We found that annotators on AMT were capable to quickly learn to produce programs by providing a carefully designed tutorial, costing 15 cents per program annotated.

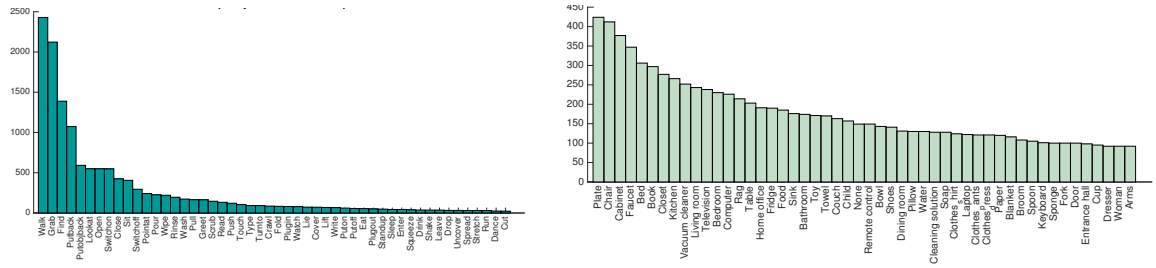
3.4 Dataset Analysis

In the first part we collected 1814 descriptions. From those, we were able to collect programs for 1703 descriptions. Some of the programs contained several "special blocks" for missing actions, which we remove, resulting in 1257 programs. We finally selected a subset of the tasks and asked workers to write programs for them, obtaining 1564 additional programs. The resulting 2821 programs form our *ActivityPrograms* dataset. On average, the collected descriptions have 3.2 sentences and 21.9 words, and the resulting programs have 11.6 steps on average.

The dataset covers 75 atomic actions and 308 objects, making 2709 unique steps. Figure 3-4.a shows a histogram of the 50 most common actions appearing in the dataset, and, Figure 3-4.b, the 50 most common objects.

The programs in the dataset represent 576 activities, each with several examples, and we analyze their diversity by comparing their programs. Table 3.1 analyzes 4 selected activities. We compute their similarities as the average length of the longest common subsequence computed between all pairs of programs within the activity. Figure 3-5 shows some example programs for the activities "Make coffee" and "Read a book".

We can also measure the similarity between activities by measuring the distance between programs. The similarity between two programs is measured as the length of their longest common subsequence of instructions divided by the length of the longest program. We can measure the similarity between 2 activities by taking the average similarity across programs belonging to the 2 activities. Table 3-6. shows the similarity matrix (sorted to better show the block diagonal structure) between different activities in our dataset. As it can be seen, the similarity measure allows to cluster semantically similar activities.



(a) (b)

Figure 3-4: Histogram of the most common actions (a) and objects (b) in *ActivityPrograms*.

Action	# Prog.	LCS	Norm. LCS
Make coffee	69	4.56	0.26
Fold laundry	11	1.29	0.08
Watch TV	128	3.65	0.40
Clean	42	0.76	0.04

Table 3.1: Analyzing diversity in the same activity by computing similarities across all pairs of the collected programs. "LCS" denotes longest common subsequence. For "norm.LCS" we normalize the LCS by the length of the longest of the two programs.

The collected programs also provide a source of commonsense knowledge. By looking at the instructions, one can infer information such as the relative location of objects (e.g. the nightstand is near the bed, Figure 3-5), or the affordance of certain obejcts (e.g. spectacles are used to read, Figure 3-5)

3.4.1 Completeness of programs

We analyze the correctness and completeness of the collected programs to execute the task they describe. To do so, we sample 100 of the collected programs, and ask 5 workers on AMT to rate each of the programs on whether they are complete, missing minor steps or missing important steps. A program is complete when it contains all the actions so that a robot could execute them, misses minor steps when the program says "sit on the sofa" but there is no previous instruction to "walk towards the sofa" and misses crucial steps when it lacks an action that is crucial to complete the activity, for example "switch on TV" in order to watch TV, or pour water into a cup if one needs to drink water. For each of the programs, we take the median score

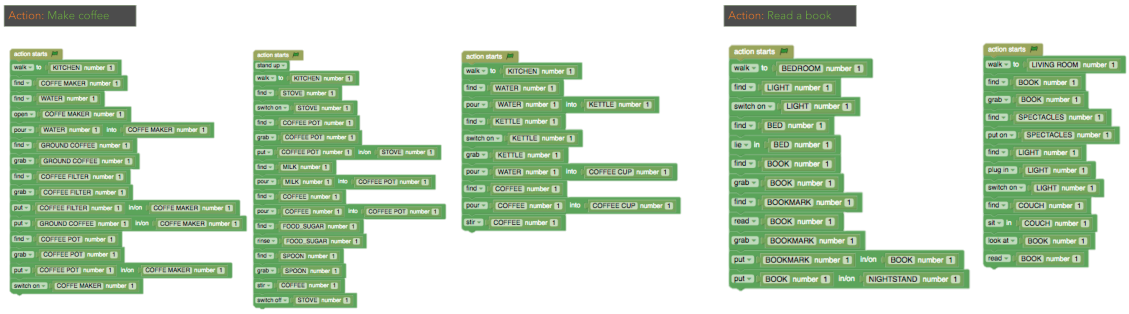


Figure 3-5: Examples of programs for the activities *Make a coffee* and *Read a book*.

from the 5 workers as a measure of the quality of that program. Results show that 64% of the programs are complete, 28% are missing minor steps and 8% are missing crucial steps. Note that many of the minor steps can be automatically corrected by, for example, adding walking actions before interacting with objects.

3.5 Discussion

This chapter presented our approach to build a knowledge base of household activities represented both via natural language descriptions and step-by-step instructions. Having access to a dataset of activity programs is highly valuable, since it provides a guide for agents to perform actions in the environment to achieve complex tasks. At the same time, having natural language descriptions of the tasks makes the dataset more easily readable for humans, but more importantly it provides a way to train models that can go from these descriptions to the activity programs, effectively allowing us to instruct robots via language. We discuss in Chapters 4 and 5 two approaches to do that.

Despite the value of building such knowledge base, our current approach also comes with limitations. First, the knowledge base is not grounded in any environment. The descriptions and programs are collected according to how the annotators think a given task would normally be done, and therefore assuming a common configuration of the environment. For instance, a program with the instructions “walk to livingroom", “turn on tv"", “watch tv" assume that there is a turned off tv inside the livingroom in the environment. If this was not the case, the program could not be executed

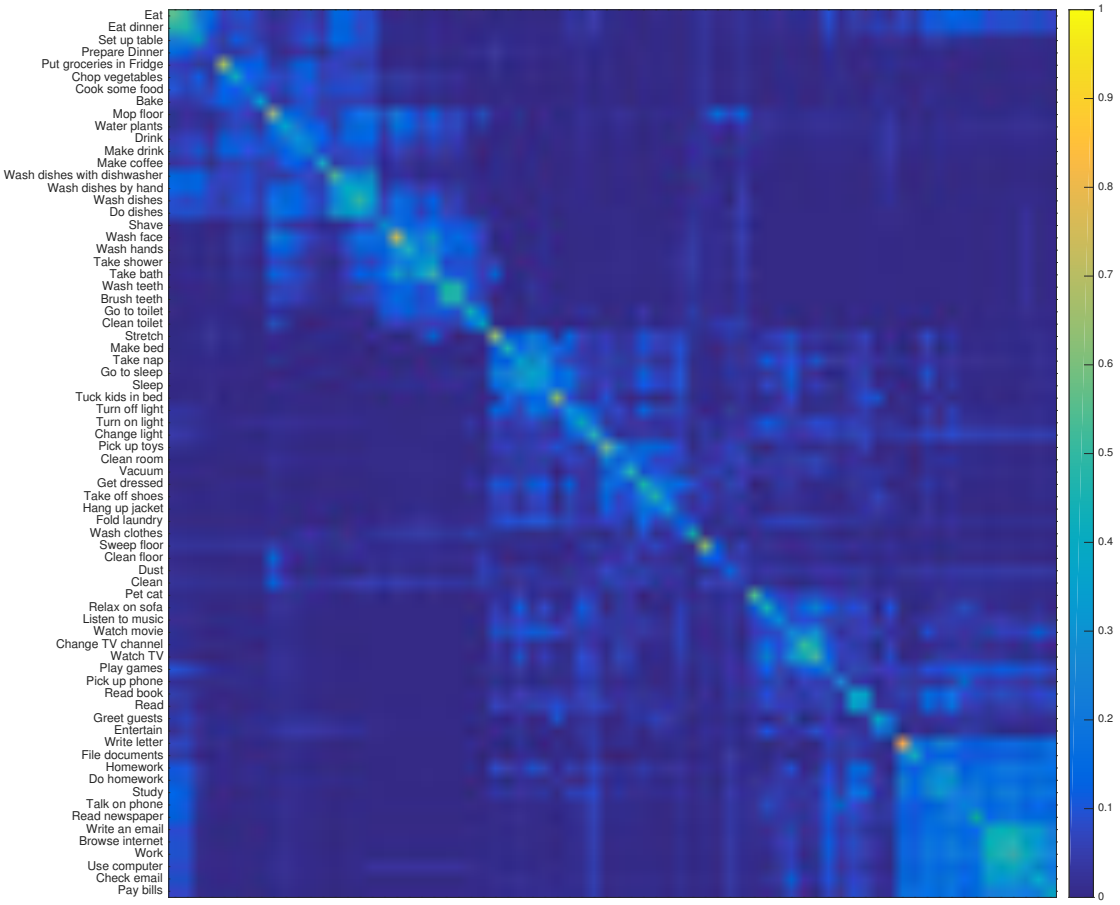


Figure 3-6: *ActivityPrograms* similarity matrix (sorted to better show the block diagonal structure) between different activities in our dataset.

and the task would fail. Programs should therefore include a series of assumptions of the environment. Second, while our programs specify all the required steps to perform an activity, there may be multiple valid ways to perform it, and it would be impossible to author a new program for each of these possibilities. Moreover, our program representations allow agents to execute tasks in an open-loop, but these tasks may fail if any of the individual instructions fails, or if the environment changes while the agent is performing the task. Representing programs as sequences of instructions makes them easier to collect and learn, but addressing the above issues would require more flexible representations or primitives, such as adding action preconditions or

describing tasks via goal specifications. We adopt the latter approach in chapters 6 and 7.

Part II

Agents that follow human instructions

In Part I, we proposed a platform for agents to learn to interact in household environments and perform daily activities in a safe and scalable manner before being deployed in the real world. Our next goal is to allow humans to command these agents to perform everyday tasks in our environment. While agents represent activities as programs, asking humans to use this representation would require them to learn a domain-specific syntax and specify all the steps involved in a given task, making the process time consuming and prone to errors. Instead, we want to be able to communicate tasks the same way we would instruct other humans, through a description or demonstration of an activity.

In this part, we present agents that can perform daily activities given this type of human input. We first describe, in Chapter 4 a model that translates language or videos into instructions that can be executed in VirtualHome, using the datasets described in Part I. In Chapter 5, we extend these agents to reason about the environment where they will perform the instructed activities, allowing them to adapt to situations that may not be explicitly instructed by humans.

Chapter 4

Agents that follow instructions and demonstrations

4.1 Introduction

For assistive agents to be helpful, we should be able to easily instruct them to perform the activities we need help with. In this thesis, we study agents that operate in a virtual environment, so our goal is to provide methods so that humans can command agents in VirtualHome. The programs and instructions presented in Part I allow agents to perform actions in our simulator, but it would require humans to learn a domain-specific syntax and rules to command them. Instead, we want agents that can be instructed just like we would teach other people to do everyday tasks, by providing a description or demonstration of the activity.

Given that we want the instructed agents to perform the activities in VirtualHome, we propose the following task: to generate an activity program, as described in Section 2.3.3, from either a natural language description or a video demonstration. We will be using the work described in Part I to achieve this task. The knowledge base described in Chapter 3, with activity descriptions paired with programs, will serve as a dataset to train a model that translates natural language descriptions into activity programs. These programs can also be executed and rendered in the simulator, allowing us to generate datasets of activity videos, with which we can train

Figure 4-1: We first crowdsource a large knowledge base of household tasks, **(top)**. Each task has a high level name, and a natural language instruction. We then collect "programs" for these tasks, **(middle left)**, where the annotators "translate" the instruction into simple code. These programs can be executed and rendered in our simulator, allowing to generate video demonstrations of the activities in our knowledge base **(bottom)**. The videos and descriptions, paired with programs, can serve as a training dataset, allowing to generate programs from human instructions or demonstrations **(blue and orange arrows)**. These programs can be further executed in tested in the simulator.

models that generate programs from video demonstrations. Finally, the programs predicted by our model can be executed again in the simulator, allowing to test the agent abilities to perform the instructed tasks. Figure 4-1 shows an overview of this pipeline.

In this chapter, we introduce a model to translate task descriptions or demonstrations into activity programs. We leverage the *ActivityPrograms* dataset and a procedurally generated dataset, which we call *SyntheticPrograms* to train the model. Finally, we analyze the generated programs when executed in new environments in VirtualHome.

4.2 Related Work

Language-driven agents: There has been an increasing interest in building language-driven agents, many of the efforts presenter here have been posterior to our work. A number of works have leveraged language to build instruction-following agents that could perform manipulation [88] or navigation [89, 155, 141, 140]. Similar to our goal, to new simulators and datasets has spun work to build agents that could perform high-level activities [127]. Most related to our work, [81, 60] use pretrained large language models to drive agents to do tasks in VirtualHome. In [81], authors fine-tune a language model on programs generated procedurally via a planning-based agent, whereas [60] uses a frozen language model with few-shot prompting from the ActivityPrograms dataset (Chapter 3). Since activities in VirtualHome contain commonsense information, large scale natural language datasets provide a useful source of information for agents to perform these tasks, even if the syntax differs from the programs in VirtualHome.

Learning from videos: Video demonstrations have also been a source to teach agents or robots to perform different kinds of tasks such as object manipulation [98, 40, 33, 124] or imitating human or animal motions [108, 107], However, in many cases they focus on low-level tasks. In [165], the authors detect objects and actions in cooking videos and generate an "action plan" using a probabilistic grammar that can drive robots to perform complex tasks, and [169] uses YouTube cooking videos to learn higher-level collaborative plans. In our work we study high-level tasks in a wider range of domains within the household.

Code synthesis: Given that we represent activities via programs, our task here can be seen as that of language-driven program synthesis, where the goal is to generate code that solves a problem from a natural language specification. Program synthesis has been a long studied research topic [50, 91, 138], where the emergence of deep learning has allowed to make significant progress. Following the development of large language models [21], recent works have achieved great success in program generation [82, 30, 10], reaching human performance. Previous work [66, 92] has also

generated code to answer visual questions about images. Our work differs in the domain, and works with text or video as input.

4.3 Method

We aim to generate a program for the activity from either a natural language description or from a video demonstration. Note that once a script is generated, an agent could perform the task it describes in a different environment, using new 3D models and planning paths.

We aim to generate programs that are 1) consistent with the input description or video and 2) executable in VirtualHome, avoiding actions such as standing up before having sit or closing a fridge that is already closed. A script fulfills the second criteria when it is fully executable in the environment. We measure the first criteria by calculating the normalized longest common subsequence (LCS) with the ground-truth script

$$\frac{|LCS(Prog_{gt}, Prog_{pred})|}{max(|Prog_{gt}|, |Prog_{pred}|)} \tag{4.1}$$

We simplify our programs and remove the instance number for the task of prediction, our scripts are now a sequence of steps of the form:

$$step_t = [action_t] < object_{t,1} > ... < object_{t,n} > \tag{4.2}$$

Where n can be $0, 1$ or 2 depending on the action. The LCS serves as a measure similar to the intersection over union (IOU): the ground-truth and predicted programs can have gaps between the matching steps, but the order of these steps should be correct.

We treat the task of transcribing an input (description or video) into a program as a translation problem. We adapt the seq2seq model [144] for our task, and train it with Reinforcement Learning to optimize the two objectives.

Our model consists of an RNN encoder that encodes the input sequence into a hidden vector representation, and another RNN acting as a decoder, generating one step of the program at a time. We use the same architecture for both description and

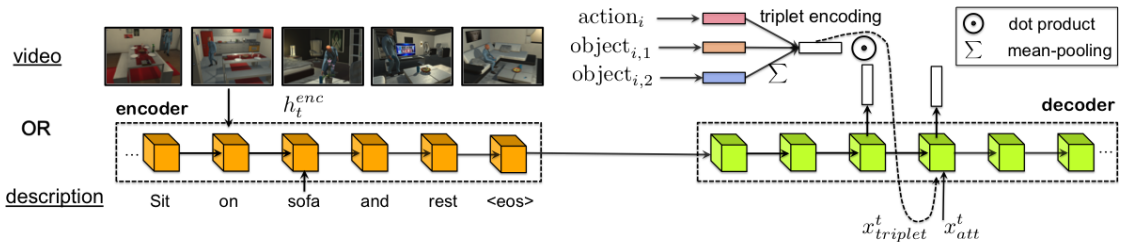


Figure 4-2: Our encoder-decoder LSTM for generating programs from natural language descriptions or videos.

video and assume the input has been transformed to a sequence of vector embeddings. We use a LSTM with 100-dim hidden states as our encoder. At each step t, our RNN decoder decodes a step which takes the form of 4.2. Let $\mathbf{x}_t$ denote an input vector to our RNN decoder at step t, and let h^t be the hidden state. Here, h^t is computed as in the standard LSTM using tanh as the non-linearity. Let a_i be a one-hot encoding of an action i, and o_i a one-hot encoding of an object. We compute the probability p_i^t of an instruction i at step t as:

$$\tilde{a}_i = W_a a_i, \;\; \tilde{o}_{i,n} = W_o o_{i,n}, \;\; v_i = \text{mean}(\tilde{a}_i, \tilde{o}_{i,1}, ..., \tilde{o}_{i,n})$$

$$p_i^t = \text{softmax}_i(\frac{v_i}{\|v_i\|}^T \cdot W_v(h^t \| \mathbf{x}_t^{att})) \tag{4.3}$$

where W_a and W_o and W_v are learnable matrices, and v_i denotes an embedding of an instruction.

The input vector $\mathbf{x}_t$ concatenates multiple features. In particular, we use the embedding v of the step with the highest probability from the previous time instance of the decoder. Following [144], we further use the attention mechanism over the encoder's states to get another feature $\mathbf{x}_t^{att}$. In particular:

$$\alpha_j^t = \text{softmax}_j(v^T \left(W_{att} \left(h^t \| h_{enc}^j\right)\right)) \tag{4.4}$$

$$\mathbf{x}_t^{att} = \sum_j \alpha_j^t h_{enc}^j \tag{4.5}$$

where W_{att}, v are learnable parameters. An overview of the model can be seen in Figure 4-2.

4.3.1 Learning and inference

. Our goal is to generate programs that are both close to the ground-truth programs in terms of their LCS (eq. 4.1) and are also executable by our renderer. To that end, we train our model in two phases. Firstly, we pre-train the model using cross-entropy loss at each time step of the RNN decoder. We follow the typical training strategy where we make a prediction at each time instance but feed in the ground-truth step to the next time instance. We use the word2vec [95] embeddings for matrices W_a and W_o and fix their weights.

In the second stage, we treat program generation as an Reinforcement Learning problem, where the agent is learning a policy that generates steps to compose a program. We follow [116], and use policy gradient optimization to train the model, using the greedy policy as the baseline estimator. In particular, given the policy p_θ defined in eq.4.3, we define the sequence generated by the greedy policy as $\hat{w} = \{\hat{w_1}, ..., \hat{w_T}\}$ with:

$$\hat{w_t} = \arg\max_{w_t} p_\theta(w^t) \tag{4.6}$$

And the sequence w^s generated by sampling from the policy p_θ. Our reward becomes $r(w^s, g) - r(\hat{w}, g)$. Given that the baseline $r(\hat{w}, g)$ does not depend on the sampled w^s, we have that

$$\mathbb{E}_{w^s \sim p_\theta}[r(\hat{w}, g)\nabla_\theta \log p_\theta(w^s)] = r(\hat{w}, g)\sum_{w^s} \nabla_\theta p_\theta(w^s) = r(\hat{w}, g)\nabla_\theta \sum_{w^s} p_\theta(w^s) = 0 \tag{4.7}$$

And therefore

$$-\mathbb{E}_{w^s \sim p_\theta}[(r(w^s, g) - r(\hat{w}, g))\nabla_\theta \log p_\theta(w^s)] = -\mathbb{E}_{w^s \sim p_\theta}[(r(w^s, g)\nabla_\theta \log p_\theta(w^s)] \tag{4.8}$$

Which corresponds to the expected gradient of the loss using REINFORCE [158]. This means that using the baseline will not affect the expected gradient, while allowing to reduce its variance.

We exploit two different kinds of rewards $r(w^s, g)$ for RL training, where w^s denotes the sampled program, and g the ground-truth program. To ensure that the generated program is semantically correct (follows the description/video), we use the normalized LCS metric (eq. 4.1) as our first reward $r_{LCS}(w^s, g)$. The second reward comes from our simulator, and measures whether the generated program is executable or not. This reward, $r_{sim}(w^s)$, is a simple binary value. We start by training using the LCS reward alone, and fine-tune the best model using a balance of the two rewards, as $r(w^s, g) = r_{LCS}(w^s, g) + 0.1 \cdot r_{sim}(w^s)$.

So far we did not describe the input to the RNN encoder. Our model accepts either a language description or a video depicting the action, which are converted into a sequence of vector embeddings to serve as input to the encoder shown in Figure 4-2. We explain in the following sections how the embeddings are obtained for each modality.

4.3.2 Textual Description

To encode a textual description, we split the description by its words, remove punctuation symbols and use word2vec [95] embedding trained on GoogleNews to encode each word.

4.3.3 Video

To generate programs from videos, we partition each video into 2-second clips, corresponding to 9 frames and encode each of the clips as the embedding of the instruction happening at the middle frame. These embeddings serve as input to our RNN encoder. Notice that, some actions (e.g. *walk*) take longer than others (e.g. *switch on*). In order to avoid having an imbalance in the number of clips for the long actions, we allow a maximum of 5 clips for every step in the video.

To obtain the embedding of each clip, we use the Temporal Relation Network [173] with 4-frame relations to predict the embedding of an instruction (action+object+object). Given, the embedding, we obtain the probability of every instruction through eq. 4.2

Dataset	# prog.	avg # steps	avg # sent.	avg # words
ActivityPrograms	2821	11.6	3.2	21.9
SyntheticPrograms	5193	9.6	3.4	20.5

Table 4.1: We analyze programs and natural language descriptions for both real activities in *ActivityPrograms* and *SyntheticPrograms* (2) with procedural programs but real descriptions.

and train the network using cross-entropy loss.

We still have to specify what is the input of the TRN model. In order to make the model potentially generalizable to real videos, we use the semantic segmentation mask, which we obtain by training a DilatedNet [168] segmentation network using the ground-truth segmentations.

4.4 Dataset

The VirtualHome *ActivityPrograms* Dataset, introduced in Chapter 3, contains activity programs paired with natural language descriptions, allowing us to train the text-based program generation model. In order to train the video-based model, we need videos paired with programs, which we can generate through the simulator. Since the programs here represent real activities that happen in households, they contain significant variability in actions and objects that appear in steps. While our ultimate aim is to be able to animate all these actions in our simulator, the simulator only supports the 14 most common actions appearing in the dataset[1]. Therefore, we create another dataset, which we call *SyntheticPrograms*, with programs containing only these actions in their steps. The creation of this dataset is explained below.

We synthesized 5,193 programs using a simple probabilistic grammar encoding activities with actions supported by VirtualHome, such as *watch TV*, *work on computer* or moving objects. For each program, we asked a human annotator to describe it in natural language. Although these programs were not given by annotators, they produced reasonable activities, creating a much larger dataset of paired descriptions-

[1]The actions supported by the simulator are listed in Table 2.2. From those, the actions *Walk Towards*, *Walk Forward*, *Turn Left*, *Turn Right* do not appear in the *ActivityPrograms* dataset.

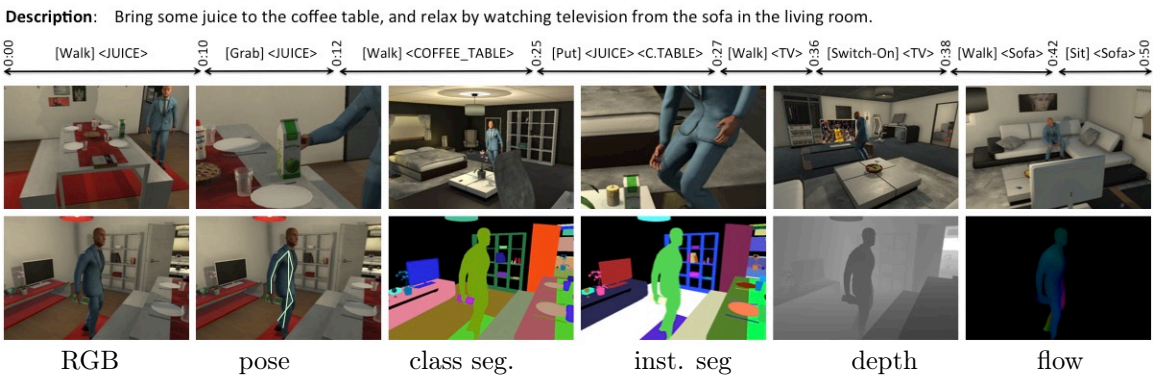


Figure 4-3: A sample of the **VirtualHome SyntheticPrograms Dataset**. The program is generating using a probabilistic grammar, and described by a human annotator in natural language **(top row)**. Then, it is animated in VirtualHome by randomizing the selection of homes, agents, cameras, as well as the placement of a subset of the objects, the initial location of the agent, the speed of the actions, and choice of objects for interactions. Videos have ground-truth: **(second row)** time-stamp for each atomic action, **(bottom)** 2D and 3D pose, class and object instance segmentation, depth and optical flow.

programs at a fraction of the cost. The dataset statistic are shown in Table 4.1, together with the *ActivityPrograms* Dataset, for comparison. Note the similarity in both datasets, both in number of instructions and length of the natural language descriptions.

To generate the activity videos, we execute each of the programs in *video mode*, as described in Section 2.4.1. First, we select an agent and an apartment to perform the activity. We use apartments 1-4 (see Figure 2-2) for videos in the training set, apartment 5 for the validation set, and apartments 1-6 testing our video-to-script model. We then populate the apartment with objects, so that the program can be executed. Then, we place 6-9 static cameras in each room, 26 per home on average, allowing a third person view of the action. During recording, we switch between cameras based on agent's visibility. The resulting dataset contains synthetic activity programs paired with human descriptions and videos with time-stamps of the atomic actions, and different visual modalities. Figure 4-3 shows a sample of the dataset.

4.5 Experiments

In our experiments we exploit both of our datasets: *ActivityPrograms* containing descriptions and programs for real activities, and *SyntheticPrograms dataset* that contains synthesized programs, yet natural descriptions to describe them. *SyntheticPrograms dataset* further contains videos animating the programs.

4.5.1 Instruction Classification from Video

We first evaluate our model for the task of video-based action and action-object-subject (step in the program) classification. Here, we partition each video in 2-sec clips, and use the clip-based TRN on the predicted segmentation mask to perform classification. We compute performance as the mean per-class accuracy across all 2-sec clips in *test*. We consider accuracy at the level of *action*, *object*, or the whole *instruction*. To better understand the generalization properties of the video-based models, we further divide the *test* set into videos recorded in *homes seen* at train time, and videos in *homes not seen* at train time. We report the results in Table 4.2. To set the lower bound, we also report a simple *random retrieval* baseline, in which a program is randomly retrieved from the training set. In Figure 4-4 we show the confusion matrix for action classification. Notice that the model is biased towards the *walk* action but that is also the most common action in the programs, and we empirically found out that this setting provided better script generation results. The model struggles to differentiate between *switch on* and *switch off*, which makes sense given that this information is generally not encoded in the semantic segmentation. We can also see that the model mistakes Watch with *sit/stand up*, which is due to the fact that the video is clipped without taking into account action boundaries. Given that *watch* is normally preceded by a *sitting action* (e.g. for *watch tv*) followed by *standing up*, it is likely that some shots fall in the intersection between these actions.

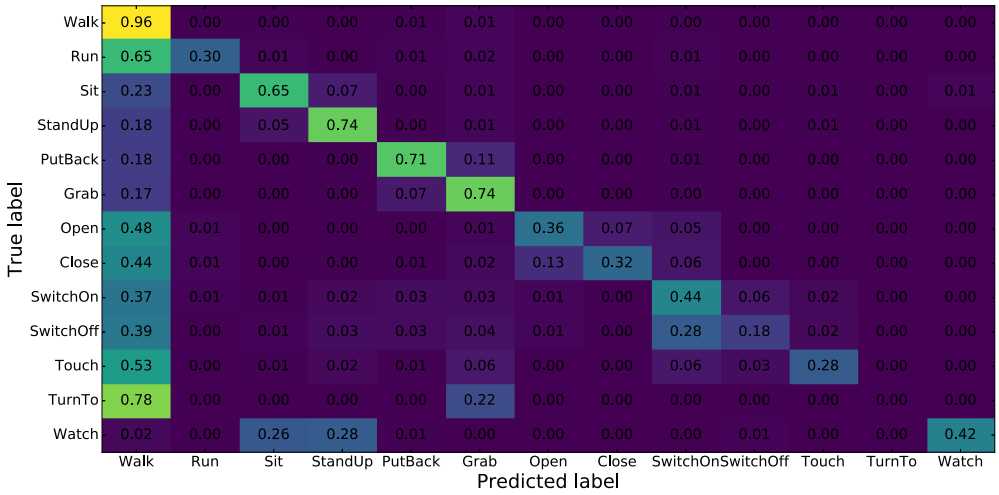


Figure 4-4: Confusion matrix for action classification in 2-sec clips.

	Action	Objects	Steps	Mean
Rand. Retrieval	8.30%	1.50%	0.51%	3.43%
Seen homes	70.32 %	42.14 %	23.81 %	45.42%
Unseen homes	31.34%	14.55%	11.48%	19.12%
All	46.85%	25.76%	18.41%	30.34%

Table 4.2: Accuracy of *video-based action classification* and *action-subject-object* (step in the program) prediction in **2-sec clips** from our VirtualHome Activity dataset.

Effect of input

We also study the effect of using ground-truth segmentation or RGB images in the clip classification task. The results are shown in table 4.3. As it can be seen, there is a significant performance drop from ground-truth segmentation to the predicted one, which suggests that more effort should be put in training the segmentation. Many of the interactions in VirtualHome are done with small objects, with which segmentation networks typically struggle with. The model using RGB images performs better than the one using the predicted segmentations, but drops significantly in performance when testing on apartments unseen during training, which suggests it would not fit for transferring to unseen environments. Given that our future goal is to have this models working on real environments, we resort to using semantic segmentation information for our video-based prediction task.

	Action	Objects	Steps	Mean
Seen homes	78.06 %	73.00 %	56.42 %	69.16%
Unseen homes	54.89%	52.94%	43.50%	50.44%
All	62.58%	61.18%	48.49%	57.42%

(a)

Action	Objects	Steps	Mean
77.46 %	60.21 %	44.45 %	60.71%
40.56%	26.56%	26.46%	31.19%
54.60%	40.44%	35.17%	43.40%

(b)

Table 4.3: Accuracy of *video-based action classification* and *action-subject-object* (step in the program) prediction in **2-sec clips** using ground-truth segmentation (a) and RGB images (b).

4.5.2 Program Generation

We now evaluate the task of program generation. We evaluate program induction using the normalized LCS, as described in eq. 4.1. We also compute accuracies for *actions* and *objects* alone. Since LCS does not measure whether the program is valid, we report another metric that computes the percentage of generated programs that are executable in our simulator.

Language-based prediction

Since we have descriptions for all activities, we first evaluate how well our model translates natural language descriptions into programs. We report results on *ActivityPrograms* (real activities), as well as on *VirtualHome Activity* datasets (where we first only consider descriptions, not videos). We compare our models to four baselines: 1) *random sampling*, where we randomly pick both an action for each step and its arguments, 2) *random retrieval*, where we randomly pick a program from the training set, 3) *skipthoughts*, where we embed the description using [72, 176], retrieve the closest description from training set and take its program, 4) our model trained with MLE (no RL). Table 4.5 and 4.4 shows the results. Note that the retrieval baselines (*skipthoughts* and *random retrieval*) are always executable, since our training set scripts were generated to be executable. We can see that our model outperforms all baselines on both datasets. Our RL model that exploits LCS outperforms the MLE model on both metrics (LCS and executability). Our model that uses both rewards slightly decreases the LCS score, but significantly improves the executability metrics. Figure 4-5 shows some example results for *ActivityPrograms*, trained using the

Method	Action	Objects	Steps	Mean	Simulator (%)
Rand. Sampling	.226	.039	.020	.095	0.6%
Rand. Retrieval	.473	.079	.071	.207	100.0%
Skipthoughts	.642	.272	.252	.389	100.0%
MLE	.777	.723	.686	.729	38.6%
PG(LCS)	.803	.766	.732	.767	35.5%
PG(LCS+Sim)	.806	.775	.740	.774	39.8%

Table 4.4: Programs from description: Accuracy on *SyntheticPrograms*. We evaluate using the normalized longest common subsequence, mimicking IoU for programs, as well as the percentage of scripts executable in the simulator.

Method	Action	Objects	Steps	Mean
Rand. Sampling	.106	.018	.004	.043
Rand. Retrieval	.320	.037	.032	.130
Skipthoughts	.469	.297	.266	.344
MLE	.497	.392	.340	.410
PG(LCS)	.522	.433	.387	.447

Table 4.5: Programs from description: Accuracy on *ActivityPrograms*. Since real programs are mostly not executable in our simulator due to the lack of implemented actions, we cannot report the executability metric or use it as a reward.

LCS reward and Figure4-6 shows results on *SyntheticPrograms*, trained using both rewards.

Video-based prediction

We also report results on the most challenging task of video-based program generation. The results are shown in Table 4.6. One can observe that RL training with LCS reward improves the overall accuracy over the MLE model (the generated programs are more meaningful given the description/video), however its executability score decreases. This is expected: MLE model typically generates shorter programs, which are thus more likely to be executable (an empty program is always executable). A careful balance of both metrics is necessary. RL with both the LCS and the simulator reward improves both LCS and the executability metrics over the LCS-only model. Figure 4-7 shows example results for script generation from video on *SyntheticPrograms* test set.

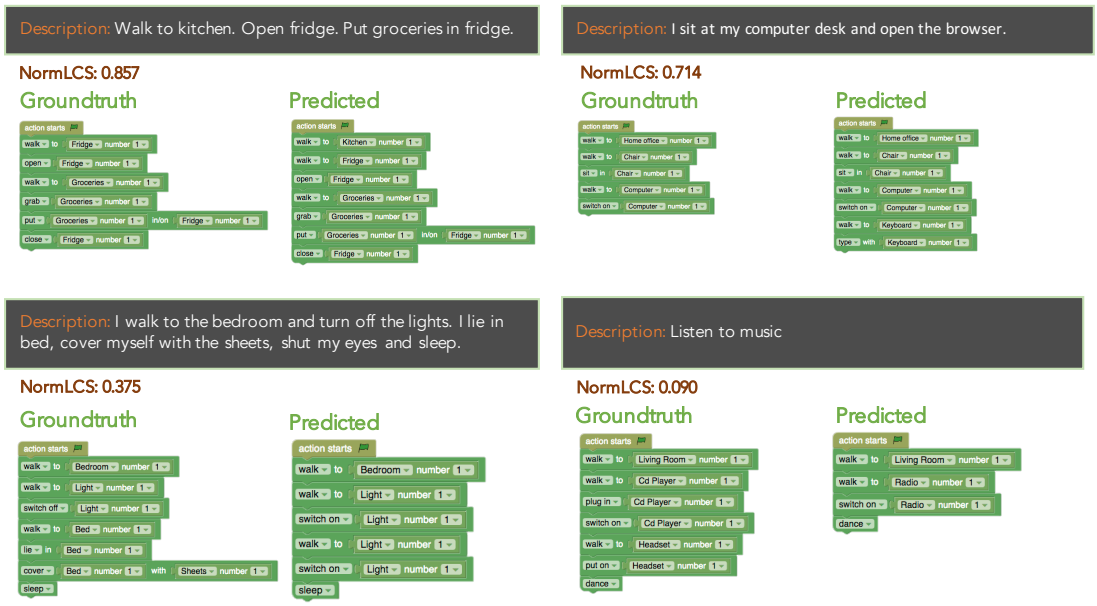


Figure 4-5: Example results for language-based prediction on *ActivityPrograms* dataset.

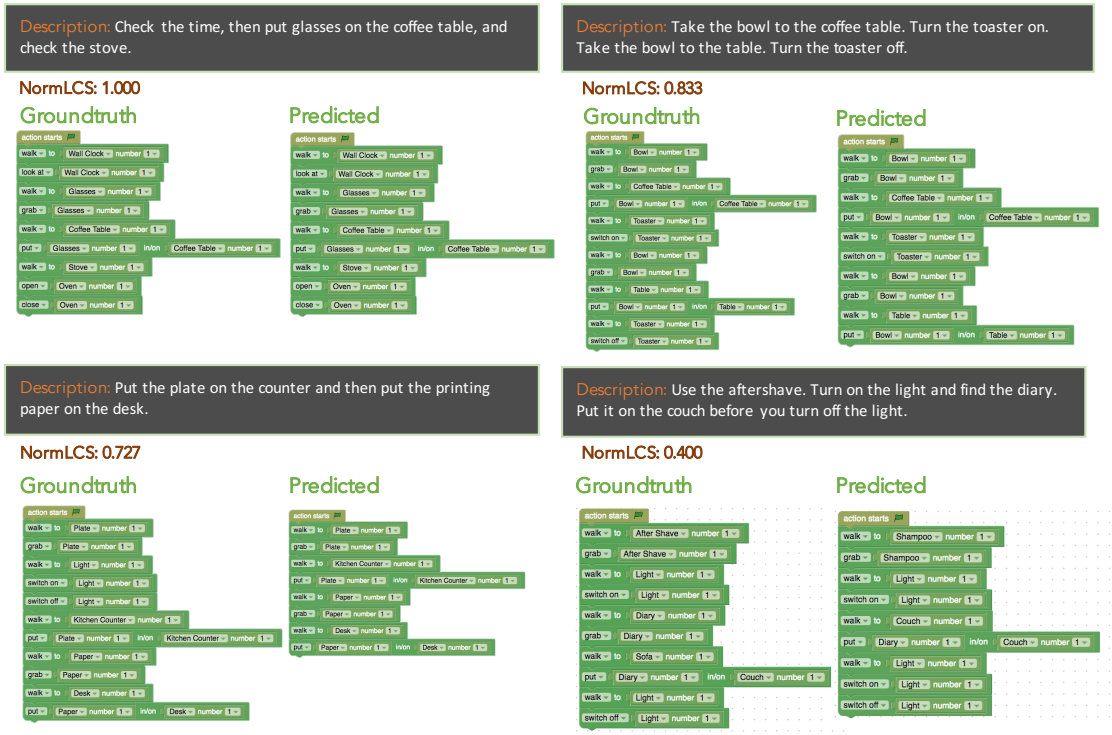


Figure 4-6: Example results for language-based prediction on *SyntheticPrograms* dataset.

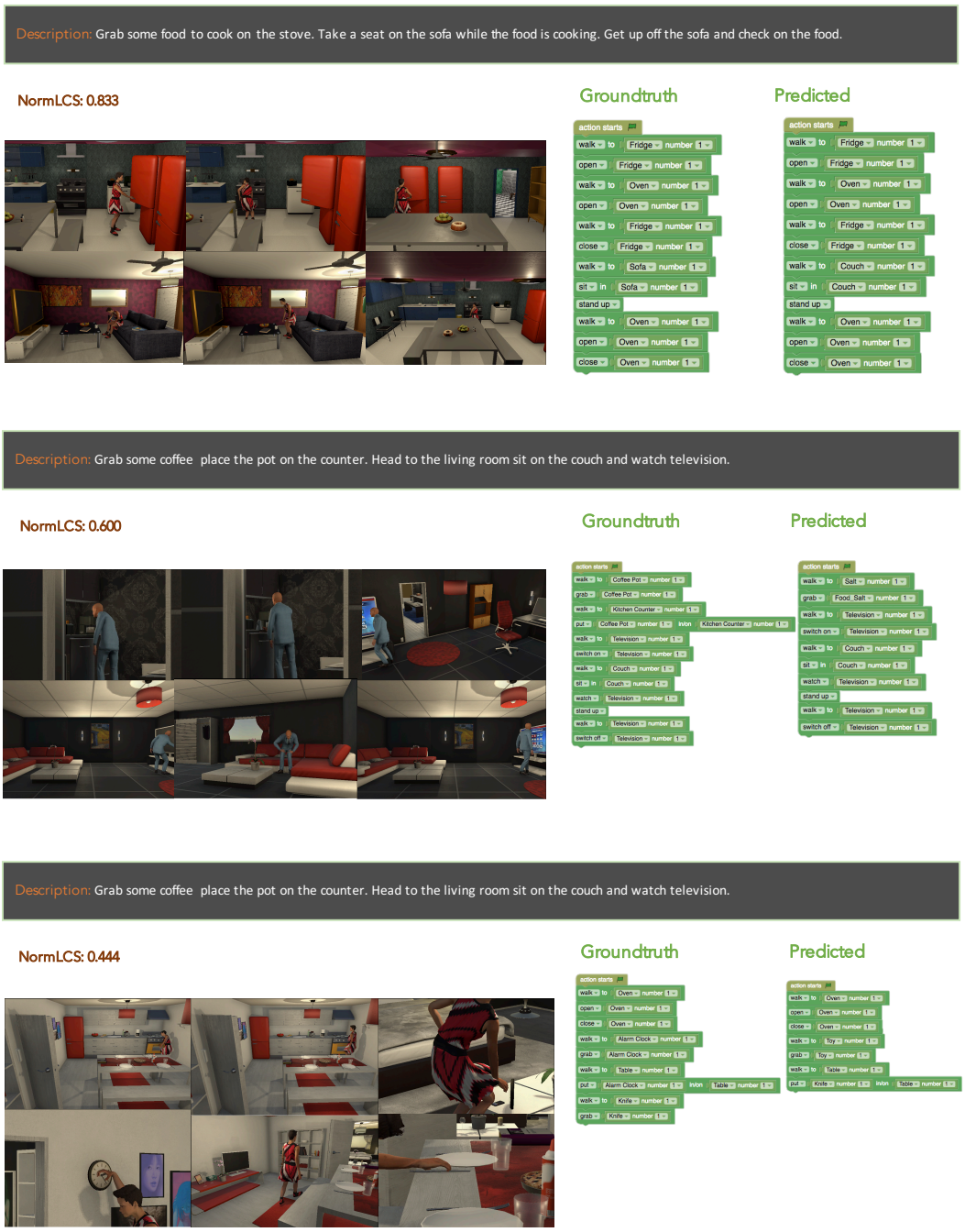


Figure 4-7: Example results for video-based prediction on *SyntheticPrograms* dataset.

	Action	Objects	Steps	Mean	Simulator
Rand. Retrieval	.473	.079	.071	.207	100.0%
MLE	.735	.359	.341	.478	19.4%
PG(LCS)	.761	.383	.364	.502	19.0%
PG(LCS+Sim)	.751	.377	.358	.495	22.4%
PG(LCS+Sim) Seen homes	.851	.556	.528	.645	24.6%
PG(LCS+Sim) Unseen homes	.680	.250	.236	.389	20.9%

Table 4.6: Video-based program generation.

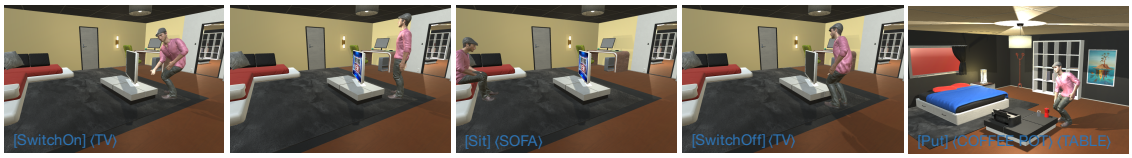
Executing programs in VirtualHome

. Given that we are optimizing our programs to be executable in VirtualHome, we can try running the script predictions to generate new videos. In Figure 4-8 we show a few examples of our agent executing programs generated from natural descriptions. To understand the quality of our simulator as well as the plausibility of our program evaluation metrics, we perform a human study. We randomly selected 10 examples per level of performance: (a) $[0.95-1]$, (b) $[0.8-0.95]$, (c) $[0.65-0.8]$, and (d) $[0.5-0.65]$. For each example we had 5 AMT workers judge the quality of the performed activity in our simulator, given its language description. Results are shown in Figure 4-9. One can notice agreement between our metrics and human scores. Generally, at perfect performance the simulations got high human scores, however, there are examples where this was not the case. This may be due to imperfect animation or planning in our simulator, or the fact that the generated scripts do not have information about object instances. Future models should be able to incorporate such information in the generation of scripts.

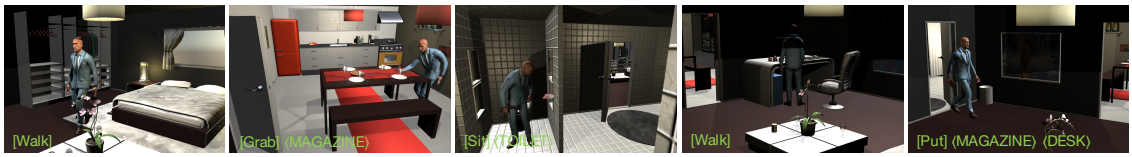
Implications

The high performance of text-based activity animation opens exciting possibilities for the future. It would allow us to replace the more rigid program synthesis that we used to create our dataset, by having annotators create these animations directly via natural language. Similarly, the text-based program generation could allow to reduce human annotation by crowd-sourcing scripts for activities from existing text corpora. We leave this as an avenue for future work.

Description: Get an empty glass. Take mil from refrigerator and open it. Pour mil into glass.



Description: Go watch TV on the couch. Turn the TV off and grab the coffee pot. Put the coffee pot on the table and go turn the light on.



Description: Look at the clock then get the magazine ands use the toilet. When done put the magazine on the table.

Description: Take the face soap to the kitchen counter and place it there. Turn toaster on and then switch it off. Place the pot on the stove.

Figure 4-8: Videos generated from descriptions in *SyntheticPrograms*. We first use the language-based model to predict programs from the given descriptions. Then, we execute and render these programs in a new environment.

4.6 Discussion

In Part I of this thesis, we introduced a simulator to represent household activities, and a knowledge base with activity descriptions and programs, allowing to perform these activities in the simulator. In this chapter, we closed the loop and proposed a simple model that infers an activity program from either a video or a textual description, allowing agents to be ‘’driven" by naive users via natural language or a video demonstration. We then showed examples of agents performing these programs in our simulator.

While these efforts bring us closer to the goal of building robots that can assist humans in household environments, the current approach presents several limitations. VirtualHome does not provide a perfect representation of a real environment, and therefore it will be challenging to transfer what is learned in simulation to the real

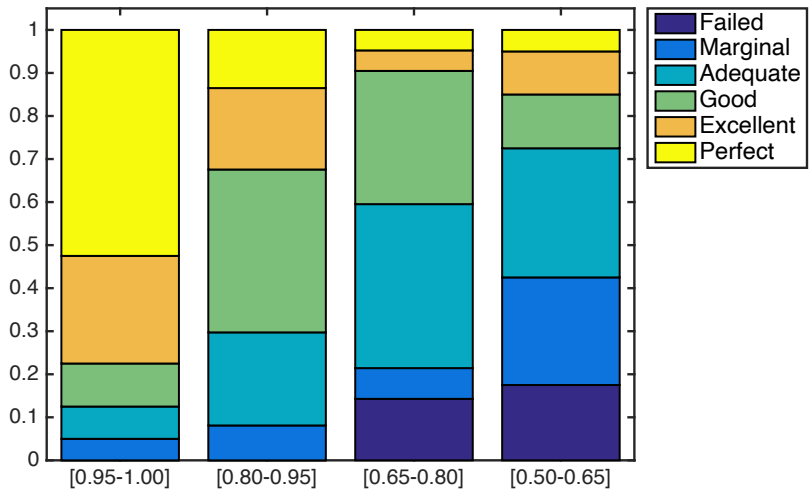


Figure 4-9: Human judgement of videos generated from text descriptions.

world. This thesis focuses on building assistive agents via simulation and leave this for future work. In this chapter, we assumed that humans just provide task specifications to the agents, but that they are not present in the environment as these agents perform the task. This means that the agent only has to focus on interacting in the environment, without accounting for the fact that there may be other humans in it. We will be addressing this in Part III. The approach presented in this chapter allowed us to have a single agent performing instructed tasks in the environment, but it had an important assumption: that the environment where the agent would perform the task was consistent with the description of the task to perform, or with the environment where the human had previously demonstrated the task. When the agent sees a human going to the cabinet and getting coffee grounds to make a coffee, it will reproduce the same steps to perform the task. However, if the new environment has the coffee grounds in the freezer, the agent will still go to the cabinet, failing to perform the instructed task.

Thus, we aim to build agents that not only follow human instructions, but that they ground them in the environment where they will perform the task. This will allow us to build agents that can perform tasks in a wide set of environments and conditions, even when they differ from the conditions where the task was instructed. In the next chapter, we present an approach to build such agents.

Chapter 5

Environment Aware Agents via Activity Sketches

5.1 Introduction

We want agents to be able to perform everyday tasks such as setting up the table, preparing coffee, or even sit on the couch and watch TV. An agent should learn to perform these tasks from high-level descriptions or visual demonstrations. We presented in Chapter 4 an approach to do that. The challenge here is on how to generalize the acquired knowledge to new environments. For instance, if we want to learn to make coffee, an agent could first watch a video of someone making coffee in order to extract the sequence of steps (program) that need to be executed. However, when trying to make coffee in an environment that differs from the environment in which the demonstration took place, the agent has to adjust the program so that it can be executed. For instance, the agent could find that the coffee machine is unplugged, or that it is in the living room instead of in the kitchen, or that someone else is currently using it and the agent needs to wait. Being able to perform these adjustments requires access to a common-sense knowledge database that allows the agent to decide which steps in the demonstration are essential in the task definition, and how the program needs to be modified (by adding/removing steps) in order to accomplish the task in a new environment.

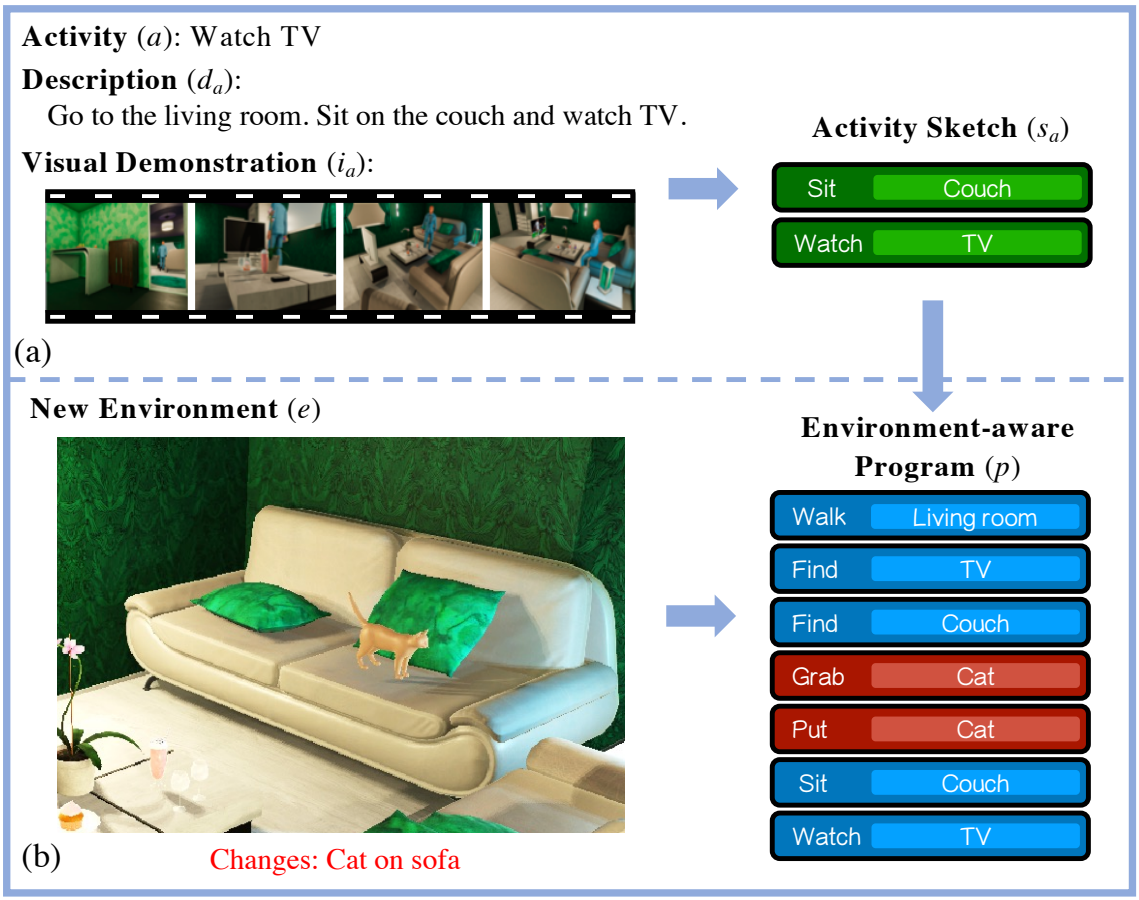


Figure 5-1: Overview of the *environment-aware program generation*. Our goal is (a) generating a sketch s_a distilling the essential steps of the given demonstration i_a or description d_a and (b) given a new environment e, generating a program p, adapting the sketch to e. The program contains the instructions to perform the activity (blue blocks) as well as instructions to deal with the environment (red blocks, grabbing the cat to sit).

To address the generalization problem we represent activities with "sketches", representations inspired by work in programming languages [138, 97]. We define sketches as high-level representations of the steps needed to perform a task but leaving holes that need to be completed for the program to become executable in a particular environment.

Figure 5-1 illustrates our goal. Given a visual demonstration or a description of someone going to watch TV, we want to extract the sketch of the activity (fig. 5-1a). In this example, the sketch consists of two steps: `Sit Couch`, `Watch TV`. The fact

that the demonstration had the person going to the living-room is not important. Watching TV requires us to go to the room that contains the TV. Executing this activity in a new environment requires expanding the sketch into a complete program (fig. 5-1b). The sketch expansion depends on the state of the environment. In this example, it turns out that there is a cat on the couch. To sit on the couch, we need to push away the cat first. We show the automatically generated program that includes all the steps needed to complete the task and to deal with the cat.

To expand sketches into programs, it is necessary to have access to commonsense knowledge about how to perform daily activities in different environments. The *ActivityPrograms* dataset, presented in Chapter 3, contains around 3k activities with associated programs, but has no information about how these activities can be done in different environments. Therefore, we need to extend the dataset to include a larger set of actions and over 30k programs and collect sketches of the activities via crowd-sourcing. Each environment is represented as a graph with 300 objects and 4000 spatial relations on average.

We then propose a model to generate programs by selecting, for every step, a node in the environment graph representing an object of interaction. To do so, we exploit Graph Neural Networks (GNNs) to reason about the states and relations between the objects in the environments. Furthermore, we propose ResActGraph, a model that reasons about the changes in the graph induced by the agent's previous steps to generate the goal program.

In this chapter, we propose an approach to build *Environment-Aware* agents. For this, we 1) introduce sketches as environment-independent representations of an activity, 2) build a database of commonsense knowledge of activities and sketches, and 3) present a method to generate programs from sketches that accomplish the task in a new environment.

5.2 Related Work

Learning from demonstrations. Learning from visual demonstrations or language descriptions has been of increasing interest in both robotics and computer vision. However, this has been mostly focused on learning low-level tasks rather than the high semantic level activities that we tackle in this work. For example, [147, 93, 6] focus on learning to navigate in environments or manipulate objects form language, while in visual imitation learning, multiple works have used videos to learn to manipulate objects under low supervision regimes [98, 40] or imitate kinetic human behaviors [94]. Our work is more closely related to semantic planning [59, 175, 15], which focuses on modeling sequences of composite and semantically loaded actions. Learning those requires inferring and modeling the sub-goals of a given task. [143] proposes to encode the goals and task constraints via programs , whereas we consider language inputs coupled with environments. Huang et al. [59] represent such goals as a graph, with nodes being actions and edges being precondition states, while we propose program representations. Furthermore, to perform such tasks, it is necessary to know what the constraints of the given environment where it will be executed are. Similarly to [15], we propose to encode knowledge of the environments and how they can constrain the activities to be performed.

Program Synthesis by Sketching. Our approach for environment-aware program generation is partially inspired by performing program synthesis by sketching. In [138], a sketch expresses the high-level structure of an implementation but leaves holes in place of low-level details, which corresponds to our model that derives the details based on the environmental constraints so as to execute the programs smoothly.A recent body of work has developed neural approaches to program generation using user-provided examples [46, 18], visual demonstrations [142], and descriptions [36]. The work [97] is the most related to ours. They learn a model that predicts sketches of programs relevant to a label and the predicted sketches are concretized into code using combinatorial techniques. The main difference between our work and theirs is not only that our sketches are inferred from visual or textual data, but that we focus

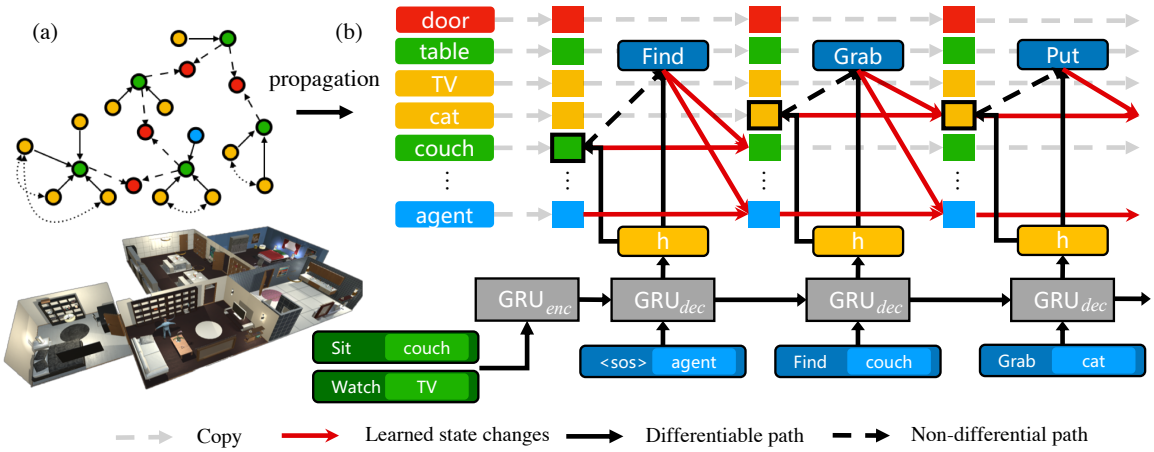


Figure 5-2: (a) We extract the ground truth environment graph from VirtualHome and perform message passing on the graph. (b) At every time step, the decoder perform sequential classification over the hidden states of the graphs (top row). The selected nodes are shown in bold border blocks. We also model the environment changes induced by the generated programs (solid red arrows).

on how to incorporate the environmental constraints in program generation.

5.3 Problem formulation

The goal of environment-aware program generation is to predict, given a demonstration or description of an activity and an environment, a program that can execute the activity in such environment. We define this task and the corresponding notation in this section.

Let A and E be the universe of activities and environments. An activity a (e.g. watch TV) can be represented as a set of programs P_a containing a sequence of instructions (e.g. `TurnOn TV`, `Sit Sofa`, `Watch TV`), which vary depending on how the activity is performed. Let $\mathbb{1}(p, e)$ be an indicator function determining if p can be executed in e (e.g. an agent can not grab cups inside a closed cabinet). Given an activity $a \in A$ and an environment $e \in E$, our goal is to learn a model that generates p such that

$$p \in P_a, \mathbb{1}(p, e) = 1 \tag{5.1}$$

We use the corresponding visual demonstrations $i_a \in I_a$ or descriptions $d_a \in D_a$ to specify a. However, i_a and d_a are implicitly conditioned on certain environments which might differ from the current one e. Therefore, directly inferring $\hat{p}$ does not ensure that $\hat{p}$ satisfies eq. 5.1 given e.

Inspired by [138], we introduce program sketches $s_a \in S$ as *environment-independent* representations of the activities. We thus change the constraint in eq. 5.1 to be:

$$p \in P_{s_a}, \mathbb{1}(p, e) = 1 \tag{5.2}$$

With the program sketches as proxy representations, we can divide the task into two sub-problems: a model that predicts a sketch $\hat{s_a}$ from a demonstration i_a or a description d_a and another model that predicts a program $\hat{p}$ given the predicted sketch $\hat{s_a}$ and an environment e:

$$\begin{aligned} \hat{p} &= f_{sketch2prog}(\hat{s_a}, e), \text{where} \\ \hat{s_a} &= f_{demo2sketch}(i_a) \text{ or } \hat{s_a} = f_{desc2sketch}(d_a) \end{aligned} \tag{5.3}$$

Here, p and s_a are a sequence of instructions. Each instruction is represented by an action and up to two arguments representing objects of interaction $(\alpha, \beta^1, \beta^2)$.[1]

5.4 Method

In this section, we present our approach to the *environment-aware program generation* task. First, we introduce *ResActGraph* to generate programs from sketches and target environments. Later on, we describe how we predict sketches from demonstrations or descriptions.

[1]The number of arguments depends on the type of the action, see Table 2.2 for the list of actions along with the number of arguments.

5.4.1 Program generation from sketches and graphs

We frame the program generation task as a seq2seq problem, where an encoder encodes the input sketch and the decoder generates the target program one instruction at a time, composed by an action and object arguments. Given that the program must be grounded in a target environment, instead of predicting the objects from a fixed taxonomy, the model predicts for each instruction object instances that are present in the environment. This has two benefits: (1) It avoids referring to object instances that do not exist in the environment. (2) It allows the model to use information of each instance within the environment, such as its state or relations with other objects, to predict the appropriate instruction.

To do that, we encode the scene as a graph $G = (\mathcal{V}, \mathcal{R})$ modeling the dependencies of the object instances. The node $v \in \mathcal{V}$ indicates the object instance and each node has a label, including the object class c_v, its states l_v, and properties $prop_v$. Note that $\mathcal{V}$ includes a node for the agent itself. The edge $r \in \mathcal{R}$ encodes the spatial relations, including ON, IN_OBJ, IN_ROOM, CLOSE_TO, and FACE_AT, between every two object instances.

The node labels and relations are used to obtain vector embeddings for each instance which are used by the decoder to predict the environment-aware program, as we describe in the following section.

5.4.2 ResActGraph

We adopt the GGNN [83] framework to obtain the hidden states of the nodes and capture the object relations in the environment graph. The hidden states of each node v are initialized by its label $(c_v, l_v, prop_v)$:

$$h_v^0 = \tanh(g_{init}([W_c c_v, W_l l_v, W_{prop} prop_v]) \tag{5.4}$$

We apply one-hot encoding to the label and set W_c, W_l, W_{prop} as learnable weights. g_{init} is a network composed of fully connected layers that combine all the information.

At propagation step k, each node's incoming information x_v^k is determined by

aggregating the hidden states of its neighbors $v' \in \mathcal{N}(v)$ at the previous step $k-1$:

$$x_v^k = \sum_{j \in L(\mathcal{R})} \sum_{v' \in \mathcal{N}_j(v)} W_{p_j} h_{v'}^{k-1} + b_{p_j} \tag{5.5}$$

$L(\mathcal{R})$ denotes the set of edge labels and the linear layer W_{p_j} and bias b_{p_j} are shared across all nodes.

After aggregating the information, the hidden states of the nodes are updated through a gating mechanism similar to Gated Recurrent Unit (GRU) [31] as follows:

$$\begin{aligned} z_v^k &= \rho(W_z x_v^k + U_z h_v^{k-1} + b_z), \\ r_v^k &= \rho(W_r x_v^k + U_r h_v^{k-1} + b_r), \\ \hat{h}_v^k &= \tanh(W_h x_v^k + U_h(r_v^k \odot h_v^{k-1}) + b_h), \\ h_v^k &= (1 - z_v^k) \odot h_v^{k-1} + z_v^k \odot \hat{h}_v^k \end{aligned} \tag{5.6}$$

This results in a vector embedding for each object h_v^k, with information about its state and relationship with the environment.

We use one GRU to encode the sketches and another one to generate the program one instruction at a time. For time t, let $feat_{s_a} = enc(s_a)$ be the output of the sketch encoder, and h_{dec}^t the hidden states of the decoder. To predict the program instruction, $(\hat{\alpha}_t, \hat{\beta}_t^1, \hat{\beta}_t^2)$, we predict the first object argument β_t^1 over the graph nodes, use it to predict the action $\hat{\alpha}_t$ and combine this information to predict the second argument $\hat{\beta}_t^2$:

$$\begin{aligned} \hat{\beta}_t^1 &= \arg\max_{v \in \mathcal{V}} \sigma(g_{\beta^1}(h_{dec}^t, h_v^K, feat_{s_a})) \\ \hat{\alpha}_t &= \arg\max_{\alpha \in \mathcal{A}} \sigma(g_{\alpha}(h_{dec}^t, h_{\hat{\beta}_t^1}^K, feat_{s_a})) \\ \hat{\beta}_t^2 &= \arg\max_{v \in \mathcal{V}} \sigma(g_{\beta^2}(h_{dec}^t, h_v^K, feat_{s_a}, h_{\hat{\beta}_t^1}^K, \hat{\alpha}_t)) \end{aligned} \tag{5.7}$$

where $\mathcal{A}$ is all the possible actions and σ denotes the `softmax` function.

Note that so far the hidden states of the nodes h_v^K are constant over t, but we would like them to change according to the program being executed. To do that, at time t, we use the previously generated instructions $(\hat{\alpha}_{<t}, \hat{\beta}^1_{<t}, \hat{\beta}^2_{<t})$ to update the hidden states of the nodes. We set the initial state of each node as $h_v^{K_0} = h_v^K$ and update the state $h_v^{K_t}$ at time t if v is interacted by the agent at the previous time step or is the agent itself. For example, if the instruction at the previous time step is `grab mug`, we use the action embeddings of `grab` to change the hidden states of `agent` and `mug`. Let $\hat{v}$ correspond to the agent node or one of the previous arguments $\hat{\beta}^1_{t-1}, \hat{\beta}^2_{t-1}$. The state $h_{\hat{v}}^{K_{t-1}}$ is updated as follows:

$$\begin{aligned} h_{\hat{v}}^{K_t} &= h_{\hat{v}}^{K_{t-1}} + r \\ r &= \tanh(g_{res}(h_{\hat{v}}^{K_{t-1}} \odot g_e(Emb(\hat{\alpha}_{t-1}), m_{t-1}))) \end{aligned} \tag{5.8}$$

where $Emb(\hat{\alpha}_{t-1})$ is the embedding of $\hat{\alpha}_{t-1}$ and m_{t-1} is a one-hot encoding denoting if v is the subject or the object of $\hat{\alpha}_{t-1}$. g_e and g_{res} consider the change of the $h_{\hat{v}}^{K_{t-1}}$ and predict the residuals. Note that since the node `agent` is involved at every time step, it tracks the progress through the generation. The model overview is shown in Figure 5-2.

Learning. We use the cross-entropy loss function for program prediction. The GGNN and GRUs are then trained with the back-propagation through time (BPTT).

5.4.3 Inferring sketches

Activities specified by demonstrations. We use key frames $i = [i_n]_{n=1:N_{demo}}$ as the representations of the demonstrations, where N_{demo} is the length of the key frames. To be specific, we take the bird-eye view of each i_n. Besides, we also use the ground truth semantic segmentation map $i_{seg} = [i_{seg_n}]_{n=1:N_{demo}}$ as input. Two CNNs are used to extract the features separately, and we apply late fusion to extract the n^{th} visual features $feat_n$ as follows:

$$feat_n = g_{fuse}([CNN_i(i_n), CNN_{seg}(i_{seg_n})]) \tag{5.9}$$

where [,] denotes concatenation. Later on, we max-pool the features over the different steps time steps and apply a GRU to decode the sketches.

Activities specified by descriptions. We adopt the seq2seq model with a GRU encoding each word in the description and a GRU to decode each of the sketch instructions.

5.5 Dataset

Our goal is to generate a sketch s_a from a demonstration i_a or description d_a and induce a program p from s_a and a target environment e. To train a model that can do this, we need a dataset containing activities, paired with demonstrations and descriptions, sketches, programs and the environment where the program will be executed. In Chapter 3 we introduced *ActivityPrograms*, a crowd-sourced dataset with typical daily activities along with their descriptions and programs. We describe here how we can augment this dataset to allow training environment-aware agents.

5.5.1 Collecting sketches

When collecting each activity in VirtualHome, annotators imagine an environment where the activity could take place and provide a description according to it. As a result, the description and subsequent program is specific to a given environment, but may not be doable when presented with new environments or constraints (see Figure 5-3). Therefore, we need a more abstract representation of an activity, which can be consistent with multiple environments as well as the original program and description.

Inspired by [138], we collect the sketches of the activities to abstract out the components that are environment-dependent and informally define the sketches as the *environment independent representations*. Different from programming languages,

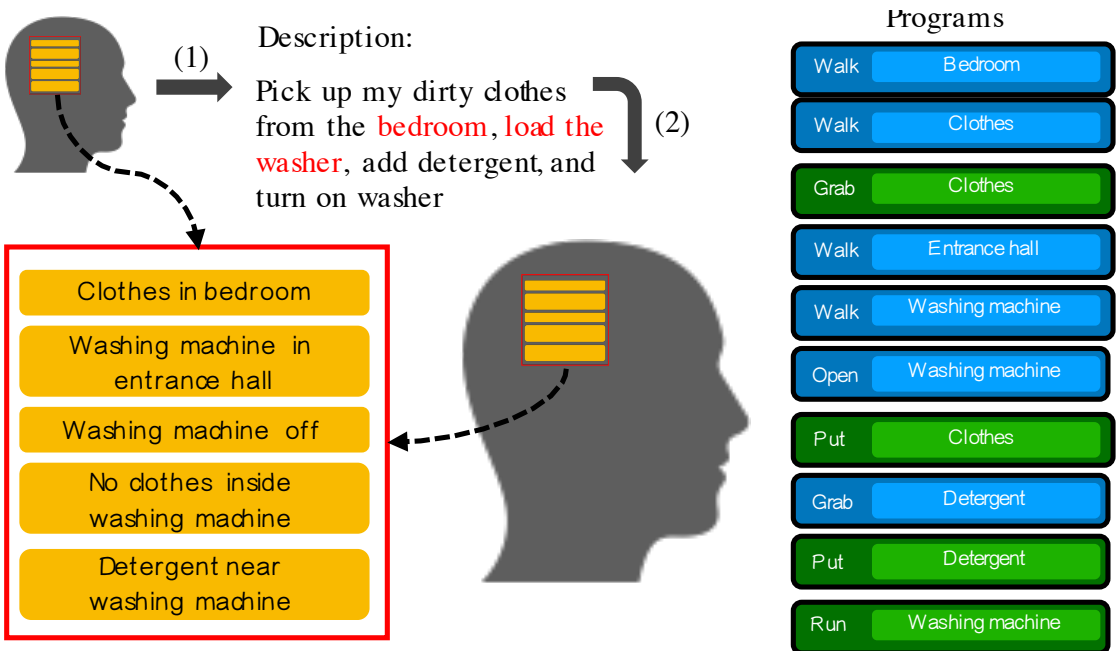


Figure 5-3: The annotators label the descriptions and programs with certain environments in mind (bottom left), resulting in the *environment-dependent* descriptions and programs. The blocks colored in blue are considered as environment-dependent components.

it is highly non-trivial to define the sketches of the activities since they depend on the commonsense of each individual. Therefore, we manually collect the sketches for each program, obtained a tuple (a, i_a, s_a, p) or activities, descriptions, sketches and programs.

5.5.2 Pairing programs with environments

We finally need to add an environment that pairs with the activity programs. Since we do not know the environment that each annotator had in mind, we need to infer it from the program. We first extract the preconditions of the programs and use them to sample feasible environments. We define preconditions of a program as the conditions that have to be true in the environment in order to execute the program in it. For example, in order to execute `Watch TV`, the tv should be on. We construct a function Φ that infers the preconditions from p given a set of rules, and sample e from the set of environments in Figure 2-2, updating it so that it satisfies such preconditions

$E_{\Phi(p)}$:

$$e \sim E_{\Phi(p)} \subset E, \text{ s.t. } \forall j \in \Phi(p), e \text{ satisfies } j \tag{5.10}$$

The above is a weak constraint over the environments, since it does provide information about objects that are not specified in the preconditions $\Phi(p)$. To get realistic environments, these objects should follow some priors. For example, couches can be occupied, but they can not be cold. Apples can be stored inside fridges, but they are seldom found in bathtubs. Thus, we build *KB-RealEnv* , a knowledge base of common-sense rules of object relationships, and use it to set the environment where the task will be executed. In particular, we 1) select environment e that satisfies the inferred pre-conditions, 2) pick objects that appear in the program but which location is not specified by a precondition, as well as an extra set of random objects to make the environment more diverse, and 3) add these objects into the environment, making sure they satisfy the rules in *KB-RealEnv*. The knowledge base can be seen on Table A.2 and A.3.

5.5.3 Generating demonstrations

We can now obtain demonstrations for the activities in our dataset by executing each activity program in the matching environment. Here, instead of letting VirtualHome perform the mapping between objects in the program and the ones in the environment, as we did in Section 4.4, we perform the assignment before executing the program, at the time of generating the matching environment. This ensures that the objects in the program match with objects that satisfy the preconditions.

5.5.4 Extending programs to diverse environments

We now have the information (a, i_a, d_a, s_a, e, p). However, relying solely on the original programs results in a limited set of preconditions and thus environments. One possible reason is that when describing activities, annotators tend to assume the simplest setting to perform the given activity. For example, when thinking of doing the

	VirtualHome	Executable	Augmented	*VirtualHome-Env*
# programs	2807	1387	27284	32761
Average length	13.26	9.61	18.58	18.79
Average # nodes	-	289	290	289
Average # edges	-	4393	4305	4319

Table 5.1: The statistics of *VirtualHome-Env*. The fact that some programs in *VirtualHome* are noisy results in less number of executable programs.

laundry, it is common to imagine that the washing machine is idle or empty. To address that, we use the VirtualHome-Symbolic simulator (described in Section 2.5), that takes a program p and environment graph e and outputs the graph corresponding to the environment after executing the program, and extend it to raise an exception if the program is not executable at a certain step. We call this simulator Ψ . Given a program with preconditions $\Phi(p)$, we start by randomly perturbing them into $\Phi(p)'$ and use eq. 5.10 to obtain e' as an environment satisfying $\Phi(p)'$. Then, we execute p in the simulator with the environment e'. Given that the environment and preconditions have changed but the program is still the same, as the program is executed, some exception will be raised from the simulator. Then, a subroutine is called to modify the program p into p', by inserting or removing instructions to correct the exception, obtaining the extended program.

For example, when we executing `Sit Sofa`, if `Sofa` is occupied, the subroutine is expected to perform actions to remove things on the `Sofa` until there is enough space to `Sit`. We manually compose the subroutines based on different types of exception, forming the knowledge base, *KB-ExceptionHandler*. We show more details of how we augment the programs in the supplementary materials.

This way, we augment over 30k tuples of sketches, environments, and programs (s_a, e', p'). Note that the sketches s_a are environment-independent, so there is no need to change them after applying the subroutines.

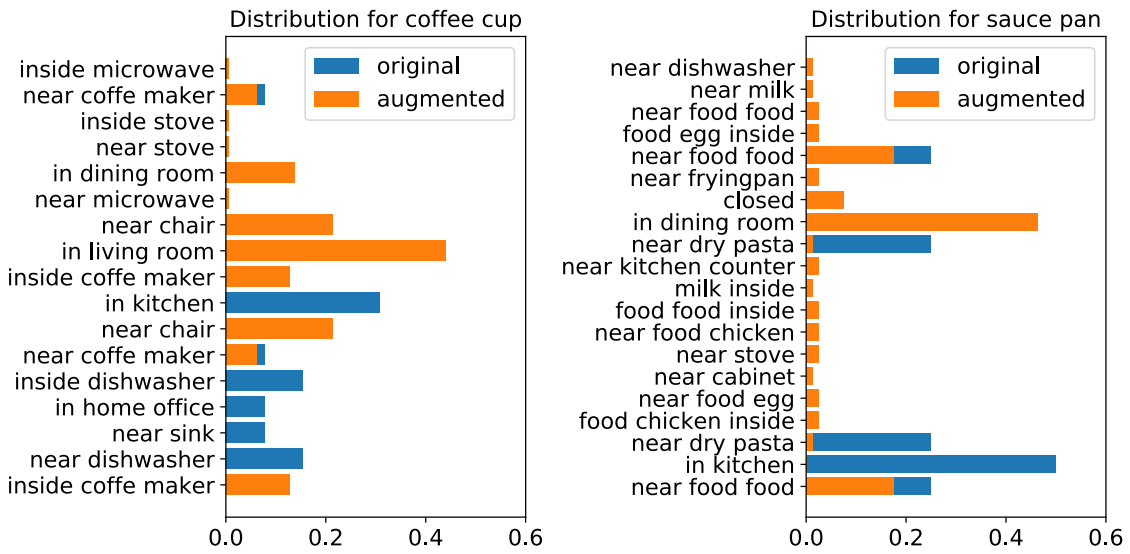


Figure 5-4: The effect of the dataset augmentation: changes in the distribution of preconditions for the objects in the environment.

5.5.5 Dataset Analysis

From the original 2807 programs in VirtualHome, we trim out the programs that can not be executed in the simulator with the environment sampled via preconditions, obtaining 1387 executable programs. Using the process described in Section 5.5.4, we extend these programs to our final dataset with around 30k programs. The significant increase in the program length is induced by the modified preconditions. For example, the agent needs to open containers to reach objects or make space to sit on a sofa. In Table 5.1, we show the statistics of the new dataset. To better understand the effects of augmentation, we show in Figure 5-4 shows the change in the distribution of preconditions of two example object classes, the cup and the sauce pan. The programs after augmentation show a less skewed distribution of preconditions, allowing for more diverse environments. Finally, we use the simulator to generate snapshots of the environment after executing each instruction of a program, as shown in Figure A-5. Note that some of the objects in the program do not have a model in the simulator, so we generate frames for a subset of 8421 programs.

	LCS	$\mathbb{F}_1$-relation	$\mathbb{F}_1$-state	$\mathbb{F}_1$	Executability	Parsability
Nearest Neighbors	.127	.019	.288	.041	-	-
Unaries	.372	.162	.142	.159	24.8%	75.3%
Graph	.404	.171	0.171	.172	23.1%	82.2%
FCActGraph	.469	.261	.273	.263	33.7%	**88.6%**
GRUActGraph	.508	**.410**	.408	.411	48.9%	87.9%
ResActGraph	**.516**	**.410**	**.420**	**.413**	**49.3%**	85.3%

Table 5.2: Induce program from ground truth sketches and ground truth graphs. ($K = 2$)

	LCS	$\mathbb{F}_1$-relation	$\mathbb{F}_1$-state	$\mathbb{F}_1$
Unaries (K=0)	.372	.160	.142	.592
ResActGraph (K=1)	.427	.262	.271	.264
ResActGraph (K=2)	**.516**	**.410**	**.420**	**.413**
ResActGraph (K=3)	.513	.399	.407	.401

Table 5.3: Ablation study of the propagation steps K.

5.6 Experiments

We split the dataset into train and test set in terms of different types of activities with ratio 7:3 and leave one apartment for the test set. We aim to test the capability of our model with novel *activities* and *environments*. We follow the same split for sketch prediction, where we only keep the original programs for the *desc2sketch*, since they contain the collected descriptions, and use the available frames for the *demo2sketch* task.

In this section, we describe the evaluation metrics, baselines. Next, we show the extensive experiment results of *ResActGraph*. Finally, we analyze the extent to which the proposed method is *environment-aware*. We will describe the implementation details in supplementary materials.

5.6.1 Evaluation Metrics

We analyze the performance of sketch prediction and program generation by measuring the normalized longest common subsequence (LCS) between the generated and ground truth sequences. LCS is sensitive to the order of the sequences and allows gaps

in between. To further measure if the generated programs achieve the specified activities, we compute the differences between the final environment graphs $\hat{G} = \Psi(\hat{p}, e)$ and $G = \Psi(p, e)$ using $\mathbb{F}_1$ scores[2]. In particular, we only compared the sub-graph containing the object instances mentioned in p and $\hat{p}$. We describe the details of $\mathbb{F}_1(\hat{G}, G)$ in the supplementary materials. Inspired by [7], we also compute $\mathbb{F}_1$-state and $\mathbb{F}_1$-relation.

Furthermore, inspired by program synthesis, we care whether the generated programs are "compilable" as well. We evaluate if the generated programs can be parsed (parsibility) and executed (executability) by the simulators. We will describe the detailed definition of them in the supplementary materials.

5.6.2 Baselines

We implement five different baselines of environment-aware agents to compare with the proposed *ResActGraph*.

Nearest Neighbors: For every example in the testing set, we retrieve the training sample that has a sketch with the highest LCS. In case of a tie, we pick the one with the most similar initial graph.

Unaries: We set $K = 0$ in Eq. 5.7. This model does not consider the relations of the objects. We use it to showcase the benefits of modeling object relations.

Graph: This model does not consider the change of graphs induced by programs (Eq. 5.8).

FCActGraph: This model uses a FC layer to model the graph changes. Specifically, it takes the $[h_v^{K_{t-1}}, Emb(\hat{\alpha}_{t-1}), m_{t-1}]$ as inputs and outputs $h_v^{K_t}$.

GRUActGraph: This model treats the graph changes as another sequence and uses a GRU to ingest $[Emb(\hat{\alpha}_{t-1}), m_{t-1}]$ as inputs and considers $h_v^{K_{t-1}}$ as hidden state to output $h_v^{K_t}$.

[2]If the generated programs cannot be parsed or cannot be executed, the $\mathbb{F}_1$ is set to be 0.

Sketch	GT Program	Generated Program
[Open] <washing machine> [Put] <basket> <washing machine> [Put] <soap> <washing machine> [SwitchOn] <washing machine>	[Walk] <bedroom> (273) [Walk] <basket> (1000) [Find] <basket> (1000) [Grab] <basket> (1000) [Walk] <bathroom> (1) [Walk] <washing machine> (1001) [Find] <washing machine> (1001) [Open] <washing machine> (1001) [Put] <basket> (1000) <washing machine> (1001) [Find] <soap> (1002) [Grab] <soap> (1002) [Put] <soap> (1002) <washing machine> (1001) [Find] <washing machine> (1001) [Close] <washing machine> (1001) [PlugIn] <washing machine> (1001) [SwitchOn] <washing machine> (1001)	[Walk] <bedroom> (273) [Walk] <basket> (1000) [Find] <basket > (1000) [Grab] <basket > (1000) [Find] <washing machine> (1001) [Open] <washing machine> (1001) [Put] <basket> (1000) <washing machine> (1001) [Find] <soap> (1002) [Grab] <soap> (1002) [Put] <soap> (1002) <washing machine> (1001) [Close] <washing machine> (1001) [Plugin] <washing machine> (1001) [SwitchOn] <washing machine> (1001)
Environment		
Washing machine (1001) is closed Washing machine (1001) is off Washing machine (1001) is unplugged Washing machine (1001) in bathroom (1) Soap (1002) inside Washing machine (1001)		

Figure 5-5: An example of the prediction of ResActGraph. We colore the LCS between the prediction and ground truth in light green. Note that the sketch is environment agnostic, so it does not specify the 'id' (the number in the parentheses) of the object instances.

5.6.3 Results

We show the results of *ResActGraph* quantitatively and qualitatively. Next, we show the ablation study of the number of the graph propagation steps. Finally, we show the prediction results of the whole system.

Program generation from sketches and graphs. The results are shown in Table 5.2. By comparing the Graph and Unaries, we show that aggregating information from neighboring nodes increases performances in nearly all metrics.

The three bottom rows of Table 5.2 show the results of models that consider graph changes induced by programs. The $\mathbb{F}_1$ scores and executability benefit the most, which is expected. For example, suppose there is a glass near an opened cabinet in the environment and the model predicts (`Grab Glass`, `Put Glass Cabinet`, `Close Cabinet`) at the first three steps, if the model wants to grab other things from the cabinet without opening it after $t \geq 4$, it fails since the cabinet is closed at $t = 3$.

Among the three bottom rows of Table 5.2, the proposed ResActGraph performs the best in $\mathbb{F}_1$. The reason is that using the residual architecture is easier for the model to learn the state "changes" compared to using FC. Using GRU to encode the state changes is also an alternative, but we observe that it converges slower since it has more number of parameters to learn and does not perform better.

Sketch		[Sit] <sofa> [Read] <book>		
Environment 1	Generated Program 1	Environment 2	Generated Program 2	
Book (263) in bedroom (23) Bookmark (27) near book (263) Bookmark (27) in bedroom (23) Sofa (101) in bedroom (23) Sofa (101) near bookmark (275) Sofa (101) is free Book (263) near sofa (1001)	[Walk] <bedroom> (23) [Walk] <book> (263) [Find] <book> (263) [Grab] <book> (263) [Find] <sofa> (101) [Sit] <sofa> (101) [Find] <bookmark> (27) [Read] <book> (263)	Sofa (101) in bedroom (23) Book (263) in bedroom (23) Bookmark (27) in bedroom (23) Sofa (101) occupied Phone (75) on sofa (101) Cards (261) on sofa (101) Game (231) on sofa (101) Check (213) on sofa (101)	[Walk] <bedroom> (23) [Walk] <sofa> (101) [Find] <sofa> (101) [Find] <phone> (75) [Grab] <phone> (75) [Release] <phone> (75) [Find] <cards> (261) [Grab] <cards> (261) [Release] <cards> (261) [Find] <game> (231)	[Grab] <game> (231) [Release] <game> (231) [Find] <check> (213) [Grab] <check> (213) [Release] <check> (213) [Sit] <sofa> (101) [Standup] [Walk] <book> (263) [Find] <book> (263) [Grab] <book> (263) [Read] <book> (263)

Figure 5-6: An example of the prediction of ResActGraph with the same sketch, but different environments. We highlight the difference between two environments with orange and color the LCS between two predictions in light green.

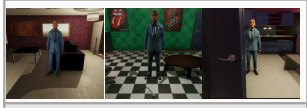
Demonstration	Environment	Generated Sketch	Generated Program	
	Phone (10) in living room (1)	[Grab] <phone>	[Walk] <living room> (1) [Walk] <phone> (10) [Find] <phone> (10)	[Walk] <phone> (10) [TurnTo] <phone> (10) [LookAt] <phone> (10) [Grab] <phone> (10)
Description	Environment	Generated Sketch	Generated Program	
Walk into the home office. Walk up to the chair, sit down in the chair. Type with the keyboard.	Chair (29) close to keyboard (2) Chair (29) is free Keyboard (2) close to computer Computer (31) in living room (1)	[Sit] <chair> [Type] <keyboard>	[Walk] <living room> (1) [Walk] <desk> (137) [Find] <chair> (29)	[Sit] <chair> (29) [Find] <keyboard> (2) [Type] <keyboard> (2)

Figure 5-7: Predictions from the ResActGraph given sketches from descriptions and demonstrations.

In Figure 5-5, we show the qualitative results of ResActGraph. Even though the generated program does not exactly match the ground truth, it reaches nearly the same environment state. In Figure 5-6, we show the results with the same sketch, but different initial environment states. Note that we only show the states and relations related to the programs. The model correctly induces correct actions w.r.t. the environment changes and the two generated programs nearly reach the same environment states.

Ablation studies. We show the effect of the number of propagation steps K in Table 5.3. Both the baseline and the proposed model benefit as K increases, and the proposed model performs better than the baseline regardless of different K. We found that the performance saturates when $K = 2$, so we fixed it for all other experiments.

Combining predicted sketches with ResActGraph. The LCS of $f_{demo2sketch}$ and $f_{desc2sketch}$ are 0.15 and 0.27 respectively. The reason why the LCS is low is that

	LCS	$\mathbb{F}_1$-relation	$\mathbb{F}_1$-state	$\mathbb{F}_1$	Executability	Parsability
Unaries	.289	.188	.191	.189	44.4%	73.4%
Graph	.297	.241	.233	.241	43.9%	89.9%
ResActGraph	**.331**	**.347**	**.339**	**.348**	**63.3%**	**92.5%**

Table 5.4: Induce program from sketches predicted from decsriptions and ground truth graphs. ($K = 2$)

	LCS	$\mathbb{F}_1$-relation	$\mathbb{F}_1$-state	$\mathbb{F}_1$	Executability	Parsability
Unaries	.257	.099	.090	.098	25.6%	74%
Graph	.284	.202	.202	.203	34.3%	82.8%
ResActGraph	**.327**	**.316**	**.323**	**.318**	**54.7%**	**86%**

Table 5.5: Inducing program from sketches predicted from demonstrations and ground truth graphs. ($K = 2$)

they are significantly shorter (on average 2.4 instructions) than the programs (on average 18.79). This makes the sketch prediction a quite challenging task where LCS is highly penalized even under small errors.

With the trained model, we can directly generate the programs from the demonstrations or descriptions. Note that we do not re-train the program generation model. In Table 5.4 and Table 5.5, we show the results of the program generation with the sketches predicted from descriptions and demonstrations respectively. The performance gap between the proposed model and the baselines becomes small. The reason is that the model is confused when the non-perfect sketches are given, resulting in similar performance. Note that all models still perform better than Unaries. Qualitative results are shown in Figure 5-7, showing that the model predicts the plausible sketch and *ResActGraph* generates plausible programs w.r.t to the sketch and the graph.

5.7 Discussion

In this Chapter, we moved beyond agents that would blindly perform activities in an environment and proposed a method to build *environment-aware agents*. We introduced sketches as environment-independent activity representations and addressed the problem in two steps: generating sketches from demonstrations or descriptions

and generating programs from sketches and graphs. To this end, we proposed a novel model, *ResActGraph* and created a dataset *VirtualHome-Env*, with sketches, environments, and programs to train and test it.

The *environment-aware program generation* is far from being solved and opens exciting research directions. While we assume access the truth state of the environment graph, one natural extension would be to predict it from environment observations [162], but this would still require an oracle providing observations of the interior of closed objects or unexplored areas, or a way to infer those from a partial observation of the environment. Additionally, though the *ResActGraph* updates the hidden states of nodes at each time step, the graph structure is only considered at the first step, when we do message passing to obtain the object embeddings. This approach works under this problem setting because there is only one agent in the environment, and therefore all the changes in the graph will be caused by this agent's actions. As a result, the *ResActGraph* model can learn to modify the embeddings of different objects according to the actions taken by the agent. This approach would not work if there was another agent interacting in the environment whose actions were not known, or in the case of a partially observable environment.

All the above are strong assumptions if we care about building assistive agents, since in most cases they will be coexisting with other humans who interact in the environment concurrently with the agent. In Part III we address these limitations, and study how to build agents that assist humans concurrently interacting in the environment.

Part III

Agents and benchmarks for Human-AI assistance

In Part I we presented a platform for agents to learn to interact in household environments and in Part II we described different approaches to command agents to perform household activities in our platform. Learning these activities in simulation provides a safe and scalable way to test agents under different conditions, before they can be deployed in the real world. However, for these agents to be effective assistants, they need to be able to cooperate with humans in the environments, adapting to their goals, preferences and behavior.

The Part III of this thesis explores this topic. We aim to build agents that instead of performing tasks instructed by humans, they can *collaborate* with them in performing these goals. For this, there are several challenges we need to address:

1. **Human Models:** We need to be able to simulate humans in simulation to reflect more realistically changing environments and so that agents can reason about their actions when collaborating in performing a given activity.

2. **Evaluation:** We need an evaluation framework to measure what constitutes effective assistance so that we can make progress in this task.

3. **Models of Assistance:** We need to build agents that can understand what are the goals of the humans and collaborate with them in achieving those goals.

In the next chapters we explore these questions. We propose agents that can serve as proxies for humans that interact in the environment to achieve different tasks. In contrast to the helper agents introduced in Part II, goals here are specified via logical predicates over the environment state (e.g. there should be 2 apples inside the fridge). Rather than using learned models, these agents are implemented via task planners, ensuring that their behavior is robust to other agents interacting in the environment. We then evaluate the effectiveness of different assistive agents by testing them with these human-like agents In particular, we measure the speedup of a task when the human is assisted by an agent compared to when the human does the task alone.

Chapter 6 explores the setting where the helper agent is shown a demonstration of a task, and is later asked to helper a human in completing the task in a new

environment. This setting is more challenging than the models introduced in Chapters 4 and 5, because both the human and the helper agents have partial observations over the environment and therefore have to explore and search for objects in order to complete tasks. We propose *Watch-And-Help*, a benchmark to study this problem quantiatively, as well as different agents to assist humans under this setting.

In Chapter 7 we extend the previous setting to agents that can offer assistance without a prior demonstration of the activity they need to help on. For this, we introduce *Online-Watch-And-Help*, a benchmark where agents have to concurrently infer what is the activity that the human is trying to do and assist in them in that activity. We propose a Neurally-Guided Inverse Planning approach to build an agent that can achieve this.

Chapter 6

Watch-And-Help: A Challenge for Human-AI Collaboration

6.1 Introduction

Humans exhibit altruistic behaviors at an early age [156]. Without much prior experience, children can robustly recognize goals of other people by simply watching them act in an environment, and are able to come up with plans to help them, even in novel scenarios. In contrast, the most advanced AI systems to date still struggle with such basic social skills.

In order to achieve the level of social intelligence required to effectively help humans, an AI agent should acquire two key abilities: i) social perception, i.e., the ability to understand human behavior, and ii) collaborative planning, i.e., the ability to reason about the physical environment and plan its actions to coordinate with humans. In this chapter, we are interested in developing AI agents with these two abilities.

Towards this goal, we introduce a new AI challenge, Watch-And-Help (WAH), which focuses on social perception and human-AI collaboration. In this challenge, an AI agent needs to collaborate with a human-like agent to enable it to achieve the goal faster. In particular, we present a 2-stage framework as shown in Figure 6-1. In the first, *Watch* stage, an AI agent (Bob) watches a human-like agent (Alice) performing

a task once and infers Alice's goal from her actions. In the second, *Help* stage, Bob helps Alice achieve the same goal in a different environment as quickly as possible (i.e., with the minimum number of environment steps).

This 2-stage framework poses unique challenges for human-AI collaboration. Unlike prior work which provides a common goal a priori or considers a small goal space [47, 26], our AI agent has to reason about what the human-like agent is trying to achieve by watching a single demonstration. Furthermore, the AI agent has to generalize its acquired knowledge about the human-like agent's goal to a new environment in the *Help* stage. Prior work does not investigate such generalization.

To enable multi-agent interactions in realistic environments, we will be testing this framework in VirtualHome. In contrast to previous chapters, here we will run the simulator interactively (that is, we will obtain observations and take actions one step at a time), and with multiple agents interacting concurrently, where one of the agents will be representing the human (Alice) and the other agent the helper (Bob). Furthermore, to test whether these agents are effective at assisting real humans, we build an interface for humans to interact with the simulator, and complete tasks assisted by the proposed agents.

We design an evaluation protocol and provide a benchmark for the challenge, including a goal inference model for the *Watch* stage, and multiple planning and deep reinforcement learning (DRL) baselines for the *Help* stage. Experimental results indicate that to achieve success in the proposed challenge, AI agents must acquire strong social perception and generalizable helping strategies. These fundamental aspects of machine social intelligence have been shown to be key to human-AI collaboration in prior work [48, 5]. In this work, we demonstrate how we can systematically evaluate them in more realistic settings at scale.

Our main contributions are: i) a new social intelligence challenge, Watch-And-Help, for evaluating AI agents' social perception and their ability to collaborate with other agents, ii) an extension to VirtualHome to allow interactions with built-in agents or real humans, and iii) a benchmark consisting of multiple planning and learning based approaches which highlights important aspects of machine social intelligence.

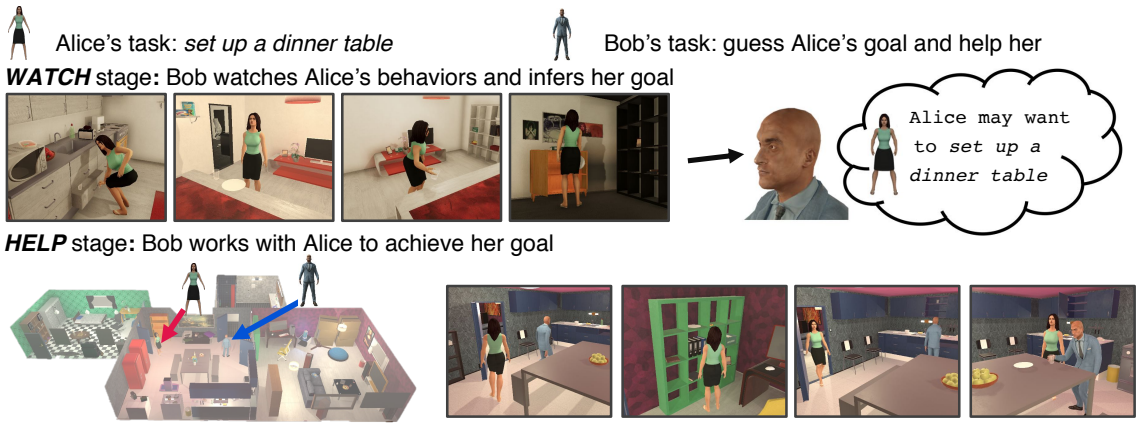


Figure 6-1: Overview of the Watch-And-Help challenge. The challenge has two stages: i) in the *Watch* stage, Bob will watch a single demonstration of Alice performing a task and infer her goal; ii) then in the *Help* stage, based on the inferred goal, Bob will work with Alice to help finish the same task as fast as possible in a *different* environment.

6.2 Related Work

Human activity understanding. An important part of the challenge is to understand human activities. Prior work on activity recognition has been mostly focused on recognizing short actions [133, 23, 41], predicting pedestrian trajectories [73, 3], recognizing group activities [129, 32, 61], and recognizing plans [70, 114]. We are interested in the kinds of activity understanding that require inferring other people's mental states (e.g., intentions, desires, beliefs) from observing their behaviors. Therefore, the *Watch* stage of our challenge focuses on the understanding of humans' goals in a long sequence of actions instead. This is closely related to work on computational Theory of Mind that aims at inferring humans' goals by observing their actions [11, 152, 113, 131]. However, in prior work, activities were simulated in toy environments (e.g., 2D grid worlds). In contrast, this work provides a testbed for conducting Theory-of-Mind type of activity understanding in simulated real-world environments.

Embodied Human-AI cooperation benchmarks. Conventional human-robot cooperation studies have typically been conducted in lab environments [47, 34, 103, 118], lacking both reproducibility and scalability. Recently there have been bench-

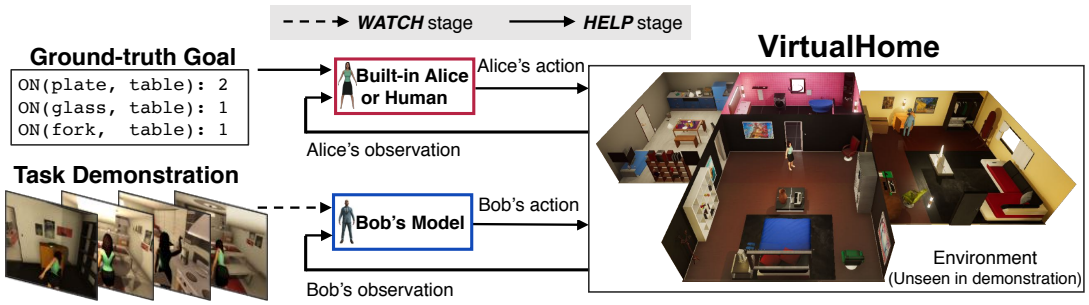


Figure 6-2: The system setup for the WAH challenge. An AI agent (Bob) watches a demonstration of a human-like agent (Alice) performing a task, and infers the goal (a set of predicates) that Alice was trying to achieve. Afterwards, the AI agent is asked to work together with Alice to achieve the same goal in a new environment as fast as possible. To do that, Bob needs to plan its actions based on i) its understanding of Alice's goal, and ii) a partial observation of the environment. It also needs to adapt to Alice's plan. We simulate environment dynamics and provide observations for both agents in our VirtualHome multi-agent platform. The platform includes a built-in agent as Alice which is able to plan its actions based on the ground-truth goal, and can react to any world state change caused by Bob through re-planning at every step based on its latest observation. Our system also offers an interface for real humans to control Alice and work with an AI agent in the challenge.

marks designed to systematically evaluate agents' ability to collaborate with human teammates [26, 13]. However, most of the existing benchmarks focus on simple game environments and assume a common goal given to both the AI and human agents a priori. The setup in WAH is much more challenging – the goal is sampled from a large space, needs to be inferred from a single demonstration, and must be performed in realistic and diverse household environments through a long sequence of actions.

6.3 The Watch-And-Help Challenge

The Watch-And-Help challenge aims to study AI agents' ability to help humans in household activities. To do that, we design a set of tasks defined by predicates describing the final state of the environment. For each task, we first provide Bob a video that shows Alice successfully performing the activity (*Watch* stage), and then place both agents in a new environment where Bob has to help Alice achieve the same goal with the minimum number of time steps (*Help* stage).

Figure 6-2 provides an overview of the system setup for the Watch-And-Help challenge. For this challenge, we will be using VirtualHome, allowing agents to interact concurrently in the environment and providing observations for the agents. Alice represents a built-in agent in the system; she plans her actions based on her own goal and a partial observation of the environment. Bob serves as an external AI agent, who does not know Alice's ground-truth goal and only has access to a single demonstration of Alice performing the same task in the past. During the *Help* stage, Bob receives his observation from the system at each step and sends an action command back to control the avatar in the environment. Alice, on her part, updates her plan at each step based on her latest observation to reflect any world state change caused by Bob. We also allow a human to control Alice in our system. We discuss how the system and the built-in agent work in Section 6.4.

Problem Setup. Formally, each task in the challenge is defined by Alice's goal g (i.e., a set of goal predicates), a demonstration of Alice taking actions to achieve that goal $D = \{s^t_{\text{Alice}}, a^t_{\text{Alice}}\}_{t=1}^{T}$ (i.e., a sequence of states s^t_{Alice} and actions a^t_{Alice}), and a new environment where Bob collaborates with Alice and help achieve the same goal as quickly as possible. During training, the ground-truth goal of Alice is shown to Bob as supervision; during testing, Bob no longer has access to the ground-truth goal and thus has to infer it from the given demonstration.

Goal Definitions. We define the goal of a task as a set of predicates and their counts, which describes the target state. Each goal has 2 - 8 predicates. For instance, "`ON(plate, dinnertable):2`; `ON(wineglass, dinnertable):1`" means "putting two plates and one wine glass onto the dinner table." The objects in a predicate refer to object classes rather than instances, meaning that any object of a specified class is acceptable. This goal definition reflects different preferences of agents (when setting up a dinner table, some prefer to put water glasses, others may prefer to put wine glasses), increasing the diversity in tasks. We design five predicate sets representing five types of household activities: 1) setting up a dinner table, 2) putting groceries / leftovers to the fridge, 3) preparing a simple meal, 4) washing dishes, and 5) reading a book while having snacks or drinks. In total, there are 30 different types of predi-

cates. In each task, the predicates of a goal are sampled from one of the five predicate sets (as a single household activity). More details about the predicate sets and goal definitions are listed in Appendix B.2.1.

6.4 Environment

Building machine social intelligence for real-life activities poses additional challenges compared to typical multi-agent settings, such as far more unconstrained goal and action spaces, and the need to display human actions realistically for social perception.

With that in mind, we use VirtualHome to test our challenge. We set the platform so that multiple agents (including real humans) can execute actions concurrently and observe each other's behaviors. Furthermore, we embed planning-based agents in the environment as virtual humans that AI agents can reason about and interact with.

In the rest of this section, we describe the observations, actions, and the built-in human-like agent that we will be using for this challenge. Appendix B.1 includes more information.

Observation space. As described in Section 2.4, VirtualHome supports symbolic and visual observations, allowing agents to learn helping behaviors under different conditions. The symbolic observations consist on a scene graph, with nodes representing objects and edges describing spatial relationships between them. To focus on the coordination aspects of the challenge, in this work we use symbolic observations. We consider two types of observations: 1) full observations, which allows agents to obtain full information about the environment via scene graph and 2) partial observations, which only allows the agents to see a local subset of objects based on a symbolic "field-of-view", that is, those objects that are in the same room as the agent and are not inside some closed container.

Action space. Agents can navigate in the environment and interact with objects in it. Remember that, to interact with objects, agents need to specify an action and the index of the intended object (e.g., "grab ⟨3⟩" stands for grabbing the object with id 3). We want agents that behave like humans, acting in the environment based on

the information they have in the moment. Therefore, an agent can only interact with objects that are within its field of sight, changing its action space at every step.

Human-like agents. To enable a training and testing environment for human-AI interactions, it is critical to incorporate built-in agents that emulate humans when engaging in multi-agent activities. [26] has attempted to train policies imitating human demonstrations. But those policies would not reliably perform complex tasks in partially observable environments. Therefore, we devise a planning-based agent with bounded rationality. This agent operates on the symbolic representation of its partial observation of the environment. As shown in Figure 6-3, it relies on two key components: 1) a belief of object locations in the environment (Figure B-3 in Appendix B.1.1), and 2) a hierarchical planner, which uses Monte Carlo Tree Search (MCTS) [22] and regression planning (RP) [75] to find a plan for a given goal based on its belief. We use *VirtualHome-Symbolic*, defined in Section 2.5 as a world-model for the planner. At every step, the human-like agent updates its belief based on the latest observation, finds a new plan, and executes the first action of the plan concurrently with other agents. The proposed design allows agents to robustly perform tasks in partially observable environments while producing human-like behaviors[1]. We provide more details of this agent in Appendix B.1.1.

6.5 Benchmark

6.5.1 Evaluation Protocol

Training and Testing Setup. We create a training set with 1011 tasks and 2 testing sets (test-1, test-2). Each test set has 100 tasks. We make sure that i) the helping environment in each task is different from the environment in the pairing demonstration (we sample a different apartment and randomize the initial state, similar to Section 5.5.2 but without considering any program or preconditions), and ii)

[1]We conducted a user study rating how realistic were the trajectories of the agents and those created by humans, and found no significant difference between the two groups. More details can be found in Appendix B.4.3.

goals (predicate combinations) in the test set are unseen during training. To evaluate generalization, we hold out apartments 6 and 7 (see Section 2.3.2) for the *Help* stage in the test sets. For the training set and test-1 set, all predicates in each goal are from the same predicate set, whereas a goal in test-2 consists of predicates sampled from two different predicates sets representing multi-activity scenarios (e.g., putting groceries to the fridge and washing dishes). Note that during testing, the ground-truth goals are not shown to the evaluated Bob agent. More details can be found in Appendix B.2. An episode is terminated once all predicates in Alice's goal are satisfied (i.e., a success) or the time limit (250 steps) is reached (i.e., a failure).

Evaluation Metrics. We evaluate the performance of an AI agent by three types of metrics: i) success rate, ii) speedup, and iii) a cumulative reward. For speedup, we compare the episode length when Alice and Bob are working together (L_{Help}) with the episode length when Alice is working alone (L_{Alice}), i.e., $L_{\text{Alice}}/L_{\text{Bob}} - 1$. To account for both the success rate and the speedup, we define the cumulative reward of an episode with T steps as $R = \sum_{t=1}^{T} \mathbb{1}(s^t = s_g) - 0.004$, where s^t is the state at step t, s_g is the goal state. R ranges from -1 (failure) to 1 (achieving the goal in zero steps).

6.5.2 Baselines

To address this challenge, we propose a set of baselines that consist of two components as shown in Figure 6-4: a goal inference model and a goal-conditioned helping planner / policy. In this paper, we assume that the AI agent has access to the ground-truth states of objects within its field of view (but one could also use raw pixels as input). We describe our approach for the two components below.

Goal inference. We train a goal inference model based on the symbolic representation of states in the demonstration. At each step, we first encode the state using a Transformer [154] over visible objects and feed the encoded state into a long short-term memory (LSTM) [55]. We use average pooling to aggregate the latent states from the LSTM over time and build a classifier for each predicate to infer its count. Effectively, we build 30 classifiers, corresponding to the 30 predicates in our taxonomy and the fact that each can appear multiple times.

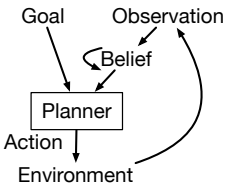


Figure 6-3: Overview of the human-like agent.

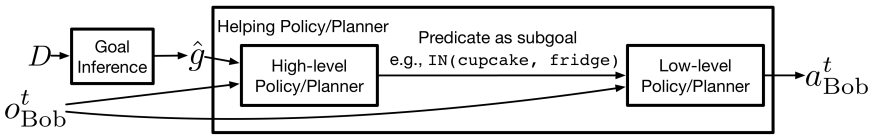


Figure 6-4: The overall design of the baseline models. A goal inference model infers the goal from a demonstration D and feeds it to a helping policy (for learning-based baselines) or to a planner to generate Bob's action. We adopt a hierarchical approach for all baselines.

Helping policy/planner. Due to the nature of the tasks in our challenge – e.g., partial observability, a large action space, sparse rewards, strict preconditions for actions – it is difficult to search for a helping plan or learn a helping policy directly over the agent's actions. To mitigate these difficulties, we propose a hierarchical architecture with two modules for both planning and RL-based approaches as shown in Figure 6-4. At every step, given the goal inferred from the demonstration, $\hat{g}$, and the current observation of Bob, a high-level policy or planner will output a predicate as the best subgoal to pursue for the current step; the subgoal is subsequently fed to a low-level policy or planner which will yield Bob's action a^t_{Bob} at this step. In our baselines, we use either a learned policy or a planner for each module. We use the symbolic representation of visible objects as Bob's observation o^t_{Bob} for all models. We summarize the overall design of the baseline models as follows (please refer to Appendix B.3 for the details of models and training procedures):

HP: A hierarchical planner, where the high-level planner and the low-level planner are implemented by MCTS and regression planning (RP) respectively. This is the same planner as the one for Alice, except that i) it has its own partial observation and thus a different belief from Alice, and ii) when given the ground-truth goal, the high-level planner uses Alice's plan to avoid overlapping with her.

Hybrid: A hybrid model of RL and planning, where an RL policy serves as the high-level policy and an RP is deployed to generated plans for each subgoal sampled from the RL-based high-level policy. This is to train an agent equipped with basic skills for achieving subgoals to help Alice through RL.

HRL: A hierarchical RL baseline where high-level and low-level policies are all

learned.

Random: A naive agent that takes a random action at each step.

To show the upper bound performance in the challenge, we also provide two oracles:

Oracle$^{\mathbf{B}}$: An HP-based Bob agent with full knowledge of the environment and the true goal of Alice.

Oracle$^{\mathbf{A,\ B}}$: Alice has full knowledge of the environment too.

6.5.3 Results

We evaluate the *Watch* stage by measuring the recognition performance of the predicates. The proposed model achieves a precision and recall of 0.85 and 0.96 over the test-1 set. To evaluate the importance of seeing the full demonstration, we test a model that takes as input the graph representation of the last observation, leading to a precision and recall of 0.79 and 0.75. When using actions taken by Alice as the input, the performance increases to a precision and recall of 0.99 and 0.99. The chance precision and recall is 0.08 and 0.09.

We report the performance of our proposed baselines (average and standard error across all episodes) in the *Help* stage in Figure 6-5. In addition to the full challenge setup, we also report the performance of the helping agents using true goals (indicated by the subscript $_{\text{TG}}$) and using random goals (by $_{\text{RG}}$), and the performance of Alice working alone. Results show that planning-based approaches are the most effective in helping Alice. Specifically, $\mathbf{HP}_{\text{TG}}$ achieves the best performance among non-oracle baselines by using the true goals and reasoning about Alice's future plan, avoiding redundant actions and collisions with her (Figure 6-6 illustrates an example of collaboration). Using the inferred goals, both **HP** and **Hybrid** can offer effective help. However, with a random goal inference ($\mathbf{HP}_{\text{RG}}$), a capable Bob agent becomes counter productive – frequently undoing what Alice has achieved due to their conflicting goals (conflicts appear in 40% of the overall episodes, 65% for *Put Groceries* and *Set Meal*). This calls for an AI agent with the ability to adjust its goal inference dynamically by observing Alice's behavior in the new environment (e.g., Alice

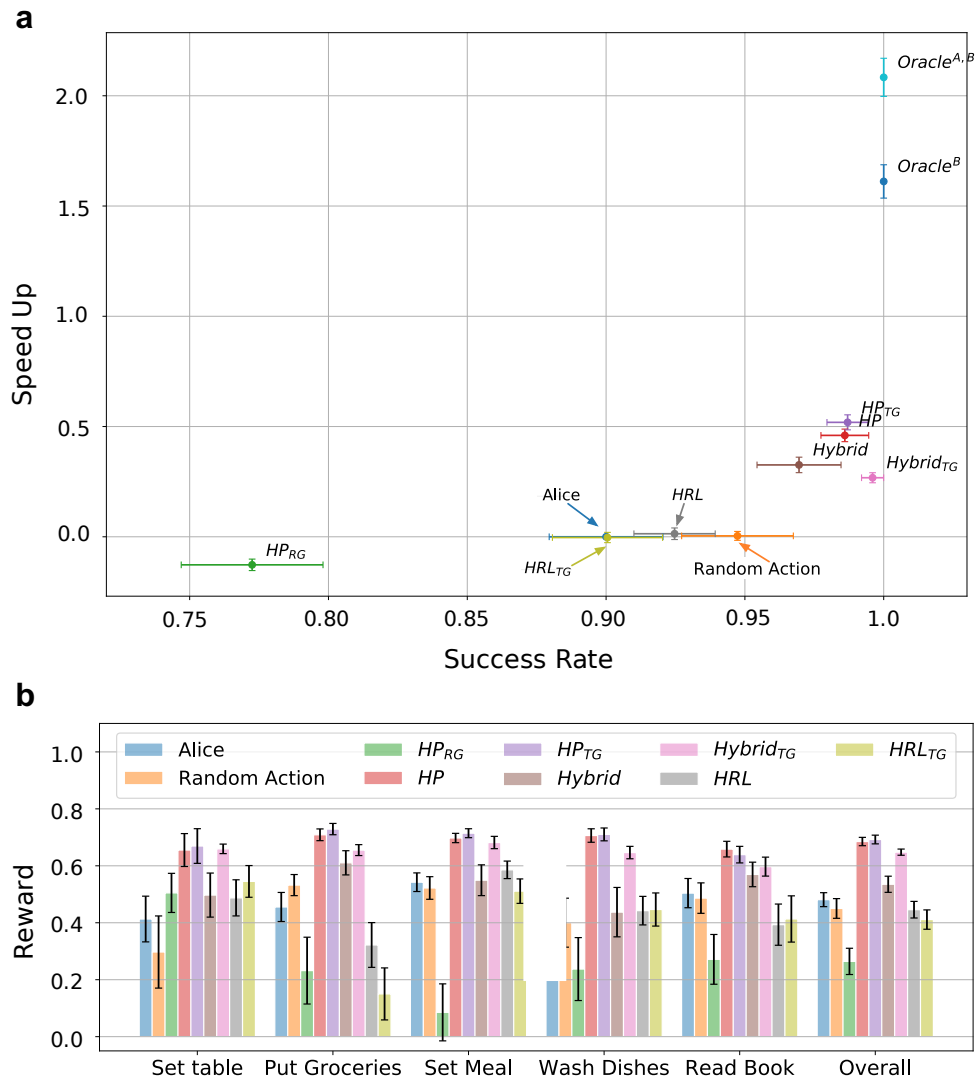


Figure 6-5: a) Success rate (x axis) and speedup (y axis) of all baselines and oracles. The performance of an effective Bob agent should fall into the upper-right side of the Alice-alone baseline in this plot. b) Cumulative reward in the overall test set and in each household activity category (corresponding to the five predicate sets introduced in Section 6.3).

correcting a mistake made by Bob signals incorrect goal inference). **HRL** works no better than **Random**, even though it shares the same global policy with **Hybrid**. While the high level policy selects reasonable predicates to perform the task, the low level policy does not manage to achieve the desired goal. In most of the cases, this is due to the agent picking the right object, but failing to put it to the target location afterwards. This suggests that it is crucial for Bob to develop robust abilities to

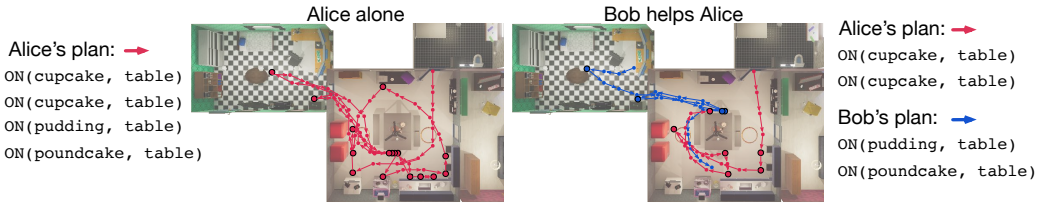


Figure 6-6: Example helping plan. The arrows indicate moving directions and the circles with black borders indicate moments when agents interacted with objects. When working alone (left), Alice had to search different rooms; but with Bob's help (right), Alice could finish the task much faster.

achieve the subgoals. There is no significant difference between **Random** and **Alice** baselines ($t(99) = -1.38$, $p = 0.17$).

We also evaluate the baselines in the test-2 set, containing tasks with multiple activities. The goal inference model achieves a precision and recall of 0.68 and 0.64. The performance gap from test-1 indicates that the model fails to generalize to generalize to multi-activity scenarios, overfitting to predicate combinations seen during training. For the *Help* stage, we evaluate the performance of Alice alone, as well as the best performing baseline, **HP**. Alice achieves a success rate of 95.40 ± 0.01, while the **HP** baseline achieves a success rate of 88.60 ± 0.02 and a speedup of 0.21 ± 0.04. Compared to its performance in the test-1 set, the **HP** baseline suffers a significant performance degradation in the test-2 set, which is a result of the lower goal recognition accuracy in the *Watch* stage.

To better understand the important factors for the effectiveness of helping, we analyze the helping behaviors exhibited in our experiments and how they affect Alice from the following aspects.

Predicting Alice's Future Action. When coordinating with Alice, Bob should be able to predict Alice's future actions to efficiently distribute the work and avoid conflicts (Figure 6-7ab).

Helping Alice's Belief's Update. In addition to directly achieving predicates in Alice's goal, Bob can also help by influencing Alice's belief update. A typical behavior is that when Bob opens containers, Alice can update her belief accordingly and find the goal object more quickly (Figure 6-7c). This is the main reason why Bob with random actions can sometimes help speed up the task too.

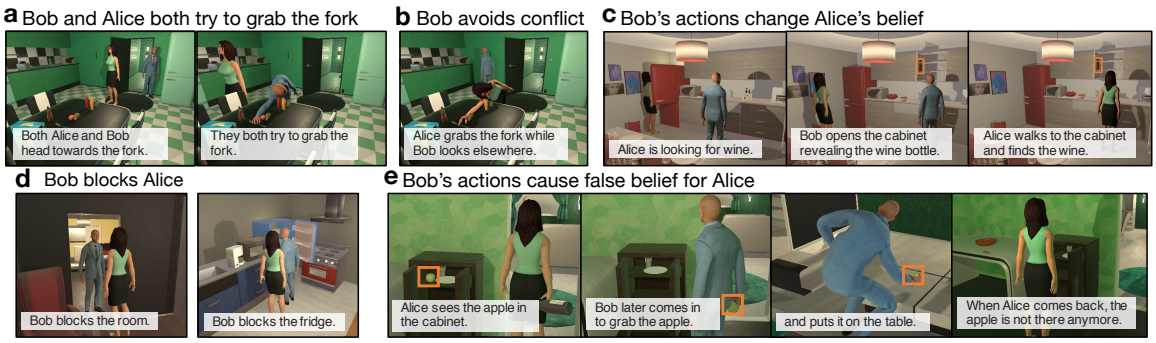


Figure 6-7: Example helping behaviors. We show more examples in the supplementary video.

Multi-level Actions. The current baselines do not consider plans over low-level actions (e.g., pathfinding). This strategy significantly decreases the search space, but will also result in inefficient pathfinding and inability to predict other agents' future paths. Consequently, Bob agent sometimes unintentionally blocks Alice (Figure 6-7d). A better AI agent should consider actions on both levels.

False Belief. Actions taken by an agent may cause another agent to have false beliefs (Figure 6-7e).

6.6 Human Experiments

Our ultimate goal is to build AI agents that can work with real humans. Thus, we further conduct the following two human experiments, where Alice is controlled by a real human.

Experiment 1: Human performing tasks alone. In this experiment, we recruited 6 subjects to perform tasks alone by controlling Alice. Subjects were given the same observation and action space as what the human-like agent had access to. They could click one of the visible objects (including all rooms) and select a corresponding action (e.g., "walking towards", "open") from a menu to perform. They could also choose to move forward or turn left/right by pressing arrow keys. We evaluated 30 tasks in the test set. Each task was performed by 2 subjects, and we used the average steps they took as the single-agent performance for that task, which is

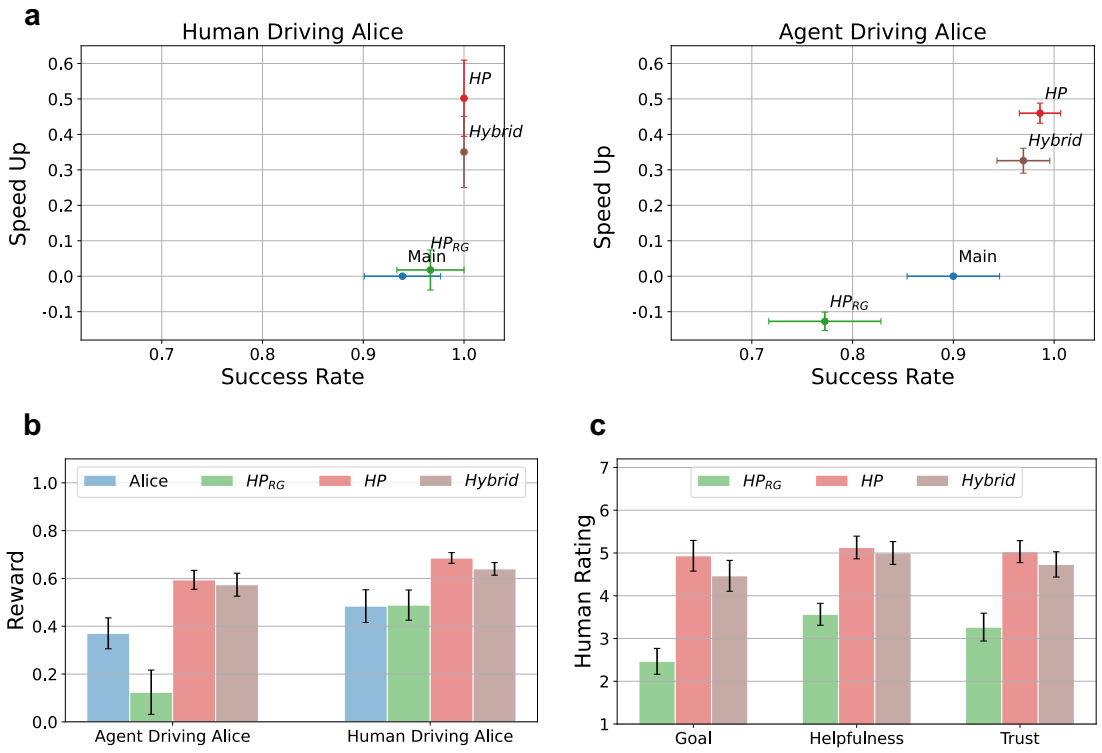


Figure 6-8: a) Success rate (x axis) and speedup (y axis). b) Cumulative reward with real humans or with the human-like agent) Subjective ratings from Exp. 2. Here, **Alice** refers to humans or the human-like agent acting alone, whereas **HP**, **Hybrid**, and $\mathbf{HP}_{\mathrm{RG}}$ indicate different AI agents helping either humans or the human-like agent. All results are based on the same 30 tasks in the test set.

then used for computing the speedup when AI agents help humans. The performance of a single agent when being controlled by a human or by a human-like agent in these 30 tasks is shown in Figure 6-8ab with the label of **Alice**. Human players are slightly more efficient than the human-like agent but the difference is not significant, as reported by the t-test over the number of steps they took ($t(29) = -1.63$, $p = .11$).

Experiment 2: Collaboration with real humans. This experiment evaluates how helpful AI agents are when working with real humans. We recruited 12 subjects and conducted 90 trials of human-AI collaboration using the same 30 tasks as in Exp. 1. In each trial, a subject was randomly paired with one of three baseline agents, **HP**, **Hybrid**, and $\mathbf{HP}_{\mathrm{RG}}$, to perform a task. After each trial, subjects were asked to rate the AI agent they just worked with on a scale of 1 to 7 based on three criteria commonly used in prior work [56]: i) how much the agent knew about the true goal

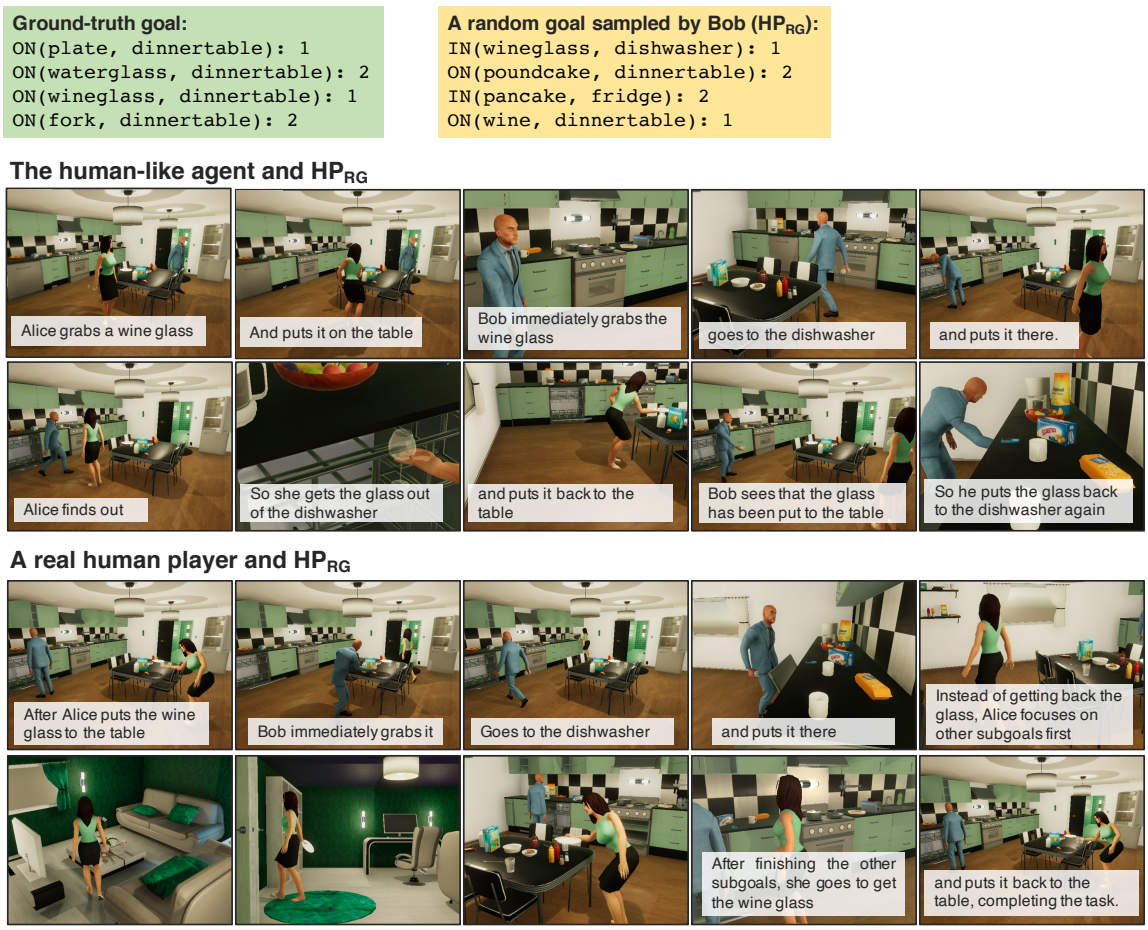


Figure 6-9: An example of how real human differs from the human-like agent when working with an AI agent (i.e., $\mathbf{HP}_{\mathrm{RG}}$) with a conflicting goal. In this example, Bob incorrectly thinks that Alice wants to put the wine glass to the dishwasher whereas Alice actually wants to put it to the dinner table. When controlled by a human-like agent, Alice enters into a loop with Bob trying to change the location of the same object. The real human player, on the other hand, avoids this conflict by first focusing on other objects in the goal, and going back to the conflicting object after all the other goal objects have been placed on the dinner table. Consequently, the real human completes the full task successfully within the time limit.

(1 - no knowledge, 4 - some knowledge, 7 - perfect knowledge), ii) how helpful you found the agent was (1 - hurting, 4 - neutral, 7 - very helpful), and iii) whether you would trust the agent to do its job (1 - no trust, 4 - neutral, 7 - full trust). For a fair comparison, we made sure that the random goal predictions for $\mathbf{HP}_{\mathrm{RG}}$ were the same as the ones used in the evaluation with the human-like agent.

As shown Figure 6-8, the ranking of the three baseline AI agents remains the same when the human-like agent is replaced by real humans, and the perceived performance

(subjective ratings) is consistent with the objective scores. We found no significant difference in the objective metrics between helping humans and helping the human-like agent; the only exception is that, when paired with real humans, $\mathbf{HP}_{\mathrm{RG}}$ had a higher success rate (and consequently a higher average cumulative reward). This is because humans recognized that the AI agent might have conflicting subgoals and would finish other subgoals first instead of competing over the conflicting ones with the AI agent forever, whereas the human-like agent was unable to do so. Figure 6-9 shows an example. This adaption gave humans a better chance to complete the full goal within the time limit. We provide more details of the procedures, results, and analyses of the human experiments in Appendix B.4.

6.7 Discussion

In this chapter, we studied how to build and test agents that could assist humans in household environments. A key difference with the previous work on this thesis, is that here we were interested in agents that would not only perform activities, but coordinate with humans in achieving them. For this, we proposed an AI challenge to demonstrate social perception and human-AI collaboration in common household activities, and we adapted VirtualHome to test an AI agent's ability to reason about other agents' mental states and help them in unfamiliar scenarios. Our experimental results demonstrate that the proposed challenge can systematically evaluate key aspects of social intelligence at scale. We also show that our human-like agent behaves similarly to real humans in the proposed tasks and the objects metrics are consistent with subject ratings.

While this challenge makes a step towards a better of evaluation of assistive agents, many challenges remain. Here, we developed an agent with bounded rationality to serve as a human proxy, but this may not be representative of how humans behave in an environment, much less when there is another agent trying to assist them. One way to address this is to learn a model from human behavior that can be deployed as a built-in agent, but human behavior data is generally hard to collect and there

are no guarantees that the learned model will be robust to new environments or to a new agent concurrently interacting in the environment to try to help the human.

Similarly, this challenge focused in agents that could infer human goals and coordinate with them using non-verbal cues. This setting opens up exciting directions of future work in goal inference and theory of mind, but it would be worth exploring how to build assistive agents when direct communication is allowed. Part II explored agents that would perform activities from language instructions, but in order to be effective assistants, agents should be able to understand commands as the task is happening, and similarly be able to ask questions or communicate information about the task to humans when appropriate.

Perhaps one of the most restrictive assumptions in our setting is the fact that agents need a *Watch* phase to infer the goal of the human before starting collaboration. While this assumption allows us to simplify our framework, separating goal inference and coordination, it also reduces the usefulness of our approach. On one hand, it is not realistic to assume that an agent will have access to a demonstration of the task they need to help with, and in such case it may be more efficient to simplify specify the task via language or the logical predicates. On the other hand, separating goal inference and coordination prevents agents to re-evaluate the goal they need to help with as the task takes place, which can result in agents hindering in the true task if the inferred goal is not correct, as shown in Figure 6-9.

In the next chapter, we will be relaxing this assumption, and propose an agent that can help in performing tasks without a prior demonstration or description. The proposed agent will therefore infer the goal of the task as it happens and decide how to best help humans with the information it has available.

Chapter 7

Online Probabilistic Assistive Agents

7.1 Introduction

There has been growing interest in engineering socially intelligent AI agents, such as autonomous vehicles or service robots, that can safely and productively work with humans in the real world. Prior work on human-AI cooperation has achieved some success in scenarios where AI agents are given the true human goals a priori or only need to help humans in simple environments with a small state space. However, it remains very challenging to build AI assistants that can help humans perform all the activities of daily life in more natural settings, such as in our homes, where the space of human goals is vast and a person's goal at any point in time will not generally be known with certainty.

In the previous chapter, we presented a benchmark and baselines to develop agents that could assist humans in a variety of tasks. In that setting, helper agents were given a demonstration of a task and were then asked to collaborate with a human or human-like agent in completing that same task in a new environment. While this setting allows us to test important aspects of Human-AI collaboration, such as goal inference and coordination, it is also limited. In many real-world scenarios agents may not have access to information about the task beforehand, and need to infer it, to be able to help, as the human is performing it.

Our goal here is to build agents that can help people perform a wide range of

tasks under this scenario. Our AI agents must have the ability to infer the true goals of humans based on past observations in an online fashion, plan how to help humans without disrupting them, and adapt to their behaviors by simultaneously updating goal inference and helping strategies as the task progresses (as illustrated in Figure 7-1). Such ability comes naturally to people but has proven difficult for AI agents to date, due to two main technical obstacles. On the one hand, online goal inference in realistic environments is extremely difficult due to large state, action, and goal spaces; on the other hand, inaccurate or ambiguous goal inferences often lead to ineffective or even counterproductive attempt to help in systems that are not aware of their own uncertainty.

To address these challenges, we propose a novel human-AI cooperation method, NOPA (Neurally-guided Online Probabilistic Assistance). As illustrated in Figure 7-2, NOPA consists of two main components: (1) a neurally-guided online goal inference module and (2) an uncertainty-aware helping planner. The neurally-guided online goal inference modules first produces bottom up goal proposals from a neural network and then maintains a set of predictions of goals and future trajectories consistent with the observed actions via particle filtering and inverse planning. This ensures that inferences are both fast and robust. Given the latest predictions and their certainty, the helping planner first identifies a subgoal that is most valuable to help with and then plans the corresponding helping actions using a symbolic planner. The resulting helping plan can adapt to all levels of uncertainty in the predictions. For instance, when there are multiple possible target locations for a goal object, the AI agent will deliver the object to the human agent instead of risking misplacing the object.

For evaluation, we present a new embodied AI challenge, Online Watch-And-Help (O-WAH). Unlike most existing challenges on AI assistance [26], in O-WAH, a helper agent (controlled by AI) needs to infer the goal of a main agent in an online fashion and simultaneously help achieve the inferred goal as efficiently as possible. We evaluate agents built with NOPA and several baselines in a range of household tasks, helping the main agent controlled by either a human player or a planner-based agent. The experimental results show that NOPA significantly outperforms all baselines. We are

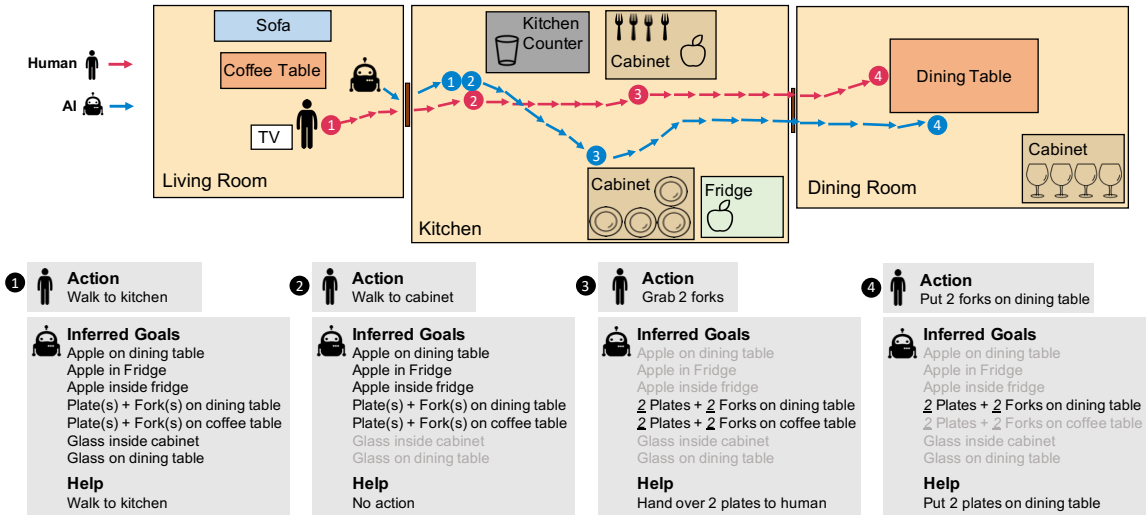


Figure 7-1: Illustration of successful online assistance to a human user, where the arrow shows the moving directions and the numbers indicate key steps. Without prior observation, the AI agent initially has no knowledge about the human's goal, thus would opt to observe. As it observes more human actions, it becomes more and more confident in its goal inference, so it would dynamically adjust its helping subgoal. For instance, in this figure, the AI first sees the human walking towards a cabinet and consequently infers that the goal involves objects inside of the cabinet. After the human grabs 2 forks, the AI now infers the goal is to put 2 sets of dinning pieces (plates and forks) to the dinning table or the coffee table; but since it is uncertain about the goal location, it will hand over 2 plates to the human instead of randomly guessing a location.

also able to observe intelligent helping strategies emerging from NOPA adapting to new observations.

In summary, our main contribution includes (1) a neurally-guided online probabilistic assistance method for effective human-AI cooperation in complex settings and (2) a new embodied human-AI cooperation challenge, Online Watch-And-Help, as a testbed for training and testing AI assistants to perform online goal inference and helping in realistic virtual home environments.

7.2 Related Work

Embodied Human-AI cooperation benchmarks. We reviewed in Chapter 6 some of the existing benchmarks aimed at testing Human-AI cooperation. In this

chapter, we extend Watch-And-Help to an online-assistance setting. As a result, the AI helper needs to be able to infer the goal of the main agent in an online fashion, and has to be able to help even when the goal is not fully known. We show in Table C.1 in Appendix C a table comparing similar benchmarks testing Human-AI cooperation. Watch-And-Help and Online Watch-And-Help present a much larger action and goal space than previous work, and test generalization to new environments.

Online goal inference. Recent online goal inference approaches generally fall into two categories – (1) feedforward prediction that directly maps observed past trajectories to possible goals, typically enabled by goal prediction networks [113, 24, 85, 99, 35, 171, 166, 149, 90], or (2) generative approaches such as inverse planning [25, 114, 137, 11, 152, 172, 100, 146] and inverse reinforcement learning [2, 52, 63, 163] which conduct inference by comparing generated plans or policies of given goal hypotheses with observed actions. Feedforward methods can learn to recognize useful patterns for fast inference and can perform reasonably well in simple tasks (such as destination prediction for pedestrians [85]) when trained with a large amount of data. However, in uncertain or unfamiliar scenarios, generative approaches, in particular, inverse planning-based methods, often outperform feedfoward prediction due to their ability to imagine rational behaviors under various conditions. One of the main limitations of inverse planning-based methods is that they can be extremely slow if the goal space is very large and may have to rely on manually designed heuristics to speed up the inference [172, 100]. Our work integrates both types of approaches to achieve both speed and robustness.

Embodied Human-AI cooperation with unknown goals. There has been a rich history of research on human-AI cooperation. Many of the existing works assume a known common goal shared among human and AI partners [47, 77, 103, 118, 62, 26, 13, 58, 159]. However, in the real world, AI agents often need to infer humans' goals on the fly. There has been work on helping with inferred goals [39, 86, 5, 52, 64] which shows that an accurate goal inference can improve the objective and perceived performance of AI agents. However, when the goal inference is uncertain, helping with inferred goals often leads to counterproductive behaviors such as undoing

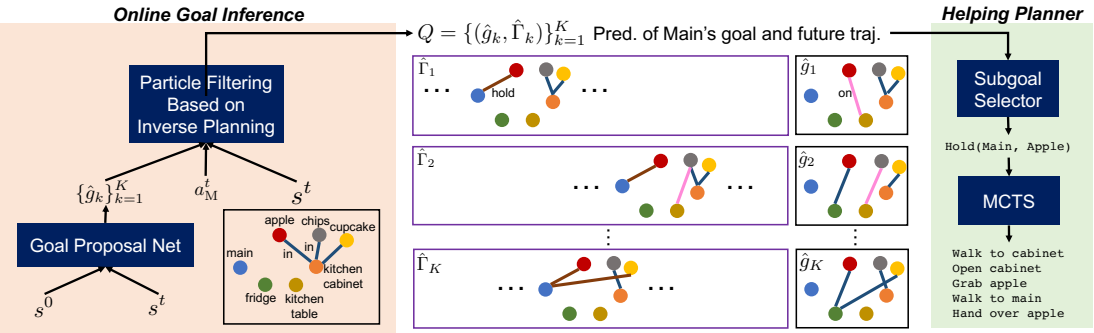


Figure 7-2: Overview of our approach, which consists of an online goal inference module and a helping planner. We represent states and goals using scene graphs (see 2.3.2). Here, s^t is the state at time t; a_{M}^t is the main agent's action at time t; $\hat{g}_k$ is the k-th goal proposal; and $\hat{\Gamma}_k$ is the prediction of the main agent's future trajectory corresponding to $\hat{g}_k$.

finished goals, as seen in Chapter 6. For this, some prior work devised planners under uncertain goal inference in simple environments [126]. There has also been a recent study proposing a goal-agnostic assistance framework via empowerment [37], which aims at changing the states to maximize an agent's ability to reach as many goals as possible regardless of its true intent. Despite its success in certain domains, assisting humans in the real-world settings without the knowledge of their goals would often result in counterproductive behaviors. Our work investigates how to design an uncertainty-aware planner that intelligently adjusts the helping behavior ranging from goal-agnostic strategies to goal-specific plans in a complex environment.

7.3 Neurally-guided Online Probabilistic Assistance

7.3.1 Problem setup

We define the online assistance problem as a mixed-observability Markov decision process (MOMDP) [105], where a ***helper*** agent needs to infer a ***main*** agent' goal and help the main agent achieve its goal faster. This can be formalized by the tuple $\langle \mathcal{S}, \mathcal{G}, \mathcal{A}_{\mathrm{H}}, \mathcal{O}, \mathcal{T}_{\mathcal{S}}, \mathcal{T}_{\mathcal{G}}, Z, R_{\mathrm{H}}, \gamma \rangle$. The overall state has two components: the world state, $s \in \mathcal{S}$, which is fully observable to the helper agent, and the main agent's goal, $g \in \mathcal{G}$, which is partially observable to the helper agent. $\mathcal{A}_{\mathrm{H}}$ is the action space of the helper agent. The helper agent's observation consists of the world state and the main agent's

Algorithm 1 NOPA
Given the main agent's actions $a_M^{\leq t}$ and environment states $s^{\leq t}$ up to time t, infers main agent's goals and commands helper actions a_H^t to assist on those.

Input: $\Gamma_{\mathrm{M}}^0 = \{(s^0, a_{\mathrm{M}}^0)\}$, s^t, K, $T_{\max}$, T_{prop}, q, w_r, w_c, w_m, $L_{\max}$
$t \leftarrow 1$, $l \leftarrow 0$
$Q \leftarrow \emptyset$
repeat
    $Q, l \leftarrow$ **GoalInf**$(t, Q, \Gamma_{\mathrm{M}}^{t-1}, s^t, q, K, l, T_{\text{prop}})$
    $\Gamma_{\mathrm{H}}^t \leftarrow$ **HelpPlanner**$(Q, s^0, s^t, w_r, w_c, w_m, L_{\max})$
    Execute the first action from the helping plan a_{H}^t
    Observe $a_{\mathrm{M}}^t, s^{t+1}$ from the environment
    $t \leftarrow t + 1$
until $t = T_{\max}$ or the true goal has not been reached

action, i.e., $\mathcal{O} = \mathcal{S} \times \mathcal{A}_{\mathrm{M}}$. $\mathcal{T}_{\mathcal{S}}(s, g, a_{\mathrm{H}}, s') = p(s'|s, g, a_{\mathrm{H}})$ is the transition function for the world state, and $\mathcal{T}_{\mathcal{G}}(s, g, a_{\mathrm{H}}, s', g') = p(g'|s, g, a_{\mathrm{H}}, s')$ is the transition function for the goal. $Z(s', g', a_{\mathrm{H}}, o) = p(o = (s, a_{\mathrm{M}})|s', g', a_{\mathrm{H}})$ is the conditional probability function for the observation result. At step $t+1$, the helper infers the main's goal given main's past trajectory upon t, i.e., $\Gamma^t = \{(s^\tau, a_{\mathrm{M}}^\tau)\}_{\tau=1}^t$. The expected reward function for the helper agent is defined as $R_{\mathrm{H}}(s, a|\Gamma^t) = E_{p(g|\Gamma^t)}[\mathbb{1}(s = g)] - c_{\mathrm{H}}(a)$, where $c_{\mathrm{H}}(a)$ is the cost for action a, and $\mathbb{1}(\cdot)$ checks if the goal is satisfied in the current world state s. γ is the discount factor. Note that in this work, we assume full observability for both agents, which is a common setting in prior work on cooperation with unknown goals [86, 52, 64]. It is possible to extend the setting to partial observability in the future where the helper agent needs to decide when to follow and watch the main agent and when to execute physical actions.

7.3.2 Method Overview

To solve the online assistance problem formalized above, we propose NOPA (Neurally-guided Online Probabilistic assistance). Figure 7-2 provides an overview of NOPA, showing the two main components: i) Neurally-guided online goal inference, and ii) an uncertainty-aware helping planner. As sketched in Algorithm 1, NOPA updates a set of particles conditioned on observed states and Main agent's actions. Each particle

includes prediction of both the final goal and the main agent's future trajectory leading to the corresponding goal. Common assistance frameworks [86, 52, 64] typically only consider the final goal for helping. However, when there is unceratinty in the goal inference, an intelligent AI assistant should seek intermediate subgoals that can be helpful with high certainty. For that, we predict the main agent's future trajectory leading to the predicted final goal for each particle. We represent both intermediate states and final goals as a set of edges in a scene graph [67], $\langle O, E \rangle$, as shown in Figure 7-2. Each node, $o \in O$, represents an entity (agents or objects); and each edge, $e \in E$, corresponds to a predicate (e.g., `IN(apple, kitchencabinet)`), indicating the spatial relationship between two entities. Such representations have been widely adopted in a broad range of robotics and embodied AI tasks [164, 101, 174, 139]. Given the particles at each step, the uncertainty-aware helping planner assesses the value of the edges that appear either in the final goals or in the intermediate states of the predicted trajectories. The most valuable edge is then selected as the helping subgoal. We introduce the two components in the remaining of this section and provide more implementation details in the supplementary material.

7.3.3 Neurally-guided Online Goal Inference

Unlike prior work on online goal inference, the main objective of the online goal inference in this work is to help the downstream task, i.e., assisting the main agent in achieving the intended goal. This introduces additional challenges: (1) the helper agent has to estimate the uncertainty in the inference instead of only predicting the most probable goal; (2) it has to ensure that the inference is resilient in a dynamic environment; and (3) the inference has to be efficient so that the helper can have a prompt reaction to offer assistance. For this, we propose a neurally-guided online goal inference algorithm as summarized in Algorithm 2, which combines inverse planning and a neural network. This hybrid approach takes advantage of both types of goal inference approaches.

Neurally-guided Goal Proposals. We use a goal proposal network (GPN) to learn a proposal distribution $q(g|s^0, s^t)$, from which we can sample K goal proposals

Algorithm 2 GoalInf

Computes a set of goal proposals Q', given previous goal proposals Q and main agent's actions and environment states, up to time $t-1$, $\Gamma_{\mathrm{M}}^{t-1}$.

1: **Input:** t, Q, $\Gamma_{\mathrm{M}}^{t-1} = \{(s^\tau, a_{\mathrm{M}}^\tau)\}_{\tau=0}^{t-1}$, s^t, q, K, l, T_{prop}
2: **Output:** Updated proposals Q' and steps since last proposal sampling l'
3: $Q' \leftarrow \emptyset$
4: **if** $Q \neq \emptyset$ and $l < T_{\mathrm{prop}}$ **then**
5: **for** $k = 1, \cdots, |Q|$ **do**
6: **if** a_{M}^{t-1} is part of the plan $\hat{\Gamma}_k$ **then**
7: $Q' \leftarrow Q' \cup \{(\hat{g}_k, \hat{\Gamma}_k)\}$
8: **end if**
9: **end for**
10: **end if**
11: **if** $Q' = \emptyset$ **then** ▷ Build proposals
12: **for** $k = 1, \cdots, K$ **do**
13: $\hat{g}_k \sim q(g|s^0, s^t)$ ▷ Sample a goal proposal
14: $\hat{\Gamma}_k \leftarrow \mathbf{MCTS}(s^t, \hat{g}_k, T_{\mathrm{prop}})$ ▷ Sample a plan
15: $Q' \leftarrow Q' \cup \{(\hat{g}_k, \hat{\Gamma}_k)\}$
16: **end for**
17: $l' \leftarrow 0$
18: **else**
19: $l' \leftarrow l + 1$
20: **end if**
21: **return** Q', l'

$\{\hat{g}_k\}_{k=1}^K$ given the initial state s^0 and the current state s^t (Algorithm 2, line 13). Each goal proposal is a set of goal predicates. Here, we only consider the first state and the current state instead of a sequence of past states for the input to GPN so that the GPN trained on episodes with only the main agent performing the tasks can still be robustly applied to the helping condition where the sequence of state changes could become drastically different from the training sequences.

Rejection and Resampling. Prior work [100, 128, 44] has demonstrated that the goal prediction from feedfoward networks may not be able to reliably assess the uncertainty in the inference and often fail to accurately predict the goals in unseen environments. Thus, instead of directly using the goals proposed by the GPN, we use inverse planning to evaluate these proposals and reject the ones that are inconsistent with the observed main agent's actions. To model an agent's behavior given each goal $\hat{g}_k$ with bounded rationality, we use the built-in planner to predict future trajectory of the main agent in the next T_{prop} steps $\hat{\Gamma}_k = \{(\hat{s}^{t+\tau_k}, \hat{a}^{t+\tau_k})\}_{\tau=1}^{T_{\mathrm{prop}}}$. Specifically, the level

of rationality can be adjusted by the number of simulations and the length of rollouts. We then create K particles $Q = \{\hat{g}_k, \hat{\Gamma}_k\}$ (Algorithm 2, line 13-14). Whenever we observe a new action from the main agent, a^t_{M}, we check if it is part of the predicted plan for each particle (Algorithm 2, line 5-9). If for a particle k, the action is not included in the predicted plan, then it suggests that the rational behavior under the corresponding goal is not consistent with the observed action. Thus this particle is likely to have an incorrect goal and needs to be rejected. When there is no particle left or we have reached the prediction horizon T_{prop}, we resample another K goals from the GPN based on the latest state and create new particles.

7.3.4 Uncertainty-aware Helping Planner

Our helping planner (Algorithm 3) considers all edges that appear in the final goal and the intermediate states in the predicted Main agent's plans as candidate helping subgoals. Additionally, the helper agent may find that the objects it grabs are no longer needed when it updates its goal inference or after the main agent achieves the corresponding subgoals. To allow the helper agent to return those objects to their initial locations, the helping subgoal space also includes edges in the initial state (Algorithm 3, line 9-10). For each edge e, we estimate how long it would take the main agent to reach that subgoal, $L_{\mathrm{M}}(e)$. If this edge appears in one of the predicted trajectories in the particles, we can conveniently estimate $L_{\mathrm{M}}(e)$ based on when it appears in the trajectories (Algorithm 3, line 15-16). If it only appears in the final goals, we then use a fixed length, $L_{\max}$, to anticipate how many steps it would take the main agent to reach that subgoal (Algorithm 3, line 18). We can also use MCTS to search for a plan for the helper agent, $\Gamma_{\mathrm{H}}(e)$, to reach the same subgoal (Algorithm 3, line 25-26). Let $L_{\mathrm{H}}(e)$ be the length of the helper's plan, we then define the benefit of helping with subgoal e as the speed up the helper agent can offer by reaching the subgoal e, i.e., $\max(L_{\mathrm{M}}(e) - L_{\mathrm{H}}(e), 0)$. To account for the uncertainty in inference, we estimate how likely e is going to be necessary, $p(e)$, by counting how many particles include e in either the intermediate states or the final goal (Algorithm 3, line 16, 19).

Algorithm 3 HelpPlanner

Given a set of goal proposals Q and the initial and current state of the environment, s_0, s_t, computes a plan for the helper agent Γ_{H}^t.

Input: Q, s^0, s^t, w_r, w_c, w_m, $L_{\max}$
Output: helping plan Γ_{H}^t
for $e \in \mathcal{E}$ **do**
    $L_{\mathrm{M}}(e) \leftarrow \infty$
    $p(e) \leftarrow 0$
    $V(e) \leftarrow -\infty$
    $\Gamma_{\mathrm{H}}(e) \leftarrow \emptyset$
    if e appears in the initial state **then** ▷ Subgoals for restoring the initial state
        $L_{\mathrm{M}}(e) \leftarrow 0$
        $p(e) \leftarrow 1$
    else
        for $k = 1, \cdots, |Q|$ **do** ▷ Estimate when the main agent can achieve subgoal e
            if e appears in the predicted states in $\hat{\Gamma}_k$ **then**
                Let $\tau_k(e)$ be the first step when e appears in the predicted states
                $L_{\mathrm{M}}(e) \leftarrow \min(L_{\mathrm{M}}(e), \tau_k(e))$
                $p(e) \leftarrow p(e) + 1/|Q|$
            else if e appears in the predicted goal $\hat{g}_k$ **then**
                $L_{\mathrm{M}}(e) \leftarrow \min(L_{\mathrm{M}}(e), L_{\max})$
                $p(e) \leftarrow p(e) + 1/|Q|$
            end if
        end for
    end if
    if $p(e) > 0$ **then**
        $\Gamma_{\mathrm{H}}(e) \leftarrow \mathbf{MCTS}(s^t, \{e\}, \infty)$ ▷ Compute helper's plan for subgoal e
        $L_{\mathrm{H}}(e) \leftarrow |\Gamma_{\mathrm{H}}(e)|$
        Compute $V(s^t, e)$ as Eq (7.1) ▷ Compute the value of subgoal e
    end if
end for
$e^* \leftarrow \arg\max V(s^t, e)$ ▷ Obtain subgoal with maximum value
return $\Gamma_{\mathrm{H}}(e^*)$ ▷ Obtain plan for subgoal e^*

Finally, we define a value function for selecting the best subgoal for the helper agent:

$$V(s^t, e) = w_r p(e) \max(L_{\mathrm{M}}(e) - L_{\mathrm{H}}(e), 0) - w_c L_{\mathrm{H}}(e) - w_m(\mathcal{D}(s^0, \hat{s}(e)) - \mathcal{D}(s^0, s^t)), \tag{7.1}$$

where w_r, w_c, and w_m are constant weights; $\mathcal{D}$ measures the difference between two states; and $\hat{s}(e)$ is the state after reaching the subgoal e from the current state s^t. Intuitively, the three terms of the value function evaluate i) the expected benefit of helping reach the subgoal, ii) the cost of the helper agent, and iii) the additional state change (compared with the initial state) introduced by the subgoal. These three

terms make sure that the helper agent selects a subgoal that i) is likely to speed up the task with high certainty, ii) is not too costly for the helper agent, and iii) could restore the initial states of objects that are not needed for the task respectively. After selecting the most valuable subgoal e^*, we execute the first action of the helping plan $\Gamma_{\mathrm{H}}(e^*)$.

7.4 Online Watch-And-Help

Following the problem setup defined as Section 7.3.1, we present Online Watch-And-Help (O-WAH), a new embodied human-AI cooperation challenge, in which a helper agent has to infer a main agent's goal and help reach the goal as fast as possible. This extends Watch-And-Help (WAH), presented in Chapter 6, to an online assistance problem. Like WAH, O-WAH is built in VirtualHome, focusing on household activities in indoor environments. Unlike WAH, the helper agent in O-WAH no longer watches a demonstration of the main agent, but instead has to simultaneously watch main's action, infer its goal, and decide i) whether to help and ii) how to help at each step.

Task Definitions. Similar to WAH, the goal for each task is defined by a set of predicates and their counts, representing the target locations of different objects in the environment. We sample each goal in the challenge from five general types of household tasks: *set table*, *put dishwasher*, *stock fridge*, *prepare meal*, and *get snacks*. Note that we define task types only to ensure that the goals are emulating real-life household tasks, but that this information is not provided to the helper agents. As summarized in Table 2 in the supplementary material, different kinds of uncertainty may arise from these tasks: i) uncertainty in the number of objects (e.g., the main agent may want to set up a table for 1 to 3 persons), ii) uncertainty in which objects are needed (e.g., the main agent may want to put different foods to the table for preparing a meal), and iii) uncertainty in the target locations (e.g., the main agent could put dining pieces to the kitchen table or the coffee table).

Training and Testing Sets. To create a training episode, we first sample a goal

and an initial environment using one of the five training apartments and then use a built-in planner to control the main agent to perform the task alone. The built-in planner is the same hierarchical planner as in Chapter 6. We create a large training set with 6,000 episodes and a small training set with 300 episodes. The testing set has 100 episodes in the two testing apartments unseen during training.

Evaluation Metrics. We use F1-score over the goal predicates to measure the goal inference accuracy. To evaluate the helping performance, we use speedup, where we compare the episode length when the helper agent works with the main agent (L_{H}) against the episode length when the main agent works alone (L_{M}), i.e., $L_{\mathrm{M}}/L_{\mathrm{H}} - 1$. For each episode, set a time limit of 250 steps and report the average performance across 3 runs.

7.5 Experiments

7.5.1 Baselines

To evaluate the efficacy of our approach, NOPA, we compare it against several baselines. Note that for all approaches that propose multiple goals, we use 20 proposals.

HP$_{\mathbf{GPN}}$: We adopt the best performing approach in the original Watch-And-Help challenge for this baseline, which is a hierarchical planner (HP) based on the most probable goal according to the GPN. In particular, at each step, **HP$_{\mathbf{GPN}}$** uses the goal $\hat{g} = \arg\max_g q(g|s^0, s^t)$.

AF$_{\mathbf{GPN}}$: We extend **HP$_{\mathbf{GPN}}$** by using NOPA's online goal inference module. We generate a plan for each predicated goal using HP and execute the most frequent first action among all plans.

Empowerment: In this baseline, we adopt the idea of empowerment [37], where we uniformly sample K goals at each step, predict plans and intermediate states for the goals, and select the most frequent edge in the intermediate state as the helping subgoal (i.e., the most common subgoal for *any* goal).

HP$_{\mathbf{RG}}$: A hierarchical planner based on a randomly sampled goal at the beginning

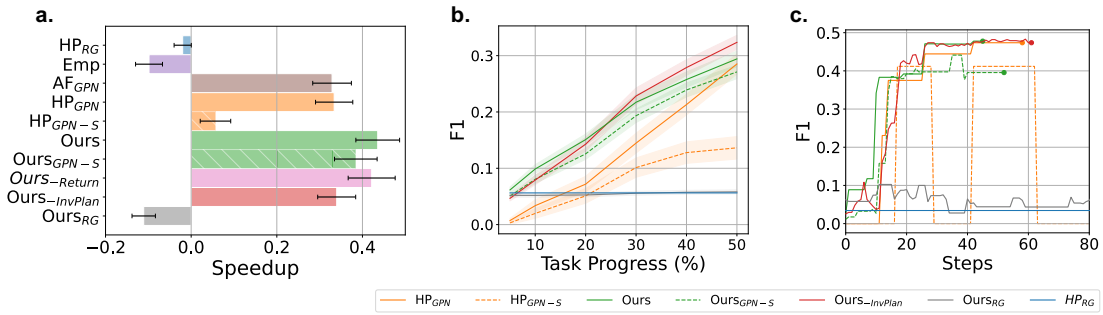


Figure 7-3: (**a**) Speedup of different methods (striped bars indicate using the small training set). Errors are standard errors. (**b**) F1-scores of the predicted goal over the course of a task. The x axis is normalized in proportion to the number of steps needed for the main agent to perform each task alone. The curves show the means and the shaded regions show the standard errors. **c)** F1-scores over time for different approaches in a single test episode, a dot indicates the number of steps a given baseline took to complete the task. The dashed lines in (**b**) and (**c**) indicate using the small training set.

of the episode.

We consider the following ablated methods to evaluate the effect of different components of NOPA.

Ours$_{\mathbf{RG}}$: We replace the proposal distribution q in Algorithm 2 with a uniform distribution.

Ours$_{\mathbf{-InvPlan}}$: Ours without using inverse planning to reject inconsistent particles.

Ours$_{\mathbf{-Return}}$: Ours without returning irrelevant objects to their initial locations ($w_m = 0$ in Eq.(7.1)).

By default, the GPN is trained on the large training set. To evaluate the sample efficiency of NOPA, we also report the performance of **Ours** and **HP$_{\mathbf{GPN}}$**, when the GPN is trained on a small training set, indicated by the subscript **GPN-S**. To measure the upper bound on the helping performance, we also implement an oracle helper **HP$_{\mathbf{GT}}$**, which knows the ground-truth goal and is controlled by an HP.

7.5.2 Results

Main Controlled by a Planner

We evaluate all methods with a main agent controlled by the built-in planner and report the helping speedup (average and standard error across episodes) in Figure 7-3**a**. For methods that have different goal inference modules, we also report the F1-score of their goal inference results in Figure 7-3**b**. The speedup of the oracle agent, operating with true knowledge about the goal, $\mathbf{HP_{GT}}$ is 1.29. NOPA (**Ours**) outperforms all baselines, offering the highest speedup. It also achieves the best goal inference accuracy at the early stage of the tasks, which serves the foundation of its successful assistance. This benefit can be more clearly seen from Figure 7-3**c** (the improvement margin appears to be smaller since the temporal normalization for each episode is different). The low speedup by **Empowerment** suggests that online goal inference is necessary for an effective assistance, despite its success in certain domains shown in prior work [37]. Given that the predicted goal may be uncertain, using multiple goal proposals leads to a better helping performance, as seen by comparing $\mathbf{Ours_{GPN}}$ with $\mathbf{HP_{GPN}}$. The effect is more pronounced when the GPN is trained with fewer data and are consequently less accurate (i.e., **GPN-S**). We also find that the neurally-guided goal proposals can greatly improve the goal inference over uniform goal proposals (i.e., $\mathbf{Ours_{RG}}$). Moreover, the results demonstrate that inverse planning is important for filtering spurious goal proposals from **GPN**, significantly improving the speedup over $\mathbf{Ours_{\text{-}InvPlan}}$ since it allows the goal inference to reach an relatively high accuracy much earlier than $\mathbf{Ours_{\text{-}InvPlan}}$ and other baselines do (see Figure 7-3**c**). Finally, by comparing NOPA with $\mathbf{Ours_{\text{-}Return}}$, we can see a marginal improvement in speedup by avoiding unnecessarily distorting the environment; $\mathbf{Ours_{\text{-}Return}}$ also causes unnecessary state changes in 70.1% of all testing episodes, while NOPA only yields unnecessary state changes in 58.9% of all testing episodes.

Figure 7-4 shows a typical successful example by NOPA, where the task is the same as the one in Figure 7-3**c**. It demonstrates that NOPA can i) achieve accurate goal inference early on by filtering out goal proposals that are inconsistent with the main

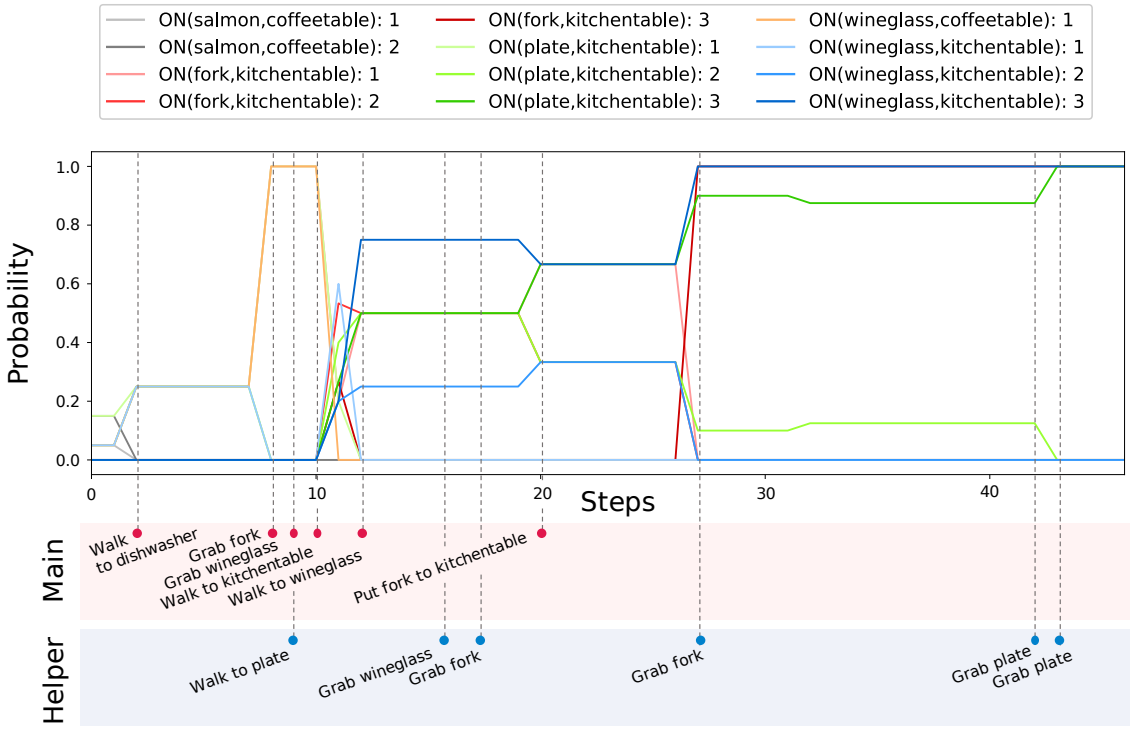


Figure 7-4: Goal inference and plans by NOPA for the task shown in Figure 7-3**c** (setting up a kitchen table for 3 persons). We show the posterior probabilities of the top predicates and their counts based on the particles at each step, key actions of the main agent (indicated by red dots), and key helping actions (indicated by blue dots). At step 2, after the main agent walks towards the dishwasher, NOPA rejects proposals involving nearby objects (e.g., apples, salmons) that are not inside of the dishwasher, increasing the probabilities for predicates about setting up a table. After the main agent grabs a fork at step 8, NOPA infers with high confidence that the goal is setting up the kitchen table for at least one person. So at the following step, the helper agent takes its very first action – walking to grab a plate. Upon seeing Main walking to the kitchen table at step 10, the coffee table is no longer considered as the goal location. After observing more actions, the inference converges to setting up the kitchen table for 3 persons.

agent's actions, ii) correctly update the goal inference and its uncertainly estimation based on more observation, iii) plan for effective helping actions based on the filtered goal proposals and the uncertainty in the inference. Note that the helper remains idle but takes a useful helping action as soon as the goal inference becomes confident and is able to avoid grabbing extra objects thanks to its gradual update of the number objects needed.

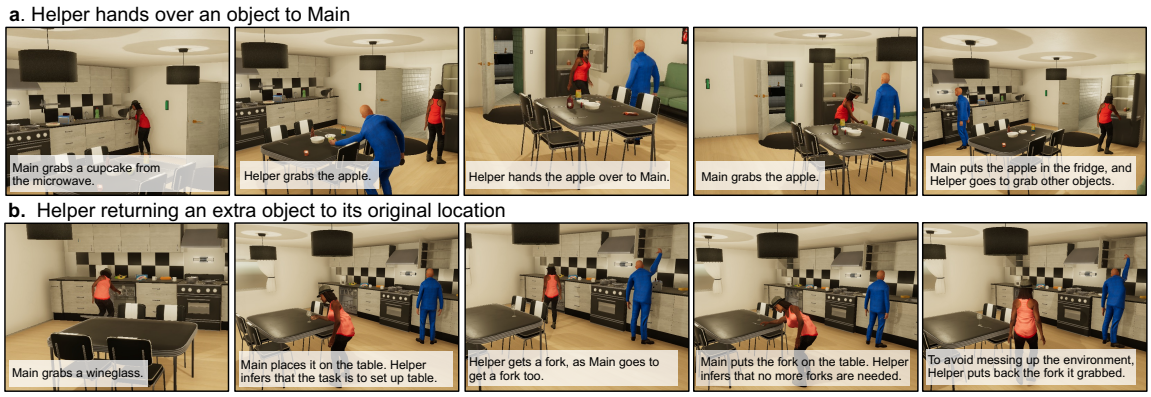


Figure 7-5: Examples of helping plans that are beyond directly achieving final goals. The main agent is in red, and the helper agent is in blue.

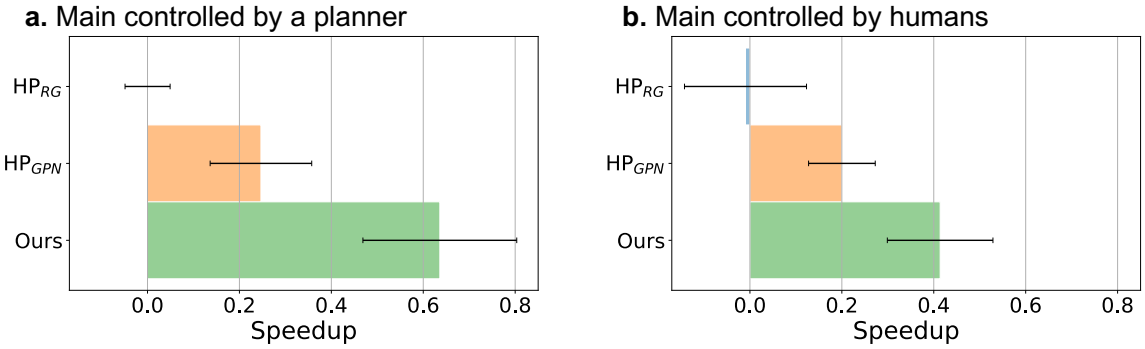


Figure 7-6: (**a**) Speedup of different methods when the main agent is controlled by the built-in planner. (**b**) Speedup of different methods when the main agent is controlled by human players. Note that all results are based on the same 10 testing episodes.

From the evaluation, we also observe diverse helping behaviors enabled by NOPA that are not just about directly achieving the inferred final goals. We show two types of behaviors in Figure 7-5. First, the helper agent sometimes select a subgoal of handing over objects to the main agent. For example, as illustrated in Figure 7-5**a**, the helper agent hands over the apple to the main agent who is right next to the fridge so that the task execution can be faster. Second, the helper agent can return extra objects to their initial locations once it realizes that they are not needed for reaching the goal (Figure 7-5**b**).

Main Controlled by Humans

To evaluate how effective AI helper agents are at assisting real humans, we conducted a human experiment where the main agent is controlled by human players. We used 10 testing episodes to run 40 trials. In each trial, a human participant was asked to either perform the task alone (to estimate the number of steps needed for completing each task without a helper agent) or work with an AI agent controlled by one of the three approaches, NOPA, $\mathbf{HP_{GPN}}$, and $\mathbf{HP_{RG}}$. Note that participants did not know which helper agent they were working with. In total, we recruited 10 participants. The participants gave informed consent, and the experiment was approved by an institutional review board.

As shown in Figure 7-6, the ranking of the methods remains consistent when the main agent is controlled by human players instead of the built-in planner. We also find that there is no significant difference in NOPA's performance under the two conditions ($t(9) = 0.87$, $\rho = 0.40$). More details about the statistical tests can be found in the supplementary materials.

7.6 Discussion

In this chapter, we aim to build agents that can help humans with no prior knowledge about what the goal they need to help with. For this, we propose a novel online assistance method, Neurally-guided Online Probabilistic Assistance (NOPA), which integrates (1) a hybrid online goal inference algorithm combining a goal proposal network and inverse planning-based particle filtering and (2) an uncertainty-aware helping planner that identifies valuable helping subgoals from both the final goals and intermediate states. For a systematic and scalable evaluation, we extend Watch-And-Help and introduce a new embodied human-AI cooperation challenge, Online Watch-And-Help, in which a helper needs to simultaneously watch a main agent's action, infer its goal, and help reach the goal as fast as possible for complex daily household tasks. We evaluate NOPA in this challenge, together with several baselines, using a main agent controlled either by a built-in planner or by humans. Our experiment

shows that NOPA significantly outperforms baselines in online goal inference as well as in helping both humans and human-proxy AI driven agents.

NOPA is currently evaluated on embodied human-AI cooperation problems in home-like environments. However, the principle behind NOPA should be applicable to a broader range of domains, as long as goals can be defined appropriately. A limitation in this setting is that we assumed that both agents had full observability over the environment, where not only both agents cannot see the full physical state, they also cannot always see each other. This is a more restrictive setting than what we explored in Chapter 6. While this is an exciting direction for future work, it will also introduce new challenges, making goal inference harder (i.e. observing the main agent searching for objects gives little information about their goal), as well as assistance, requiring an active information seeking behavior besides the present watch-and-react paradigm.

In the last two chapters of this thesis, we focused on building agents that could infer human goals and assist them in performing these goals effectively in a simulated environment. We designed benchmarks and baselines to evaluate progress in this task, and developed AI-driven agents that could simulate human-like behavior to test collaboration at scale. We further tested our methods with real humans, showing that these benchmarks can serve as a reasonable proxy for human-AI collaboration.

Moving from agents that perform activities in an environment to agents that interact with humans in the environment is crucial if we want to build effective AI assistants, and the third part of this thesis aimed to take a step towards this goal. Despite promising results, many challenges remain. Human assistance involves more challenges than helping complete a set of goals as quickly as possible, and effective agents should also reason about human preferences in the way a task should be done, safety, social norms, etc. In this sense, humans can exhibit a wide range of behaviors when attempting to complete a goal, and thus we must be cautious in developing systems and training datasets that can cover a wide range of demographics, to mitigate biases and risks, ensuring a beneficial outcome to a wider range of population.

Chapter 8

Conclusion

In this thesis, we have studied how to build agents that could assist humans in everyday activities. We argued about the necessity to test these agents in simulation before deploying them into the real world, and described a roadmap to make progress in building them. In Part I, we talked about the properties that a simulator should have in order to enable the study of AI assistants, and proposed the VirtualHome simulator and a knowledge base to study these problems. In Part II, we studied agents that could perform activities alone in the environment, given human instructions or demonstrations and in Part III we developed benchmarks and models to build collaborative agents that could assist humans in activities while these were interacting in the environment.

While this work focused on household environments and activities, our proposed approach could be applied to more general scenarios, or even collaboration settings that do not necessarily happen in a physical space. These agents could be adapted to help people search for information on the web, make a design or as avatars that could assist humans in virtual reality.

Simulators can enable agents to learn complex behaviors at scale and test them safely through a wide range of tasks and scenarios, but ultimately we care about agents that can assist humans in the real world. Using simulation to improve real world robots has proven successful in multiple problems and domains [150, 76, 8] and I believe that it will also be instrumental in building robots that can assist us at

everyday tasks. For this, there are a number of challenges we need to address.

More realistic and general environments. In order to ensure that behaviors in simulation are representative of the real world, we need to reduce the gap between these two. Part of these efforts will involve developing algorithms that can be robust to the shift between domains [150, 151], but it will also be key to make progress in building better simulation tools. Most of current simulators either focus on improving rendering towards photo-realism[121, 120], building accurate physical models for manipulation [8, 161] or providing a wide set of objects and actions for task planning. As humans, we need to reason at different levels when performing activities, and if we want robots to learn this via simulation, we need to be able to unify these different levels of abstraction. While building more realistic environments can help bridge the *sim2real* gap, to ensure that we build robust models, we need to be able to represent diversity in the environment. In this thesis we tested our agents in 7 predefined apartments in VirtualHome, with diversity coming from the placement of objects and lighting conditions. We are extending our simulator to support procedural generation of environments, allowing to represent potentially infinitely unique apartments to test our agents. Furthermore, here we focused on household activities, but in real life we do everyday tasks in our households, out in the city, and a combination of the two. Being able to represent a wide range of domains will be key if we want agents that can help us, get groceries to prepare a dinner with friends at home.

Better human models and agents. As humans, we spend as much time, if not more, learning to interact with other humans as with the environment around us. If we want to build agents that can learn to interact with us, we need to provide better human models that can be run in simulation. In Part III of this thesis, we presented a planning-based approach to simulate humans that would perform actions to accomplish goals, where a belief model would allow them to look for objects when they were not visible in the environment. However, as humans we don't only take actions based on a set of task goals; we also consider our preferences and capabilities, as well as those of other individuals or society, sometimes represented as non-written

norms. We need to build models that can incorporate those factors, and for this, it will be necessary to have 1) more flexible human models, and 2) better datasets, so that we can learn those behaviors that are difficult to formalize in a planner. In this sense, there is a great opportunity in incorporating data-driven human body models [153, 106, 53, 170, 16] into the simulator, allowing to represent more realistic human shapes and motions while still being able to support long-term behaviors. Similarly, we need to think about how to collect effectively datasets of humans to represent these behavior, showing these humans, not only interacting with the environment, but also collaborating among them. Section C.5.4 in Appendix C show our initial efforts towards these goals.

Better metrics. How do we measure the effectiveness of assistive agents? In this work we focused on measuring the accuracy with which agents executed the intended tasks and the speed-up they offered when collaborating with humans in performing those. However, we also want to build agents that we feel safe around, that can be trusted and are reliable. Qualitative human studies can help us evaluate these properties, but we need to think how these can be measured in an automatic way, so that the can be incorporated throughout the development of assistive agents.

Building general-purpose assistive agents can have profound societal impact, and designing the right platforms, algorithms and metrics will be key in making sure we reach this goal safely and effectively. We hope that the work presented in this thesis can inspire future research towards this important goal.

Bibliography

[1] `https://scratch.mit.edu/`.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[3] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[4] J.-B. Alayrac, P. Bojanowski, N. Agrawal, I. Laptev, J. Sivic, and S. Lacoste-Julien. Unsupervised learning from narrated instruction videos. In *CVPR*, 2016.

[5] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

[6] Muhannad Alomari, Paul Duckworth, Majd Hawasly, David C Hogg, and Anthony G Cohn. Natural language grounding and grammar induction for robotic manipulation commands. In *Proceedings of the First Workshop on Language Grounding for Robotics*, pages 35–43, 2017.

[7] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Spice: Semantic propositional image caption evaluation. In *European Conference on Computer Vision*, pages 382–398. Springer, 2016.

[8] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[9] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[10] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021.

[11] Chris L Baker, Julian Jara-Ettinger, Rebecca Saxe, and Joshua B Tenenbaum. Rational quantitative attribution of beliefs, desires and percepts in human mentalizing. *Nature Human Behaviour*, 1(4):1–10, 2017.

[12] Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J Tomlin. Goal-driven dynamics learning via bayesian optimization. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5168–5173. IEEE, 2017.

[13] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

[14] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020.

[15] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic roommates making pancakes. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 529–536. IEEE, 2011.

[16] Bharat Lal Bhatnagar, Xianghui Xie, Ilya Petrov, Cristian Sminchisescu, Christian Theobalt, and Gerard Pons-Moll. Behave: Dataset and method for tracking human object interactions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[17] Philipp Bomatter, Mengmi Zhang, Dimitar Karev, Spandan Madan, Claire Tseng, and Gabriel Kreiman. When pigs fly: Contextual reasoning in synthetic and natural scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 255–264, 2021.

[18] Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. *arXiv preprint arXiv:1605.06640*, 2016.

[19] Damien Bouchabou. *Human activity recognition in smart homes: tackling data variability using context-dependent deep learning, transfer learning and data synthesis*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire, 2022.

[20] Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron C. Courville. Home: a household multimodal environment. *CoRR*, abs/1711.11017, 2017.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[22] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[23] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 961–970, 2015.

[24] Zhe Cao, Hang Gao, Karttikeya Mangalam, Qi-Zhi Cai, Minh Vo, and Jitendra Malik. Long-term human motion prediction with scene context. In *European Conference on Computer Vision*, pages 387–404. Springer, 2020.

[25] Sandra Carberry. Techniques for plan recognition. *User modeling and user-adapted interaction*, 11(1):31–48, 2001.

[26] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems*, pages 5175–5186, 2019.

[27] Tathagata Chakraborti, Gordon Briggs, Kartik Talamadupula, Matthias Scheutz, David Smith, Subbarao Kambhampati, and HRI Laboratory. Planning for serendipity-altruism in human-robot cohabitation.

[28] Yu-Wei Chao, Chris Paxton, Yu Xiang, Wei Yang, Balakumar Sundaralingam, Tao Chen, Adithyavairavan Murali, Maya Cakmak, and Dieter Fox. Handoversim: A simulation framework and benchmark for human-to-robot object handovers. *arXiv preprint arXiv:2205.09747*, 2022.

[29] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

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

[32] Wongun Choi and Silvio Savarese. Understanding collective activitiesof people from videos. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1242–1257, 2013.

[33] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. In *Conference on Robot Learning*, pages 885–897. PMLR, 2020.

[34] Kerstin Dautenhahn. Socially intelligent robots: dimensions of human–robot interaction. *Philosophical transactions of the royal society B: Biological sciences*, 362(1480):679–704, 2007.

[35] Patrick Dendorfer, Aljosa Osep, and Laura Leal-Taixé. Goal-GAN: Multimodal trajectory prediction based on goal position estimation. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[36] Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793*, 2018.

[37] Yuqing Du, Stas Tiomkin, Emre Kiciman, Daniel Polani, Pieter Abbeel, and Anca Dragan. AVE: Assistance via empowerment. *Advances in Neural Information Processing Systems*, 33:4560–4571, 2020.

[38] Zackory Erickson, Vamsee Gangaram, Ariel Kapusta, C. Karen Liu, and Charles C. Kemp. Assistive gym: A physics simulation framework for assistive robotics. *CoRR*, abs/1910.04700, 2019.

[39] Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A decision-theoretic model of assistance. *Journal of Artificial Intelligence Research*, 50:71–104, 2014.

[40] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368, 2017.

[41] David F Fouhey, Wei-cheng Kuo, Alexei A Efros, and Jitendra Malik. From lifestyle vlogs to everyday interactions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4991–5000, 2018.

[42] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

[43] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel LK Yamins, James J DiCarlo, Josh McDermott, Antonio Torralba, et al. The threedworld transport challenge: A visually guided task-and-motion planning benchmark towards physically realistic embodied ai. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8847–8854. IEEE, 2022.

[44] Kanishk Gandhi, Gala Stojnic, Brenden M Lake, and Moira R Dillon. Baby Intuitions Benchmark (BIB): Discerning the goals, preferences, and actions of others. *Advances in Neural Information Processing Systems*, 34:9963–9976, 2021.

[45] Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. Vrkitchen: an interactive 3d virtual environment for task-oriented learning. *arXiv*, abs/1903.05757, 2019.

[46] Alexander L Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

[47] Michael A Goodrich and Alan C Schultz. Human-robot interaction: a survey. *Foundations and trends in human-computer interaction*, 1(3):203–275, 2007.

[48] Barbara Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 1996.

[49] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[50] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *PoPL'11, January 26-28, 2011, Austin, Texas, USA*, January 2011.

[51] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6244–6251. IEEE, 2018.

[52] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pages 3909–3917, 2016.

[53] Mohamed Hassan, Duygu Ceylan, Ruben Villegas, Jun Saito, Jimei Yang, Yi Zhou, and Michael Black. Stochastic scene-aware motion prediction. In *Proceedings of the International Conference on Computer Vision 2021*, October 2021.

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

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[56] Guy Hoffman. Evaluating fluency in human–robot collaboration. *IEEE Transactions on Human-Machine Systems*, 49(3):209–218, 2019.

[57] Gabriel Hoffmann, Haomiao Huang, Steven Waslander, and Claire Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *AIAA guidance, navigation and control conference and exhibit*, page 6461.

[58] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

[59] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration, 2018.

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

[61] Mostafa S Ibrahim, Srikanth Muralidharan, Zhiwei Deng, Arash Vahdat, and Greg Mori. A hierarchical deep temporal model for group activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1971–1980, 2016.

[62] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[63] Julian Jara-Ettinger. Theory of mind as inverse reinforcement learning. *Current Opinion in Behavioral Sciences*, 29:105–110, 2019.

[64] Shervin Javdani, Henny Admoni, Stefania Pellegrinelli, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research*, 37(7):717–742, 2018.

[65] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization. *Robotics science and systems: online proceedings*, 2015, 2015.

[66] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *arXiv:1705.03633*, 2017.

[67] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3668–3678, 2015.

[68] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247, 2016.

[69] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[70] Henry A Kautz. A formal theory of plan recognition and its implementation. *Reasoning about plans*, pages 69–125, 1991.

[71] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[72] Ryan Kiros, Yukun Zhu, Russ Salakhutdinov, Richard Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *NIPS*, 2015.

[73] Kris M Kitani, Brian D Ziebart, James Andrew Bagnell, and Martial Hebert. Activity forecasting. In *European Conference on Computer Vision*, pages 201–214. Springer, 2012.

[74] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: an interactive 3d environment for visual AI. *CoRR*, abs/1712.05474, 2017.

[75] Richard E Korf. Planning as search: A quantitative approach. *Artificial intelligence*, 33(1):65–88, 1987.

[76] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *RSS*, 2021.

[77] Przemyslaw A Lasota, Terrence Fong, Julie A Shah, et al. *A survey of methods for safe human-robot interaction*, volume 104. Now Publishers Delft, The Netherlands, 2017.

[78] Youngwoon Lee, Edward S Hu, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[79] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[80] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Elliott Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. In *5th Annual Conference on Robot Learning*, 2021.

[81] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making. *CoRR*, abs/2202.01771, 2022.

[82] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*, 2022.

[83] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

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

[85] Bingbin Liu, Ehsan Adeli, Zhangjie Cao, Kuan-Hui Lee, Abhijeet Shenoi, Adrien Gaidon, and Juan Carlos Niebles. Spatiotemporal relationship reasoning for pedestrian intent prediction. *IEEE Robotics and Automation Letters*, 5(2):3485–3492, 2020.

[86] Chang Liu, Jessica B Hamrick, Jaime F Fisac, Anca D Dragan, J Karl Hedrick, S Shankar Sastry, and Thomas L Griffiths. Goal inference improves objective and perceived performance in human-robot collaboration. In *15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016.

[87] Salvatore Loreto and Simon Pietro Romano. *Real-time communication with WebRTC: peer-to-peer in the browser*. " O'Reilly Media, Inc.", 2014.

[88] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*, 2021.

[89] Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. Improving vision-and-language navigation with image-text pairs from the web. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[90] Karttikeya Mangalam, Yang An, Harshayu Girase, and Jitendra Malik. From goals, waypoints & paths to long term human trajectory forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15233–15242, 2021.

[91] Zohar Manna and Richard J Waldinger. Toward automatic program synthesis. *Communications of the ACM*, 14(3):151–165, 1971.

[92] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *CoRR*, abs/1904.12584, 2019.

[93] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences, 2015.

[94] Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

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

[96] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[97] Vijayaraghavan Murali, Letao Qi, Swarat Chaudhuri, and Chris Jermaine. Neural sketch learning for conditional program generation. *arXiv preprint arXiv:1703.05698*, 2017.

[98] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2146–2153. IEEE, 2017.

[99] Zhixiong Nan, Tianmin Shu, Ran Gong, Shu Wang, Ping Wei, Song-Chun Zhu, and Nanning Zheng. Learning to infer human attention in daily activities. *Pattern Recognition*, 103:107314, 2020.

[100] Aviv Netanyahu, Tianmin Shu, Boris Katz, Andrei Barbu, and Joshua B Tenenbaum. PHASE: Physically-grounded abstract social events for machine social perception. In *35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[101] Son Nguyen, Ozgur S Oguz, Valentin N Hartmann, and Marc Toussaint. Self-supervised learning of scene-graph representations for robotic sequential manipulation planning. In *CoRL*, pages 2104–2119, 2020.

[102] Yinyu Nie, Angela Dai, Xiaoguang Han, and Matthias Nießner. Pose2room: Understanding 3d scenes from human activities. *arXiv preprint arXiv:2112.03030*, 2021.

[103] Stefanos Nikolaidis, Ramya Ramakrishnan, Keren Gu, and Julie Shah. Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. In *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 189–196. IEEE, 2015.

[104] Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 33–40. IEEE, 2013.

[105] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[106] Priyanka Patel, Chun-Hao P. Huang, Joachim Tesch, David T. Hoffmann, Shashank Tripathi, and Michael J. Black. AGORA: Avatars in geography optimized for regression analysis. In *Proceedings IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

[107] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 07 2020.

[108] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Trans. Graph.*, 37(6), November 2018.

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

[110] Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B Tenenbaum, Sanja Fidler, and Antonio Torralba. Watch-and-help: A challenge for social perception and human-ai collaboration. In *International Conference on Learning Representations*, 2020.

[111] Xavier Puig, Tianmin Shu, Joshua B Tenenbaum, and Antonio Torralba. Nopa: Neurally-guided online probabilistic assistance. 2022.

[112] Yuzhe Qin, Yueh-Hua Wu, Shaowei Liu, Hanwen Jiang, Ruihan Yang, Yang Fu, and Xiaolong Wang. Dexmv: Imitation learning for dexterous manipulation from human videos. *arXiv preprint arXiv:2108.05877*, 2021.

[113] Neil C Rabinowitz, Frank Perbet, H Francis Song, Chiyuan Zhang, SM Eslami, and Matthew Botvinick. Machine theory of mind. *arXiv preprint arXiv:1802.07740*, 2018.

[114] Miquel Ramırez and Hector Geffner. Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, pages 1778–1783, 2009.

[115] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):297–330, 2020.

[116] Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. *CoRR*, abs/1612.00563, 2016.

[117] John W Rowe. The us eldercare workforce is falling further behind. *Nature Aging*, 1(4):327–329, 2021.

[118] Leonel Rozo, Sylvain Calinon, Darwin G Caldwell, Pablo Jimenez, and Carme Torras. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 32(3):513–527, 2016.

[119] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and systems*, volume 2, pages 1–9. Ann Arbor, MI, USA, 2016.

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

[121] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9339–9347, 2019.

[122] Karl Schmeckpeper, Oleh Rybkin, Kostas Daniilidis, Sergey Levine, and Chelsea Finn. Reinforcement learning with videos: Combining offline observations with interaction. In *Conference on Robot Learning*, pages 339–354. PMLR, 2021.

[123] Ozan Sener, Amir Zamir, Silvio Savarese, and Ashutosh Saxena. Unsupervised semantic parsing of video collections. In *arXiv:1506.08438*, 2015.

[124] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-contrastive networks: Self-supervised learning from video. *Proceedings of International Conference in Robotics and Automation (ICRA)*.

[125] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[126] Ankit Shah, Shen Li, and Julie Shah. Planning with uncertain specifications (PUnS). *IEEE Robotics and Automation Letters*, 5(2):3414–3421, 2020.

[127] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[128] Tianmin Shu, Abhishek Bhandwaldar, Chuang Gan, Kevin Smith, Shari Liu, Dan Gutfreund, Elizabeth Spelke, Joshua Tenenbaum, and Tomer Ullman. AGENT: A benchmark for core psychological reasoning. In *International Conference on Machine Learning*, pages 9614–9625. PMLR, 2021.

[129] Tianmin Shu, Dan Xie, Brandon Rothrock, Sinisa Todorovic, and Song Chun Zhu. Joint inference of groups, events and human roles in aerial videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4576–4584, 2015.

[130] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017.

[131] Michael Shum, Max Kleiman-Weiner, Michael L Littman, and Joshua B Tenenbaum. Theory of minds: Understanding behavior in groups through inverse planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6163–6170, 2019.

[132] Gunnar A Sigurdsson, Abhinav Gupta, Cordelia Schmid, Ali Farhadi, and Karteek Alahari. Actor and observer: Joint modeling of first and third-person videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7396–7404, 2018.

[133] Gunnar A Sigurdsson, Abhinav Gupta, Cordelia Schmid, Ali Farhadi, and Karteek Alahari. Charades-ego: A large-scale dataset of paired third and first person videos. *arXiv preprint arXiv:1804.09626*, 2018.

[134] Gunnar A. Sigurdsson, Gul Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, 2016.

[135] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

[137] Shirin Sohrabi, Anton V Riabov, and Octavian Udrea. Plan recognition as planning revisited. In *IJCAI*, pages 3258–3264, 2016.

[138] Armando Solar-Lezama. *Program synthesis by sketching*. Citeseer, 2008.

[139] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Conference on Robot Learning*, pages 477–490. PMLR, 2022.

[140] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks, 2020.

[141] Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav S. Sukhatme. Embodied bert: A transformer model for embodied, language-guided visual task completion. In *EMNLP 2021 Workshop on Novel Ideas in Learning-to-Learn through Interaction*, 2021.

[142] Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. Neural program synthesis from diverse demonstration videos. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[143] Shao-Hua Sun, Te-Lin Wu, and Joseph J. Lim. Program guided agent. In *International Conference on Learning Representations*, 2020.

[144] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[145] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel X. Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. *CoRR*, abs/2106.14405, 2021.

[146] Ravi Tejwani, Yen-Ling Kuo, Tianmin Shu, Boris Katz, and Andrei Barbu. Social interactions as recursive mdps. In *Conference on Robot Learning*, pages 949–958. PMLR, 2022.

[147] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 1507–1514. AAAI Press, 2011.

[148] Tijmen Tieleman and Geoffrey Hinto. Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[149] Hung Tran, Vuong Le, and Truyen Tran. Goal-driven long-term trajectory prediction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 796–805, 2021.

[150] Joanne Truong, Sonia Chernova, and Dhruv Batra. Bi-directional domain adaptation for sim2real transfer of embodied navigation agents. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):2634–2641, 2021.

[151] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[152] Tomer Ullman, Chris Baker, Owen Macindoe, Owain Evans, Noah Goodman, and Joshua B Tenenbaum. Help or hinder: Bayesian models of social goal inference. In *Advances in neural information processing systems*, pages 1874–1882, 2009.

[153] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. In *CVPR*, 2017.

[154] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[155] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 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*, pages 6629–6638, 2019.

[156] Felix Warneken and Michael Tomasello. Altruistic helping in human infants and young chimpanzees. *Science*, 311(5765):1301–1303, 2006.

[157] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5922–5931, 2021.

[158] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992.

[159] Sarah A Wu, Rose E Wang, James A Evans, Joshua B Tenenbaum, David C Parkes, and Max Kleiman-Weiner. Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *Topics in Cognitive Science*, 13(2):414–432, 2021.

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

[161] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020.

[162] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5419, 2017.

[163] Kelvin Xu, Ellis Ratner, Anca Dragan, Sergey Levine, and Chelsea Finn. Learning a prior over intent via meta-inverse reinforcement learning. In *International Conference on Machine Learning*, pages 6952–6962. PMLR, 2019.

[164] Fujian Yan, Dali Wang, and Hongsheng He. Robotic understanding of spatial relationships using neural-logic learning. In *2020 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS), pages 8358–8365. IEEE, 2020.

[165] Yezhou Yang, Yi Li, Cornelia Fermuller, and Yiannis Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the world wide web. In *AAAI*, 2015.

[166] Yu Yao, Ella Atkins, Matthew Johnson-Roberson, Ram Vasudevan, and Xiaoxiao Du. BITRAP: Bi-directional pedestrian trajectory prediction with multi-modal goal estimation. *IEEE Robotics and Automation Letters*, 6(2):1463–1470, 2021.

[167] Sarah Young, Dhiraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. Visual imitation made easy. In *Conference on Robot Learning (CoRL)*, 2020.

[168] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *arXiv:1511.07122*, 2015.

[169] Hejia Zhang and Stefanos Nikolaidis. Robot learning and execution of collaborative manipulation plans from youtube cooking videos. *arXiv preprint arXiv:1911.10686*, 2019.

[170] Xiaohan Zhang, Bharat Lal Bhatnagar, Sebastian Starke, Vladimir Guzov, and Gerard Pons-Moll. Couch: Towards controllable human-chair interactions. In *European Conference on Computer Vision (ECCV)*. Springer, October 2022.

[171] Hang Zhao, Jiyang Gao, Tian Lan, Chen Sun, Benjamin Sapp, Balakrishnan Varadarajan, Yue Shen, Yi Shen, Yuning Chai, Cordelia Schmid, et al. TNT: Target-driven trajectory prediction. *arXiv preprint arXiv:2008.08294*, 2020.

[172] Tan Zhi-Xuan, Jordyn Mann, Tom Silver, Josh Tenenbaum, and Vikash Mansinghka. Online bayesian goal inference for boundedly rational planning agents. *Advances in Neural Information Processing Systems*, 33:19238–19250, 2020.

[173] Bolei Zhou, Alex Andonian, and Antonio Torralba. Temporal relational reasoning in videos. *CoRR*, abs/1711.08496, 2017.

[174] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6541–6548. IEEE, 2021.

[175] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 483–492, 2017.

[176] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. In *ICCV*, 2015.

Appendix A

Environment-Aware Agents

A.1 Implementation details

To train ResActGraph, we use the ground truth graphs extracted from Unity to represent the environments. We set the propagation time $K = 2$ except for the ablation studies. To train the model that ingests demonstrations to generate sketches, we use a finetuned Resnet-18 [54] to extract image features and a 3-layer CNN to extract features from semantic segmentation map. All the GRUs are with 256 hidden units except for those in GGNN are with 32.

The following is the details of the model ResActGraph, Desc2sketch, and Demo2sketch:

RNN-ResActGraph. The hidden units of GRU is 256 and the hidden units of GGNN is 32. We use Adam [71] with $3 * 10^{-4}$ as initial learning rate and mini-batch size 32. Schedule sampling is applied to mitigate the exposure bias. We use linear schedule with max probability 1.0 and min probability 0.3 and decrease in 20000 steps.

Desc2sketch. We use seq2seq model with GRU. The hidden units of GRU is 256. We use Adam with $3 * 10^{-4}$ as initial learning rate and mini-batch size 64. Schedule sampling is applied to mitigate the exposure bias. We use linear schedule with max probability 1. and min probability 0. and decrease in 20000 steps.

Demo2sketch. We finetune Resnet-18 to extract image features and build another CNN to ingest the ground truth semantic segmentation maps. The CNN struc-

	Conv_1+BN	Conv_2+BN	Conv_3+BN	FC
Input (size: 240, 320, 3)	Kernel: 5*5 Stride: 1 Padding: 2 Channel: 16	Kernel: 5*5 Stride: 1 Padding: 2 Channel: 32	Kernel: 5*5 Stride: 3 Padding: 2 Channel: 32	256

Table A.1: The model architecture of the CNN that ingests the semantic segmentation map.

ture is shown in Table A.1. On the other hand, we use GRU to decode the sketches. The hidden units of GRU is 256. We use Adam with $3 * 10^{-4}$ as initial learning rate and mini-batch size 16. Schedule sampling is applied to mitigate the exposure bias. We use linear schedule with max probability 1. and min probability 0. and decrease in 10000 steps.

A.2 Method for dataset augmentation

We formally describe how we augment the dataset of programs using exceptions. Let p be the original program and e' the modified environment. Let Γ_ζ be the function that for some exception θ, takes a program p and environment e' corrects the program into p'. Finally, let Ω be the function that takes p, e as the input and outputs its post-conditions[1]. We formally define the program extension as:

$$\begin{aligned} p' &= \Gamma_\zeta(p, e'), \\ e_{final} &= \Psi(e', p') \\ &s.t.\ \forall j \in \Omega(p), e_{final} \text{ satisfies } j \end{aligned} \tag{A.1}$$

The algorithm is shown in 4:

[1]A post-condition is a condition or predicate that must always be true just after the execution of some section of code.

A.3 Details of the Evaluation Metrics

We compute the $\mathbb{F}_1(\hat{G}, G)$ as follows:

$$\mathbb{P}(\hat{G}, G) = \frac{\|\mathbb{T}(\hat{G})\| \otimes \|\mathbb{T}(G)\|}{\|\mathbb{T}(\hat{G})\|}$$
$$\mathbb{R}(\hat{G}, G) = \frac{\|\mathbb{T}(\hat{G})\| \otimes \|\mathbb{T}(G)\|}{\|\mathbb{T}(G)\|}$$
$$\mathbb{F}_1(\hat{G}, G) = \frac{2 \cdot \mathbb{P}(\hat{G}, G) \cdot \mathbb{R}(\hat{G}, G)}{\mathbb{P}(\hat{G}, G) + \mathbb{R}(\hat{G}, G)}$$

where $\otimes$, $\mathbb{P}$, and $\mathbb{R}$ are the binary matching function, precision and recall respectively. $\mathbb{T}$ is a function that converts graphs to tuples.

Parsability refers to the syntactic correctness of the programs, evaluating whether

Algorithm 4 Pseudo-code of the method described in Section 5.5.4. Given a program p with preconditions $\Phi(p)$, the procedure produces at most n new programs.

```
procedure AUGMENT(p, Φ(p), n)
    perturbed-set ← set()
    while len(perturbed-set) < n do                    ▷ Perturb preconditions
        z ← rand(0, 1)
        Φ(p)′ ← perturb(Φ(p), z)
        e′ ∼ E_Φ(p)′                                   ▷ Sample with eq. 5.10
        perturbed-set.add(e′)
    end while
    final-progs ← list()
    p′ ← p
    foreach e′ ← perturbed-set do
        iter ← 1
        valid ← False
        while not valid and iter < maxiter do
            try
                Ψ(p′, e′)                              ▷ Execute in simulator
                final-progs.add(p′)
                valid ← True
            catch ExceptionType
                p′ ← Γ_ExceptionType(p′, e′)           ▷ Solve with eq. A.1
                iter ← iter + 1
            end try
        end while
    end foreach
    return final-progs
end procedure
```

the number of object arguments given the action is valid. `Sleep TV` is invalid, since `Sleep` does not expect any object. Executability refers to the semantic correctness, meaning that 1) the combination of objects and actions is valid (e.g., `Grab Bedroom` is invalid) and 2) the instruction can be performed under a given environment state (e.g., `Grab Mug` is only valid if the mug is close).

A.4 Further analysis of the experiments

The performance of the sketch prediction from descriptions is far better than from demonstrations (shown in Section 5.6.3). We find that only 7% of the testing vase are more easily predicted from demonstrations than descriptions, especially for the activities 'pet cat' and 'pick up phone'. One potential reason is that the average length of the descriptions of these activities (15.3 words) is longer than those of other activities (13.4).

A.5 Knowledge base

A.5.1 RealEnv

We collected two knowledge bases that aim to set up our environments. The first one, *ObjectStates KB*, contains information about which states correspond to which objects. We collected 29 different states and 9 of them are considered by the simulator. Table A.3 shows the collected states with an indication of whether they are included in the simulator.

The second knowledge base, *ObjectRelations KB*, contains typical spatial relations between objects that aims at obtaining realistic environment layouts to set up the scene for each activity. We show a few examples of spatial relations in Table A.2. In total, it contains 505 types of spatial relations included.

States	Examples	In Sim.
On/Off	Television, Computer, Microwave, Oven, Stereo	✓
Plugged/Unplugged	Television, Computer, Microwave, Oven, Stereo	✓
Open/Closed	Fridge, Cabinet, Folder, Bag, Purse, Blender	✓
Free/Occupied	Sofa, Table, Sink, Toilet, Piano, Shoe Rack, Bed	✓
Dirty/Clean	Apple, Floor, Bowl, Socks, Plate, Cupboard	
Full/Empty	Washing machine, Dishwasher, Measuring Cup	
Grabbed	Apple, Towel, Book, Glass, Plate, Toy, Cat, Pot	✓
Flushed	Toilet	
Sharpened	Knife	
Wet/Dry	Towel, Plate, Glass, Napkin, Shirt, Toothbrush	
Peeled/Sliced/Squeezed	Banana, Vegetable, Fish, Fruit, Cake	
Burning	Plate, Pot, Pan, Fish, Water, Candle, Chicken	
Frozen	Water, Vegetable, Ice cream, Wine, Hamburger	
Folded/Unfolded	Sheets, Shirt, Jacket, Ironing Board, Placemat	
Hot/Cold	Plate, Bowl, Water, Hamburger, Oven, Microwave	
Cooked/Raw/Rotten	Egg, Fish, Vegetable, Meat, Fruit, Pasta, Pizza	

Table A.2: ObjectStates KB

Object	In	On	Nearby
Wine Glass	Cupboard, Cabinet, Dishwasher	Coffee table, Dishrack, Kitchen Counter	
Milk	Fridge, Microwave, Trashcan, Sauce pan	Kitchen counter, Table, Pantry	
Remote Control	Dresser	Sofa, Love Seat, Bed, Table, Mousepad	Dvd player
Pot	Microwave, Cupboard, Cabinet	Dishrack, Kitchen Counter, Stove	
Razor	Cabinet, Trashcan, Bag, Box	Table, Bathroom counter	
Cd Player	Closet, Cabinet, Bag	Table, Couch	Stereo
Chicken	Fridge, Trashcan, Freezer, Sauce Pan, Oven	Kitchen Counter, Cutting Board, Plate	
Oven mitts	Oven, Cabinet, Basket for clothes	Table, Kitchen counter	
Document	Folderm Garbage Can, Cabinet, Filing Cabinet	Desk, Sofa, Bookshelf, Music Stand	
Sheets	Basket for clothes, Dresser, Washing Machine, Cabinet	Couch, Love Seat, Bed	

Table A.3: ObjectRelations KB

A.5.2 ExceptionHandler

We show the exceptions and subroutines used to augment the dataset in Table A.4. Each row shows the name of the exception, the case when the exception fires and a description of the subroutine that is called to correct the given exception.

A.6 Additional Qualitative Results

We now show additional qualitative results of our *RNN-ResActGraph*. In Figure A-1 and Figure A-2, we use the ground truth sketches as the input to the model. Although the generated programs are not exactly the same as the ground truth programs, the generated program does achieve similar environment state as the environment state of ground truth.

In Figure A-3, we show how the generated program changes when using the same sketches but different graphs as input. We highlight the difference between two environments with orange color. From both cases, we observe that the generated programs achieve almost the same environment state even thought they start from different environment states.

In Figure A-4, we show the generated programs from the sketches predicted from descriptions. It shows that sometimes our model cannot generate the exact same program as ground truth, but the generated program does achieve similar environment state as the environment state of the ground truth. This is also shown in the main paper where the $\mathbb{F}_1$ scores of the proposed model are higher than others.

Lastly, we show one example of the of our simulators in Figure A-5

Sketch	GT Program	Generated Program
[Grab] <tooth paste> [Put] <tooth paste> <toothbrush holder> [Switchon] <faucet> [Rinse] <toothbrush> **Environment** Faucet (70) near Sink (9) Faucet (70) is on Faucet (70) in Bathroom (32) Tooth paste (15) near Sink (9) Toothbrush holder (63) in Sink (9) Toothbrush (2) in Bathroom (32)	[Walk] <bathroom> (32) [Walk] <sink> (9) [Find] <toothbrush holder> (63) [Turnto] <toothbrush holder> (63) [Lookat] <toothbrush holder> (63) [Find] <tooth paste> (15) [Grab] <tooth paste> (15) [Find] <toothbrush> (2) [Grab] <toothbrush> (2) [Walk] <toothbrush holder> (63) [Put] <tooth paste> (15) <toothbrush holder> (63) [Find] <faucet> (70) [Rinse] <toothbrush> (2) [Walk] <toothbrush holder> (63) [Put] <toothbrush> (2) <toothbrush holder> (63)	[Walk] <bathroom> (32) [Walk] <sink> (9) [Find] <toothbrush holder> (63) [Turnto] <toothbrush holder> (63) [Lookat] <toothbrush holder> (63) [Find] <tooth paste> (15) [Grab] <tooth paste> (15) [Find] <toothbrush> (53) [Grab] <toothbrush> (53) [Walk] <toothbrush holder> (63) [Put] <tooth paste> (15) <toothbrush holder> (63) [Rinse] <toothbrush> (2) [Walk] <faucet> (70) [Switchoff] <faucet> (70)

Sketch	GT Program	Generated Program
[Wash] <hands> [Grab] <soap> [Rinse] <hands> [Wipe] <hands> **Environment** Faucet (10) is off Faucet (10) near Sink (93) Faucet (10) in Bathroom (41) Soap (77) in Bathroom (41) Towel (8) in Bathroom (41) Hands (92) near Sink (93)	[Walk] <bathroom> (41) [Walk] <sink> (93) [Find] <sink> (93) [Turnto] <sink> (93) [Find] <faucet> (10) [Switchon] <faucet> (10) [Find] <hands> (92) [Wash] <hands> (92) [Find] <soap> (77) [Grab] <soap> (77) [Scrub] <hands> (92) [Putobjback] <soap> (77) [Rinse] <hands> (92) [Switchoff] <faucet> (10) [Find] <towel> (8) [Grab] <towel> (8) [Wipe] <hands> (92) [Putobjback] <towel> (8)	[Walk] <bathroom> (41) [Walk] <sink> (93) [Find] <sink> (93) [Turnto] <sink> (93) [Find] <faucet> (10) [Switchon] <faucet> (10) [Find] <hands> (92) [Rinse] <hands> (92) [Find] <soap> (77) [Grab] <soap> (77) [Squeeze] <soap> (77) [Wash] <hands> (92) [Rinse] <hands> (92) [Switchoff] <faucet> (10) [Find] <towel> (8) [Grab] <towel> (8) [Wipe] <hands> (92) [Putobjback] <towel> (8)

Sketch	GT Program	Generated Program
[Grab] <paper towel> [Wipe] <hands> **Environment** Hands (4) near Paper Towel (54) Paper Towel (54) in Bathroom (81)	[Walk] <bathroom> (81) [Walk] <paper towel> (54) [Find] <paper towel> (54) [Turnto] <paper towel> (54) [Lookat] <paper towel> (54) [Grab] <paper towel> (54) [Find] <hands> (4) [Wipe] <hands> (4)	[Walk] <bathroom> (81) [Walk] <paper towel> (54) [Find] <paper towel> (54) [Grab] <paper towel> (54) [Find] <hands> (4) [Wipe] <hands> (4) [Putobjback] <paper towel> (54)

Figure A-1: Predictions from the RNN-ResActGraph from the GT sketch and the environment, along with the GT programs. We color the longest common subsequence in light green.

Sketch	GT Program	Generated Program
[Putoff] <clothes jacket> [Put] <clothes jacket> <hanger> **Environment** Hanger (18) near Clothes Jacket (19) Hanger (18) in Bedroom (28) Bed (4) in Bedroom (28) Clothes Jacket (19) in Character	[Walk] <bedroom> (28) [Walk] <clothes jacket> (19) [Find] <clothes jacket> (19) [Find] <clothes jacket> (19) [Putoff] <clothes jacket> (19) [Grab] <clothes jacket> (19) [Find] <hanger> (18) [Put] <clothes jacket> (19) <hanger> (18)	[Walk] <bedroom> (28) [Walk] <clothes jacket> (19) [Find] <clothes jacket> (19) [Find] <clothes jacket> (19) [Putoff] <clothes jacket> (16) [Grab] <clothes jacket> (16) [Find] <bed> (4) [Put] <clothes jacket> (19) <bed> (4)

Sketch	GT Program	Generated Program
[Lie] <bed> [Sleep] **Environment** Lamp (72) is on Lamp (72) near Bed (79) Bed (79) in Bedroom (6) Bed (79) is free	[Walk] <bedroom> (6) [Walk] <lamp> (72) [Switchoff] <lamp> (72) [Walk] <bed> (79) [Lie] <bed> (79) [Sleep]	[Walk] <bedroom> (6) [Walk] <bed> (79) [Find] <lamp> (72) [Lie] <bed> (79) [Sleep]

Sketch	GT Program	Generated Program
[Greet] <child> [Greet] <man> [Greet] <woman> **Environment** Man (48) near Child (67) Woman (59) near Child (67) Man (48) in Living Room (12)	[Walk] <living room> (12) [Walk] <child> (67) [Find] <child> (67) [Greet] <child> (67) [Find] <man> (48) [Greet] <man> (48) [Find] <woman> (59) [Greet] <woman> (59)	[Walk] <living room> (12) [Walk] <child> (67) [Find] <child> (67) [Greet] <child> (67) [Find] <woman> (59) [Greet] <woman> (59) [Find] <man> (48) [Greet] <man> (48)

Sketch	GT Program	Generated Program
[Switchon] <faucet> [Put] <glass> <sink> [Drink] <glass> **Environment** Faucet (13) near Sink (68) Faucet (13) in Dining Room (72) Water (8) near Sink (68) Glass (99) in Sink (68)	[Walk] <dining room> (72) [Walk] <sink> (22) [Find] <glass> (99) [Grab] <glass> (99) [Find] <faucet> (15) [Switchon] <faucet> (15) [Find] <water> (5) [Put] <glass> (99) <sink> (22) [Grab] <glass> (99) [Switchoff] <faucet> (15) [Drink] <glass> (99)	[Walk] <dining room> (72) [Walk] <glass> (99) [Find] <glass> (99) [Grab] <glass> (99) [Find] <faucet> (15) [Switchon] <faucet> (15) [Find] <water> (5) [Find] <sink> (22) [Put] <glass> (99) <sink> (22) [Grab] <glass> (99) [Switchoff] <faucet> (15) [Drink] <glass> (99)

Figure A-2: Predictions from the RNN-ResActGraph from the GT sketch and the environment, along with the GT programs. We color the longest common subsequence in light green.

Sketch		[open] <washing_machine> [putback] <basket> <washing_machine> [putback] <soap> <washing_machine> [switchon] <washing_machine>	
Environment 1	Generated Program 1	Environment 2	Generated Program 2
Soap (1002) near washing machine (1001) Soap (1002) inside washing machine (1001) Washing machine (1001) is opened Washing machine (1001) is on Washing machine (1001) is plugged	[Walk] <bedroom> (273) [Walk] <basket> (1000) [Find] <basket> (1000) [Grab] <basket> (1000) [Find] <washing_machine> (1001) [SwitchOff] <washing_machine> (1001) [Open] <washing_machine> (1001) [Put] <basket> (1000) <washing_machine> (1001) [Find] <soap> (1002) [Grab] <soap> (1002) [Put] <soap> (1002) <washing_machine> (1001) [Close] <washing_machine> (1001) [SwitchOn] <washing_machine> (1001)	Soap (1002) near washing machine (1001) Washing machine (1001) is closed Washing machine (1001) is off Washing machine (1001) is unplugged	[Walk] <bedroom> (273) [Walk] <basket> (1000) [Find] <basket> (1000) [Grab] <basket> (1000) [Find] <washing_machine> (1001) [Open] <washing_machine> (1001) [Put] <basket> (1000) <washing_machine> (1001) [Find] <soap> (1002) [Grab] <soap> (1002) [Put] <soap> (1002) <washing_machine> (1001) [Close] <washing_machine> (1001) [Plugin] <washing_machine> (1001) [SwitchOn] <washing_machine> (1001)

Sketch		[putback] <groceries> <fridge>	
Environment 1	Generated Program 1	Environment 2	Generated Program 2
Grocery (1001) near fridge (1000) Fridge (1000) in dining room (47) Grocery (1001) in dining room (47) Fridge (1000) is closed	[Walk] <dining_room> (47) [Walk] <fridge> (1000) [Find] <fridge> (1000) [Open] <fridge> (1000) [Find] <groceries> (1001) [Grab] <groceries> (1001) [Put] <groceries> (1001) <fridge> (1000) [Close] <fridge> (1000)	Grocery (1001) near fridge (1000) Fridge (1000) in dining room (47) Grocery (1001) in dining room (47) Fridge (1000) is opened	[Walk] <dining_room> (47) [Walk] <fridge> (1000) [Find] <fridge> (1000) [Find] <groceries> (1001) [Grab] <groceries> (1001) [Put] <groceries> (1001) <fridge> (1000) [Close] <fridge> (1000)

Figure A-3: Predictions from the RNN-ResActGraph model from the same sketches but different environments. We color the difference between environments in orange and the longest common subsequence in light green.

Description	pick up book. sit on sofa. open book. read	Description	turn on reading light. bring book to couch. sit on couch. <unk> pillow. put pillow <unk> my back. read my book.
Sketch	[sit] <chair>, [read] <book>	Sketch	[sit] <sofa>, [read] <book>
GT Program	[walk] <living_room> (21) [walk] <book> (1001) [find] <book> (1001) [grab] <book> (1001) [find] <chair> (1002) [sit] <chair> (1002) [read] <book> (1001)	GT Program	[walk] <bedroom> (276) [walk] <book> (1001) [find] <book> (1001) [grab] <book> (1001) [find] <sofa> (201) [sit] <sofa> (201) [standup] [walk] <bookmark> (1004) [find] <bookmark> (1004) [read] <book> (1001)
Generated Program	[walk] <living_room> (21) [walk] <book> (1001) [find] <book> (1001) [grab] <book> (1001) [find] <chair> (1002) [sit] <chair> (1002) [read] <book> (1001)	Generated Program	[walk] <bedroom> (276) [walk] <book> (1001) [find] <book> (1001) [grab] <book> (1001) [find] <sofa> (201) [sit] <sofa> (201) [read] <book> (1001)

Figure A-4: Predictions from the RNN-ResActGraph from the predicted sketch (from descriptions) and the environment, along with the GT programs. We color the longest common subsequence in light green.

Figure A-5: An example of snapshots generated in VirtualHome for a given program.

Exception name	When is raised	Subroutine
IS_OPEN	Character aims to open something that is already open	Remove the *Open* instruction
IS_CLOSED	Character aims to close something that is closed	Remove the *Close* instruction
STANDING	1) Character aims to stand up but is not sitting, 2) Character aims to sleep but is not lying nor sit ting	1) Remove *Stand Up*, 2) Add instruction *Sit* or *Lie*
SITTING	1) Character aims to *Sit* but is sitting, 2) Character aims to walk but is sitting	1) Remove instruction *Sit*, 2) Add instruction *Stand up*
NOT_CLOSE	Character wants to interact with an object but it is not close enough	Add instruction *Walk* towards the object
NOT_FACING	Character wants to Watch and object (e.g. TV) but is not facing it.	Add instruction textitTurnTo the object
IS_ON	1) Character switches on an object that is already on 2) Character aims to open a utility (e.g. washing machine) before turning it off	1) Remove instruction *Turn On* 2) Insert an instruction *Switch Off*.
IS_OFF	1) Character switches off an object that is off, 2) character watches TV but it is off	1) Remove instruction *Switch Off*, 2) Insert *Switch On*
PLUGGED	1) Character plugs in an object that is already plugged	Remove instruction *PlugIn*
UNPLUGGED	1) Character plugs out an object that is plugged out 2) Character switches on an object that is plugged out	1) Remove instruction *PlugOut* 2) Insert instruction *PlugIn*
OCCUPIED	There are objects in a surface where the character aims to sit or lie	Obtain all the objects in the occupied surface and insert *Grab* and *Release* instructions to remove them
INSIDE_CLOSED	The character aims to interact an object that is inside some closed one (e.g. a fridge)	Obtain the object containing the target object, and open it
FREE_HAND	The Character wants to interact with an object but both of their hands are occupied	Leave one object from the hands, Interact with object and grab object again

Table A.4: Exceptions handled to augment the programs dataset, based on perturbation of the environment

Appendix B

Watch-And-Help

B.1 Environment Details

To ensure that we can represent agents doing complex activities in household environments, we study the *Watch-And-Help* challenge in VirtualHome. In this section we describe how we set up the environment to study this challenge.

Execution

In Section 2.4, we described two modes of execution in VirtualHome, a *video mode*, to generate videos of agents performing activities, and an *interactive mode*, allowing to take actions one step at a time and observe the environment resulting from those actions. We use the *execution mode* for both the *Watch* and *Help* phase in this challenge.

Observation

The environment supports symbolic and visual observations (Figure B-1a), allowing agents to learn helping behaviors under different conditions. Here, we focus on symbolic observations, which we represent as a state graph with each node representing the class label and physical state of an object, and each edge representing the spatial relation of two objects. We study agents with full observability of the environment,

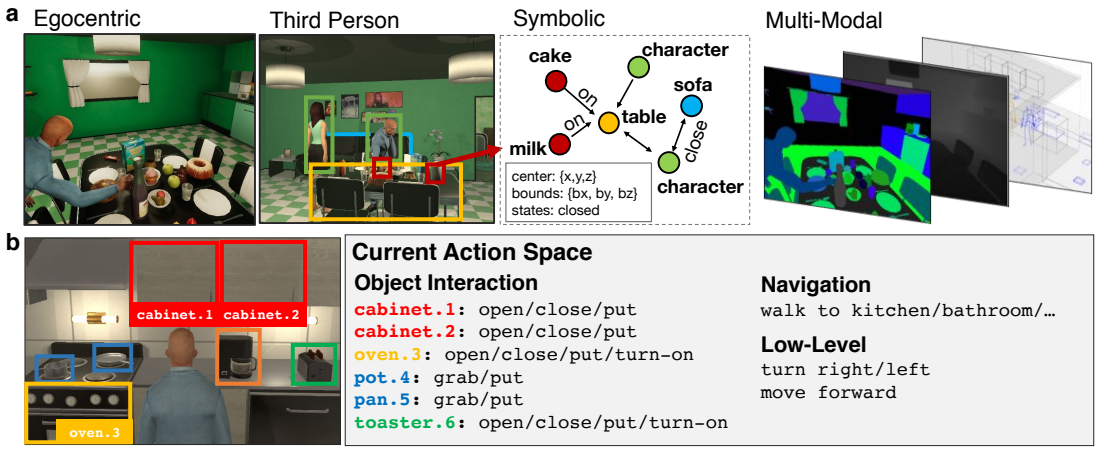


Figure B-1: a) VirtualHome provides egocentric views, third-person views and scene graphs with symbolic state representations of objects and agents. It also offers multi-modal inputs (RGB, segmentation, depth, 3D boxes and skeletons). b) Illustration of the action space at one step.

meaning that they observe at every time a full graph of the environment, and partial observability, in which they observe a subgraph with nodes corresponding to objects in the same room as the agent and not in a closed container and edges corresponding to spatial relationships between those nodes.

Action space

As shown in Figure B-1b, agents in the challenge can perform both high-level actions, such as navigating towards a known location, or interacting with an observed object, and low-level actions, such as turning or moving forward for a small step. For actions involving interactions with entities (objects or other agents), an agent needs to specify the indices of the intended entities (e.g., "grab $\langle 3 \rangle$" stands for grabbing the object with id 3). An agent can only interact with objects that are within its field of sight, and therefore its action space changes at every step. When executing navigation actions, an agent can only move 1 meter towards the target location within one step. On average, an agent's action space includes 167 different actions per step.

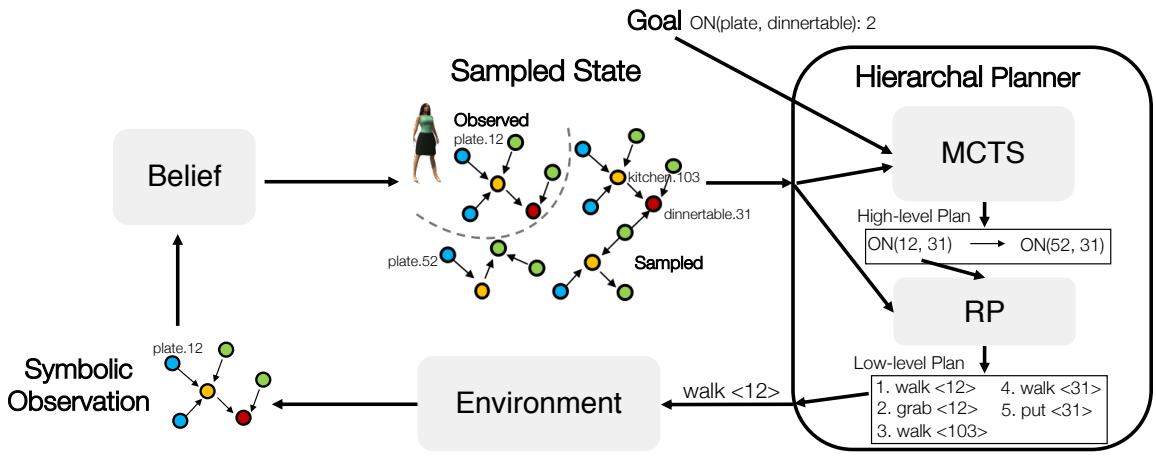


Figure B-2: Schematic of the human-like agent. Based on the state graph sampled from the belief, the hierarchical planner searches for a high-level plan over subgoals using MCTS; then RP searches for a low-level plan over actions for each subgoal. The first action of each plan is sent back to the environment for execution.

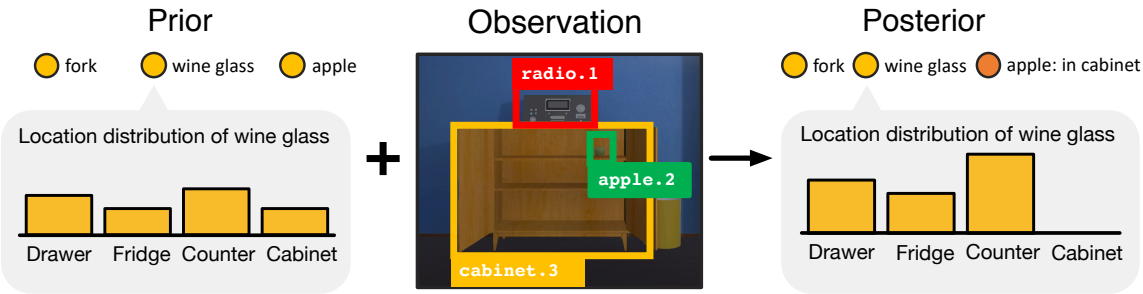


Figure B-3: The agent's belief is represented as the location distribution of objects, and is updated at each step based on the previous belief and the latest observation. In the example, the open cabinet reveals that the wine glass can not be in there, and that there is an apple inside, updating the belief accordingly.

B.1.1 Human-like Agent

We discuss how the human-like agent works in more details here. The agent pipeline can be seen in Figure B-2. The agent has access to a partial observation of the environment, limited to the objects that are in the same room and not in some closed container. The agent is equipped with a belief module (Figure B-3), that gives information about the unseen objects, under the assumption that the existence of objects in the environment is known, but not their location. For each object in the environment, the belief contains a distribution of the possible locations where it

could be. We adopt uniform distributions as the initial belief when the agent has not observed anything.

At each time, the agent obtains a partial observation, and updates its belief distribution accordingly. Then, the belief module samples a possible world state from the current distribution. To ensure that the belief state is consistent between steps, we only resample object locations that violate the current belief (e.g. an object was believed to be in the fridge but the agent sees that the fridge is in fact empty).

Based on the sampled state, a hierarchical planner will search for the optimal plan for reaching the goal, based on the goal definition. Specifically, we use MCTS to search for a sequence of subgoals (i.e., predicates), and then each subgoal is fed to a regression planner (RP) that will search for an action sequence to achieve the subgoal. For the high-level planner, the subgoal space is obtained by the intersection between what predicates remained to be achieved and what predicates could be achieved based on the sampled state. Note here each subgoal would specify an object instance instead of only the object class defined in the goal so that the low-level planner will be informed which object instances it needs to interact with. For instance, in the example illustrated in Figure B-2, there are two plates (whose indices are 12, 52) and the dinner table's index is 31 according to the sampled state. There are two unsatisfied goal predicates (i.e., two `ON(plate, dinnertable)`), then a possible subgoal space for the high-level planner would be {`ON(12, 31)`, `ON(52, 31)`}. For RP, it starts from the state defined by the subgoal and searches for the low-level plan backward until it finds an action that is part of the current action space of the agent.

To mimic human behaviors in a home setting, we also expect the human-like agent to close containers unless it needs to look inside or put objects into them. For that, we augment the MCTS-based high-level planner with heuristics for the closing behavior – the agent will close an container when it finds no relevant goal objects inside or has already grabbed/put in the all target objects out of that container. We find that this augmentation makes the overall agent behaviors closer to what a real human would do in a household environment.

Thanks to the hierarchical design, the planner for the human-like agent can be

run in real-time (on average, replanning at each step only takes 0.05 second). This also gives the agent a bounded rationality, in that the plan is not the most optimal but is reasonably efficient. The optimality of the planner can be further tuned by the hyper-parameters of MCTS, such as the number of simulation, the maximum number steps in the rollouts, and the exploration coefficients.

B.1.2 Specifications

The environment can be run in a single or multiple processes. A single process runs at 10 actions per second. We train our models using 10 processes in parallel.

B.2 More Details on the Challenge Setup

B.2.1 Predicate Sets for Goal Definitions

Table B.1: Predicate sets used for defining the goal of Alice in five types of activities.

Set up a dinner table	`ON(plate,dinnertable)`, `ON(fork,dinnertable)`, `ON(waterglass,dinnertable)`, `ON(wineglass,dinnertable)`
Put groceries	`IN(cupcake,fridge)`, `IN(pancake,fridge)`, `IN(poundcake,fridge)`, `IN(pudding,fridge)`, `IN(apple,fridge)`, `IN(juice,fridge)`, `IN(wine,fridge)`
Prepare a meal	`ON(coffeepot,dinnertable)`, `ON(cupcake,dinnertable)`, `ON(pancake,dinnertable)`, `ON(poundcake,dinnertable)`, `ON(pudding,dinnertable)`, `ON(apple,dinnertable)`, `ON(juice,dinnertable)`, `ON(wine,dinnertable)`
Wash dishes	`IN(plate,dishwasher)`, `IN(fork,dishwasher)`, `IN(waterglass,dishwasher)`, `IN(wineglass,dishwasher)`
Read a book	`HOLD(Alice,book)`, `SIT(Alice,sofa)`, `ON(cupcake,coffeetable)`, `ON(pudding,coffeetable)`, `ON(apple,coffeetable)`, `ON(juice,coffeetable)`, `ON(wine,coffeetable)`

Table B.1 summarizes the five predicate sets used for defining goals. Note that we can support more predicates for potential future extensions on the goal definitions.

B.2.2 Training and Testing Setup

During training, we randomly sample one of the 1011 training tasks for setting up a training episode. For evaluating an AI agent on the testing set, we run each testing

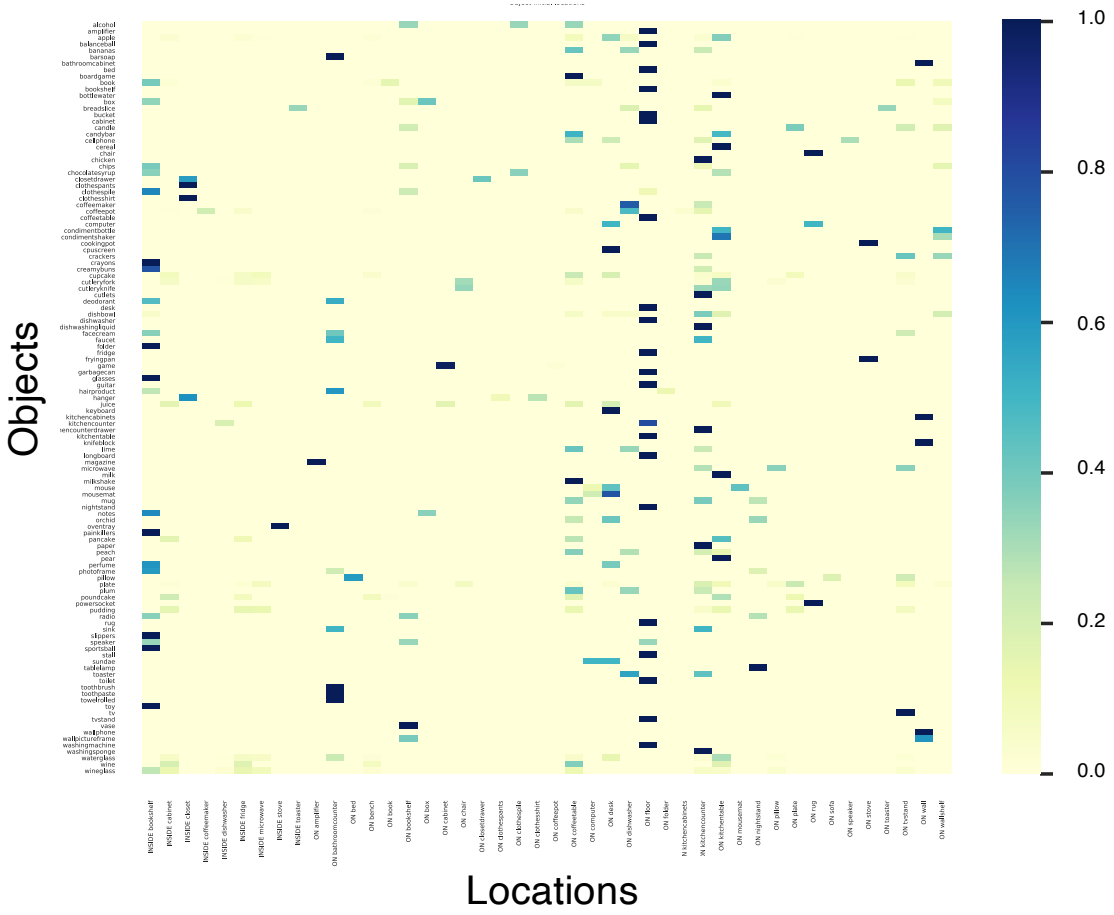


Figure B-4: Initial location distributions of all objects in the environment. Rows are objects and columns are locations. The color indicates the frequency.

task for five times using different random seeds and report the average performance.

For training goal inference, we also provide an additional training set of 5303 demonstrations (without pairing helping environments) synthesized in the 5 training apartments. Note that these demonstrations are exclusively used for training goal inference models and would not be used for helping tasks.

B.2.3 Distribution of Initial Object Locations

Figure B-4 shows the initial location distribution of all objects in the helping environments sampled for the challenge, and Figure B-5 shows the initial location distributions for only the objects involved in the goal predicates.

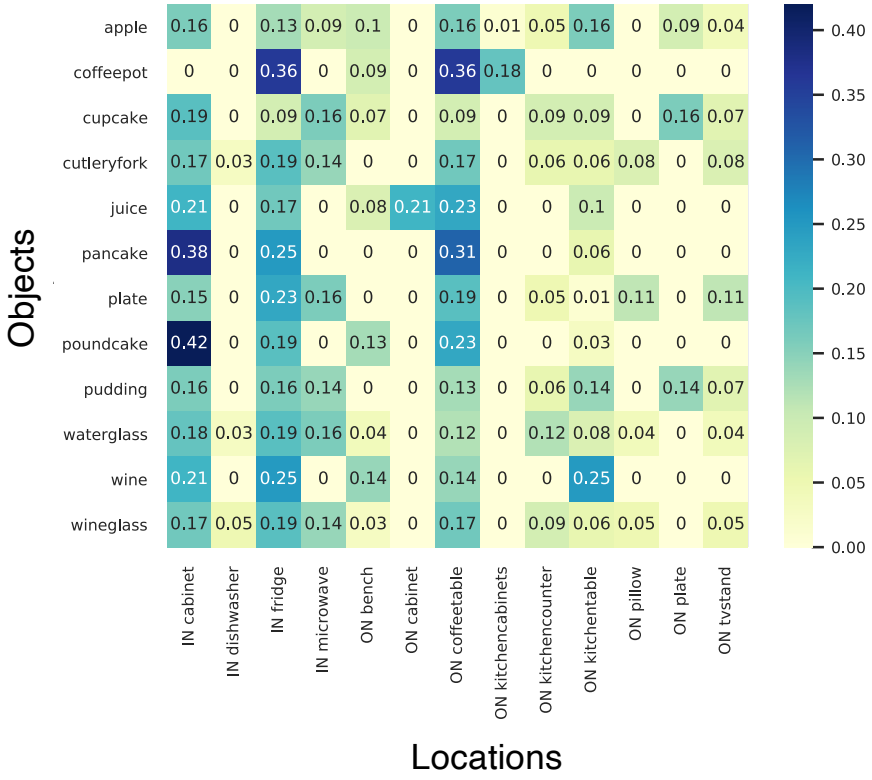


Figure B-5: Initial location distributions of the goal objects. Rows are objects and columns are locations. The color indicates the frequency.

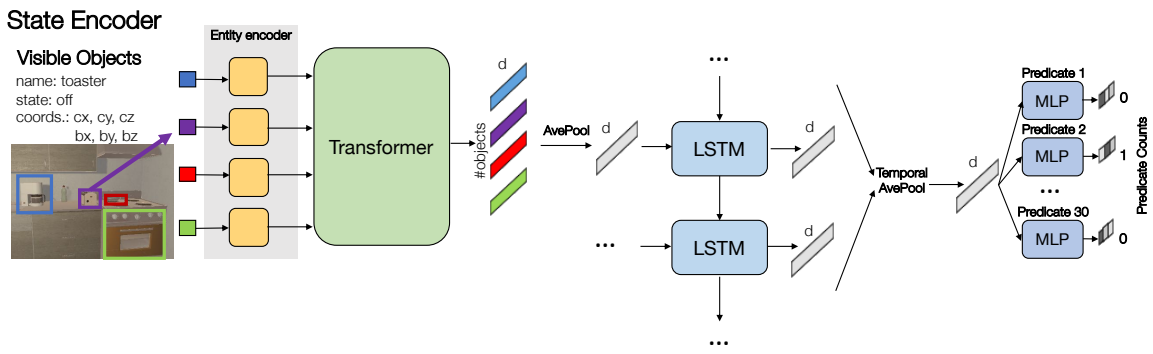


Figure B-6: Network architecture of the goal inference model, which encodes the symbolic state sequence in demonstrations and infers the count for each predicate.

B.3 Implementation Details of Baselines

B.3.1 Goal Inference Module

Figure B-6 shows the architecture of the goal inference model described in the paper, where $d = 128$ indicates the dimension of vectors. In this network, the LSTM has 128 hidden units and the MLP units are comprised of two 128-dim fully connected layers. For both node embeddings and the latent states from the LSTM, we use average pooling.

B.3.2 Hierarchical Planner

The hierarchical planner (**HP**) baseline is similar to the planner designed for the human-like agent (Section B.1.1) but has its own observation and belief. When given the ground-truth goal of Alice, the MCTS-based high-level planner will remove the subgoal that Alice is going to pursue from its own subgoal space.

B.3.3 General Training Procedure for RL-based Approaches

We train the high-level RL policy by giving ground-truth goals and by using RP as the low-level planner to reach the subgoals sampled from the high-level policy. Whenever a goal predicate is satisfied (either by Alice or by Bob), Bob will get a reward of +2; it will also get a -0.1 penalty after each time step. We adopt the multi-task RL

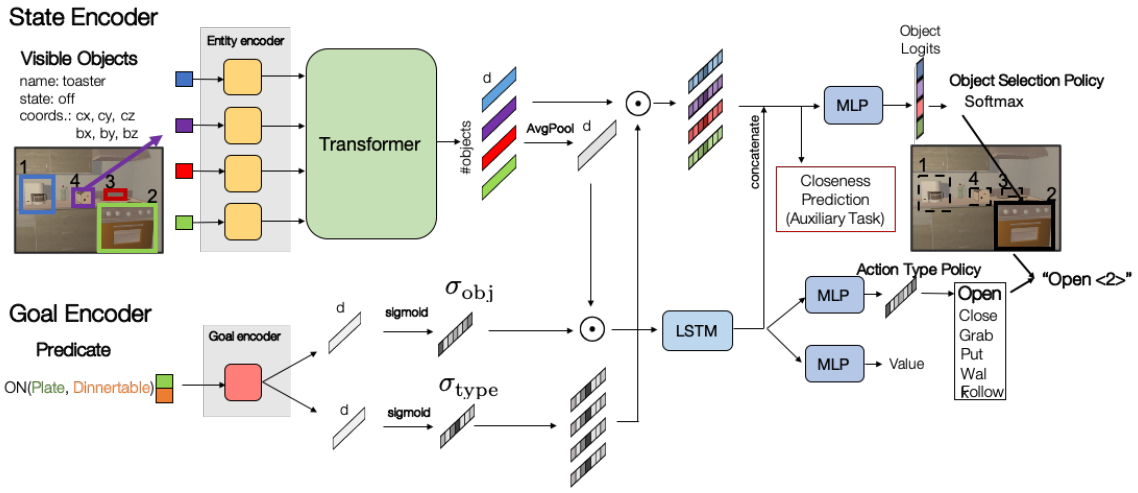


Figure B-7: Network architecture of the low-level policy in the **HRL** baseline. Note that the object selection policy also considers "Null" as a dummy object node for actions that do not involve an object, which is not visualized here.

approach introduced in [130] to train the low-level policy in a single-agent setting, where we randomly sample one of the predicates in the goal in each training episode and set it to be the objective for Bob. This is to ensure that Bob can learn to achieve subgoals through the low-level policy by himself. The **HRL** baseline is implemented by combining the high-level and low-level policies that are trained separately.

B.3.4 Low-level Policy

Figure B-7 illustrates the network architecture for the low-level policy. We use the symbolic observation (only the visible object nodes) as input, and encode them in the same way as Figure B-6 does. We encode two object classes in the given subgoal sg (i.e., a predicate) through word2vec encoding yielding two 128-dim vectors. We then concatenate these two vectors and feed them to a fully connected layer to get a 128-dim goal encoding. Based on the goal encoding, we further get two attention vectors, σ_{object} and σ_{type}. Each element of the attention vectors ranges from 0 to 1. For each object node, we use the element-wise product of σ_{object} and its node embedding to get its reshaped representation. Similarly, we can get the reshaped context representation by an element-wise product of the context embedding and σ_{type}. This is inspired by a common goal-conditioned policy network architecture [29, 130], which helps extract

state information relevant to the goal. From each reshaped node representation, we can get a scalar for each object representing the log-likelihood of selecting that object to interact with for the current action. After a softmax over all the object logits, we get the object selection policy $\pi_{\text{object}}(k|o^t, sg)$, where k is the index of the object instance selected from all visible objects (which also includes "Null" for actions that do not involve an object). For encoding the history, we feed the reshaped context representation to an LSTM with 128 hidden units. Based on the latent state from the LSTM, we get i) the action type policy $\pi_{\text{type}}(a|o^t, sg)$, which selects an action type (i.e., "open," "close," "grab," "put," "walk," or "follow"), and ii) the value function $V(o^t, sg)$. The sampled k and a jointly define the action for the AI agent. Note that some sampled combinations may not be valid actions, which will not be executed by the VirtualHome environment.

In addition to the policy and value output, we also build a binary classifier for each visible node to predict whether it is close enough for the agent to interact with according to the symbolic graphs. This closeness prediction serves an auxiliary prediction which helps the network learn a better state representation and consequently greatly improves the sample efficiency.

In each training episode, we randomly sample a predicate from the complete goal definition as the final goal of the agent. The agent gets a reward of 0.05 for being close to the target object and/or location, and a reward of 10.0 when it grabs the correct object or puts it to the correct location. Note that when training the low-level policy, we set up a single-agent environment to ensure that the AI agent can learn to achieve a predicate by itself.

We adopt a 2-phase curriculum learning similar to [130]: In the first phase, we train a policy for grabbing the target object indicated in the goal. During this phase, a training episode terminates whenever the agent grabs the correct type of object. In the second phase, we train another policy which learns to reuse the learned grabbing policy (which is deployed whenever the "grab" action type is sampled) to get the goal object and then put the grabbed object to target location specified in the goal.

We use off-policy advantage actor-critic (A2C) [96] for policy optimization. The

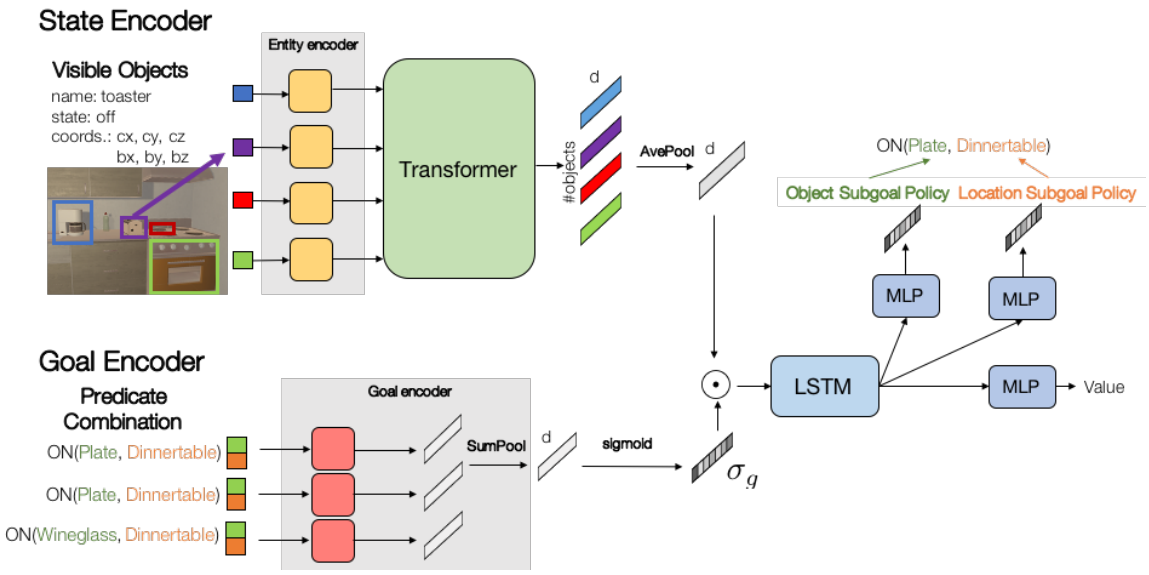


Figure B-8: Network architecture the high-level policy for the **Hybrid** and the **HRL** baselines.

network is updated by RMSprop [148] with a learning rate of 0.001 and a batch size of 32. The first phase is trained with 100,000 episodes and the second phase is trained with 26,000 episodes.

B.3.5 High-level Policy

As Figure B-8 depicts, the high-level policy (used by **Hybrid** and **HRL** baselines) has a similar architecture design as the low-level policy. Compared with the low-level policy, it does not need to define object selection policy; instead, based on the latent state from the LSTM, it outputs the policy for selecting the first and the second object class in a predicate to form a subgoal[1]. It also augments the goal encoder in the low-level policy with a sum pooling (i.e., Bag of Words) to aggregate the encoding of all predicates in a goal, where predicates are duplicated w.r.t. their counts in the goal definition (e.g., in Figure B-8, `ON(plate, dinnertable)` appears twice, which means there are should be 2 plates on the dinnertable). Similar to the low-level policy, we get an attention vector σ_g from the goal encoding to reshape the state representation.

[1]Note that this is different from the subgoals generated from the high-level planner (Section B.1.1), which would specify object instances.

In total, the network has three outputs: the object subgoal policy for sampling the object class name in the subgoal, the location subgoal policy for sampling the target location class name in the subgoal, and a value function.

The high-level policy is trained with a regression planner deployed to find a low-level plan for reaching that subgoal. Note that the regression planner searches for a plan based on a state sampled from the agent's belief maintained by a belief module discussed in Section B.1.1. It will also randomly select object instances from the sampled state that fit the defined object classes in the subgoals sampled from the high-level policy.

Similar to the low-level policy, we use off-policy A2C for policy optimization, and the network is updated by RMSprop with a learning rate of 0.001 and a batch size of 16. We first train the high-level policy in a single-agent setting where the AI agent is trained to perform a task by itself; we then finetune the high-level policy in the full training setting where the human-like agent is also present and works alongside with the AI agent. During training, we always provide the ground-truth goal of Alice to the AI agent.

B.4 Additional Details of Human Experiments

B.4.1 Human subjects

Both the collection of human plans as well as the evaluations in our user studies were conducted by recruited participants, who gave informed consent.

B.4.2 Procedure for Collecting Human Plans

To collect the tasks for both experiments, we built a web interface on top of VirtualHome, allowing humans to control the characters in the environment. Specifically, the subjects in our human experiments were always asked to control Alice. At every step, humans were given a set of visible objects, and the corresponding actions that they could perform with those objects (in addition to the low-level actions), matching

the observation and action space of the human-like agent. When working with an AI agent, both the human player and the AI agent took actions concurrently.

In both experiments, human players were given a short tutorial and had a chance to get familiar with the controls. They were shown the exact goals to be achieved, and were instructed to finish the task as fast as possible. For each task, we set the same time limit, i.e., 250 steps. A task is terminated when it exceeds the time limit or when all the goals specified have been reached.

The 30 tasks used in the human experiments were randomly sampled from the test set and were evenly distributed across 5 task categories (i.e., 6 tasks for each category).

In Experiment 2, each subject was asked to perform 7 or 8 trials. We made sure that each subject got to play with all three baseline AI agents in at least 2 trials.

B.4.3 Subjective Evaluation of Single Agent Plans

To evaluate whether people think the human-like agent behaves similarly to humans given the same goals, we recruited another 8 subjects. We showed each subject 15 videos, each of which is a video replay of a human or the human-like agent performing one of the 30 tasks (we randomly selected one human video and one built-in agent video for each task). For each video, subjects were given the goal and asked to rate how much they agreed with the statement, "the character in the video behaves similarly to a human given the same goal in this apartment," on a Likert scale of 5 (1 is "strongly disagree," 3 is "neutral," and 5 is "strongly agree")[2]. The average ratings for the characters controlled by the human-like agent and by the real humans are 3.38 ($\pm$0.93) and 3.72 ($\pm$0.92) respectively. We found no significant difference between the ratings for the human-like agent's plans and the ratings for the real humans' plans in our tasks, as reported by a paired, two-tailed t-test ($t(29) = -1.35$, $p = .19$). This demonstrates that the proposed human-like agent can produce plans that are similar to real humans' plans in our challenge.

[2]Since we focus on the agents' plans in this work, users were instructed to focus on the actions taken by the agents, rather than the graphical display of their body motion.

Based on the free responses collected from the subjects who rated these videos, human plans look slightly more efficient sometimes since they do not look for objects in unlikely places and avoid moving back and forth between rooms frequently. The human-like agent behaves similarly in most of the time but would occasionally search through the rooms in a counter-intuitive order due to its bounded rationality and the fact that plans are sampled stochastically.

B.4.4 Additional Quantitative Analyses of Human Experiment Results

To evaluate whether the performance of a baseline AI agent helping the human-like agent reflects the performance of it helping real humans, we conduct paired, two-tailed t-test for the three baselines in Experiment 2 based on their cumulative rewards. For $\mathbf{HP}_{\mathrm{RG}}$, there is a significant difference between helping the human-like agent and helping real humans ($t(29) = -2.36$, $p = .03$) as discussed in Section 6.6. However, there is no significant difference for **HP** ($t(29) = -1.78$, $p = .1$) and **Hybrid** (($t(29) = -0.5$, $p = .62$)). This validates that, in general, collaboration with the human-like agent is comparable to collaboration with real humans. Given these analyses, the training and evaluation procedure[3] presented in this paper is both scalable and comprehensive.

[3]I.e., i) training AI agents with the human-like agent, and then ii) evaluating them both with the human-like agent (in a larger test set), and with real humans (in a smaller but representative test set).

Appendix C

Online Probabilistic Assistive Agents

C.1 Assumptions and Benchmarks in Prior Work

Ref.	Task	Domain	Obs.	#Actions	Online goal inf.	#Goals	Unseen env.	Scene graph repr.
[39]	Doorman	Gridworld	Full	9	Yes	3	No	Yes
[39]	Kitchen	2D symbolic game	Full	6	Yes	8	No	Yes
[27]	USAR[1]	Gridworld	Full	7[2]	No	13	No	Yes
[64]	Grasping	Lab environment	Full	3	Yes	3	No	Yes
[64]	Feeding	Lab environment	Full	3	Yes	3	No	Yes
[26]	Overcooked	Gridworld	Full	6	No	N/A	No	Yes
[159]	Overcooked	Gridworld	Full	5	No	N/A	No	Yes
[37]	Goal reaching	Gridworld	Full	5	Yes	25	No	Yes
[37]	Lunar landing	Atari game	Full	6	Yes	1D space[3]	No	No
[110]	WAH	3D virtual apt.	Partial	167	No	N/A	Yes	Yes
Ours	Online WAH	3D virutal apt.	Full	167	Yes	661	Yes	Yes

Table C.1: A summary of the assumptions and benchmarks in the prior work and in our work. Specifically, for each task, we summarize the following information: (1) the domain; (2) the observability assumption; (3) the action space; (4) whether there is a need for online goal inference; (5) the goal space if there is online goal inference; (6) whether there is generalization evaluation in unseen environments; (7) and whether the scene graph representation can be applied.

We summarize problem setups, assumptions, and benchmarks in the prior work on AI assistance in Table C.1. Compared with the prior work, our work proposes a far more challenging benchmark in much more realistic environments with significantly larger action and goal spaces. The assumptions in our problem setup are also not

[1]Urban Search and Rescue

[2]The action space is fully specified in the paper; we estimate that there are 7 actions based on the description in the paper.

[3]The goals are landing locations in a 1D continuous space.

stronger than those of the prior work. Moreover, our evaluation focuses on the generalization in unseen environments unlike the evaluations in the majority of the prior work. Last but not the least, the scene graph representations can also be applied to almost all the tasks in the prior work, except for the Atari game domain. This comparison suggests that the evaluation in our Online Watch-And-Help challenge is a stronger indicator of how well an online assistance method performs in complex environments compared with the existing benchmarks. It also shows that the assumptions in NOPA are reasonable for a broad range of tasks.

C.2 Details on the Online Watch-And-Help Challenge Setup

C.2.1 Task definitions

Table C.2 summarizes 5 task categories used in the challenge and provides a description of how to sample goals for each of them. All goals allow an ideal observer to predict at least one predicate before a task is finished, so that an intelligent helper agent is always guaranteed to have a chance to offer effective help in our online assistance setting. These goals demonstrate different *levels* of uncertainty. For instance, there is a single possible goal for the task *get snacks*, but *put dishwasher* has 315 unique possible goals. In addition, these goals also exhibit different *types* of uncertainty common in real-world scenarios, such as the objects that are needed (stocking the fridge may require to put different groceries in the fridge), the quantity of objects (e.g., setting up a table for 1 to 3 people), or the target locations (e.g., an agent could set up a kitchen table in the kitchen or a coffee table in the living room). In total, there are 11 types of goal objects and 5 types of target locations, creating 119 unique combinations of goal predicates and their counts.

Task Name	Goal definition	Goals
Set table	Set number of people N ∼ U(1,3) Set OBJ ∼ choice([waterglass, wineglass]) Set LOC ∼ choice([kitchentable, coffeetable]) Put N plate, N fork, N OBJ on LOC	12
Put dishwasher	Set OBJ_POOL = [fork, plates, waterglass, wineglass] Set N ∼ U(3,7) Put N objects from OBJ_POOL on dishwasher	315
Stock fridge	Set OBJ_POOL = [salmon, apple, cupcake, pudding] Set N ∼ U(3,7) Put N objects from OBJ_POOL in fridge	315
Prepare meal	Set number of people N ∼ U(1,3) Set OBJ ∼ choice([cupcake, pudding]) Set LOC ∼ choice([kitchentable, coffeetable, stove]) Put N salmon, N apple, N OBJ on LOC	18
Get snacks	Put 1 remote, 1 condiment, 1 chips on coffeetable	1

Table C.2: Overview of the challenge tasks. For a given task type, we sample an instantiation of the task according to the task definition. The third column shows some examples of tasks.

C.2.2 Observation space

Both the main agent and the helper agent receive symbolic observations of the environment, represented as a scene graph, with nodes representing objects, and edges representing spatial relationships between them. Unlike the original Watch-And-Help (Chapter 6), the agents have full observability. That is, at any time, they have information about the states of all objects and agents in the environment. In the future, we intend to extend the current challenge to a partial observability setting.

C.2.3 Action space

Agents can navigate in the environment and interact with the objects in it. Each action consists of a verb, indicating the type of action, and an index, indicating which object the agent should interact with. For the agent to interact with an object in the environment, the object needs to be in the same room as the agent, and not inside a closed container. This means that, while agents can see all objects throughout an episode, they can only interact with the ones that they can physically

reach at each step. Agents can navigate towards different rooms and objects in the environment, moving a certain distance towards the object or the room every time the call a navigation action. The main agent moves 1 meter for every navigation action, whereas the helper agent moves a distance of 3 meters. This allows the helper to be able to move around faster and thus ensures that it can provide effective assistance.

C.2.4 Built-in planner

The built-in planner is the same as the one provided in the original Watch-And-Help challenge (Chapter 6). It uses MCTS to search for subgoals (note that these are different than the helping subgoals that the helper agent would consider); for a given subgoal, it uses a regression planner to search for actions to reach the subgoal. The subgoal space includes all possible objects the main agent may grab and all the possible locations the main agent can put the grabbed objects to in the entire environment.

C.3 Implementation Details

C.3.1 Working Example

Figure C-1 shows a working example of NOPA. NOPA first produces a set of goal proposals from the GPN. In this example, it predicts that part of the goal could be putting an apple (the red node) onto the kitchen table (the light brown node connected by a pink edge) or putting an apple into the fridge (the green node connected by a dark blue edge), among goal predicates that involve other objects. It then uses inverse planning to reject proposals that are inconsistent with the observed main agent's actions. For instance, the action does not appear in the plan for the second particle, so the corresponding particle is then rejected. NOPA evaluates all helping subgoal candidate in the remaining particles and selects the most valuable one based on the value function. In this example, for all the final goals, the main agent has to grab the apple regardless of the goal location; and the helper agent can achieve that subgoal

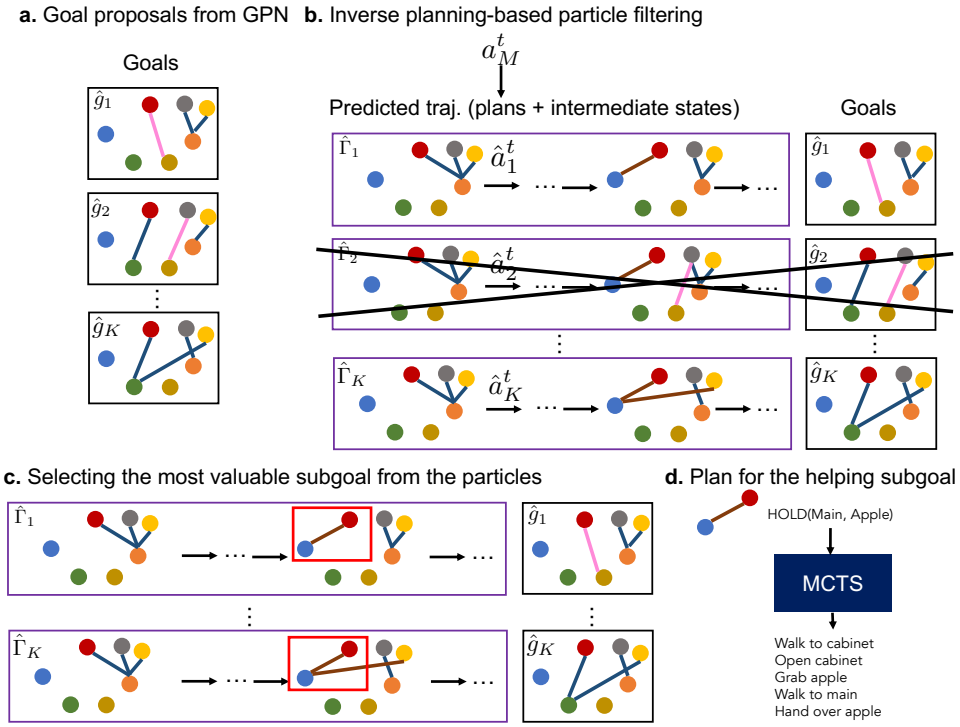


Figure C-1: A working example of NOPA. (**a**) Sampling goal proposals from the GPN. (**b**) Predicting the future plans and trajectories for the proposed goals, and rejecting the hypotheses if the observed main agent's action does not appear in the corresponding plans. (**c**) Selecting the most valuable helping subgoal (the edge highlighted in the red box) based on the value function. (**d**) Planning for the helper agent's actions to reach the selected helping subgoal.

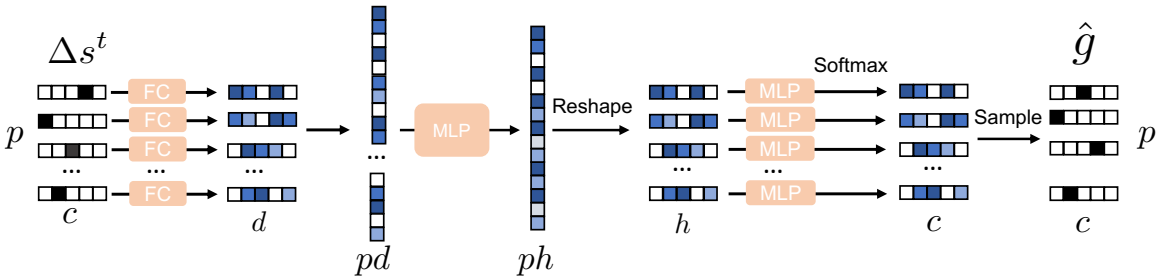


Figure C-2: The architecture of the goal proposal network. Δs^t is a matrix encoding the difference between the predicate counts in the states s^t and s^0. p is the number of all predicate types, c is the maximum number of count, and d and h are dimensions of intermediate layers.

faster than the main agent. Therefore, the predicate `HOLD(Main, Apple)` produces a high value based on the value function. Finally, NOPA uses an MCTS planner to generate the helping actions given the helping subgoal.

C.3.2 Goal Proposal Network

Figure C-2 illustrates the architecture of the goal proposal network used in NOPA. We represent the state at each step, s_t, as a vector of counts of different predicates. We then represent the change of state from s^0 to s^t as the change in the counts of different predicates. This change is denoted as Δs^t, where each row is a one-hot vector, indicating the change in the count of a specific predicate. Note that the predicates here are about the locations of the objects. Therefore, if the count of one predicate increases, the count for another predicate will decrease accordingly. E.g., if an apple is moved from the fridge to the kitchen table, then the count for `IN(apple, fridge)` will decrease and the count for `ON(apple, kitchentable)` will decrease. Since we want to focus on where the objects are moved to for the goal inference, in Δs^t, we only record the increase in the count of each predicate, and set any decrease in the count to 0. We feed each row of Δs^t to a fully-connected layer with a dimension of d. We flatten the resulting matrix into a vector with pd dimensions, which is encoded to a vector of ph dimensions by an MLP (N_1 layers, h dimensions). The vector is then reshaped to a $p \times h$ matrix. An MLP (N_2 layers, h dimensions) is applied to each row followed by a softmax activation. This gives us a proposal distribution of the count for each predicate. The final goal proposal $\hat{g}$ can be sampled from these p distributions, representing the intended change in the count of all predicates at the end of the task. We train the GPN with a cross-entropy loss based on the counts of the predicates in the ground-truth goals and use Adam [71] for the optimization.

Dataset to Train and Test the Goal Proposal Network

Using the task definitions in Table C.2, we sample a set of initial environments and goals, and run our planning-based agent to generate a task demonstration for each sampled environment and goal. This results in a training set of 6000 demonstrations, and a test set of 100 demonstrations. Note that the 100 demonstrations from the test set correspond to the environments and tasks included and the challenge, and are used to measure the speed up of the proposed agents.

The training set has 375 unique goals (e.g. combinations of predicates and counts), or 41 unique goals if we do not consider the counts. The testing set has 53 unique goals, or 34 when we do not consider the numbers of objects. On average, both the training set and the testing set contain 4.6 predicates per episode, and 2.5 unique predicates without considering the numbers of objects.

C.3.3 Details of NOPA

We follow the implementation described in [135] for the MCTS planner used in the helping planner. In particular, the MCTS planner searches for macro actions including get a certain object as well as put a certain object to a specified location. We then use regression planing to search for a sequence of actions to achieve each macro action.

We define the distance metric between two states used in the value function, $\mathcal{D}$, as the difference in the counts of the predicates appearing in the states. For instance, if one state has 2 `ON(apple, kitchentable)` and the other has 1 `ON(apple, kitchentable)` and 1 `IN(apple, fridge)`, then the distance between these 2 states is 1.

C.3.4 Hyperparameters for NOPA

We summarize the hyperparamters used for NOPA in our experiments in Table C.3.

C.3.5 Computational Resources

We use a single GPU (NVIDIA GeForce GTX 1080) and a 32-core CPU. It takes 1 - 3 sec for NOPA to update the helping action for a step, which is fast enough for real-time interactions with real human players.

C.4 Further Analysis of Results

To compare the performance of NOPA against baselines in tasks with different levels of difficulties, we show the speedup (Figure C-3) as well as the F1-scores of goal

Hyperparameter	Value
K	20
$T_{\max}$	250
T_{prop}	15
w_r	1
w_c	1
w_d	5
$L_{\max}$	100
p	136
c	9
d	100
h	128
N_1	2
N_2	4
Learning rate for GPN	0.0009
Batch size	256
The number of epochs	300

Table C.3: Hyperparamters used in the experiments.

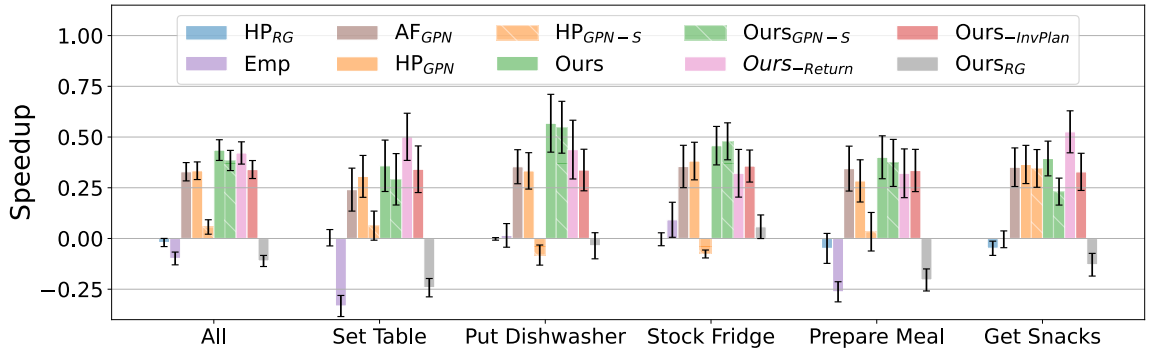


Figure C-3: Speedup of different methods (striped bars indicate using the small training set). Errors are standard errors. We show the performance in the overall test set and in each type of task.

inference (Figure C-4) in each type of task. The two most difficulty task types are *Put Dishwasher* and *Stock Fridge* since there much more possible goals in these two types of tasks compared with other types, causing a higher degree of uncertainty in the goal inference. The results shown in Figure C-3 and Figure C-4 suggest that NOPA has the largest improvement margin over baselines when the goal inference is uncertain. This further demonstrates the benefit provided by NOPA for online assistance in complex settings where the goal space is large.

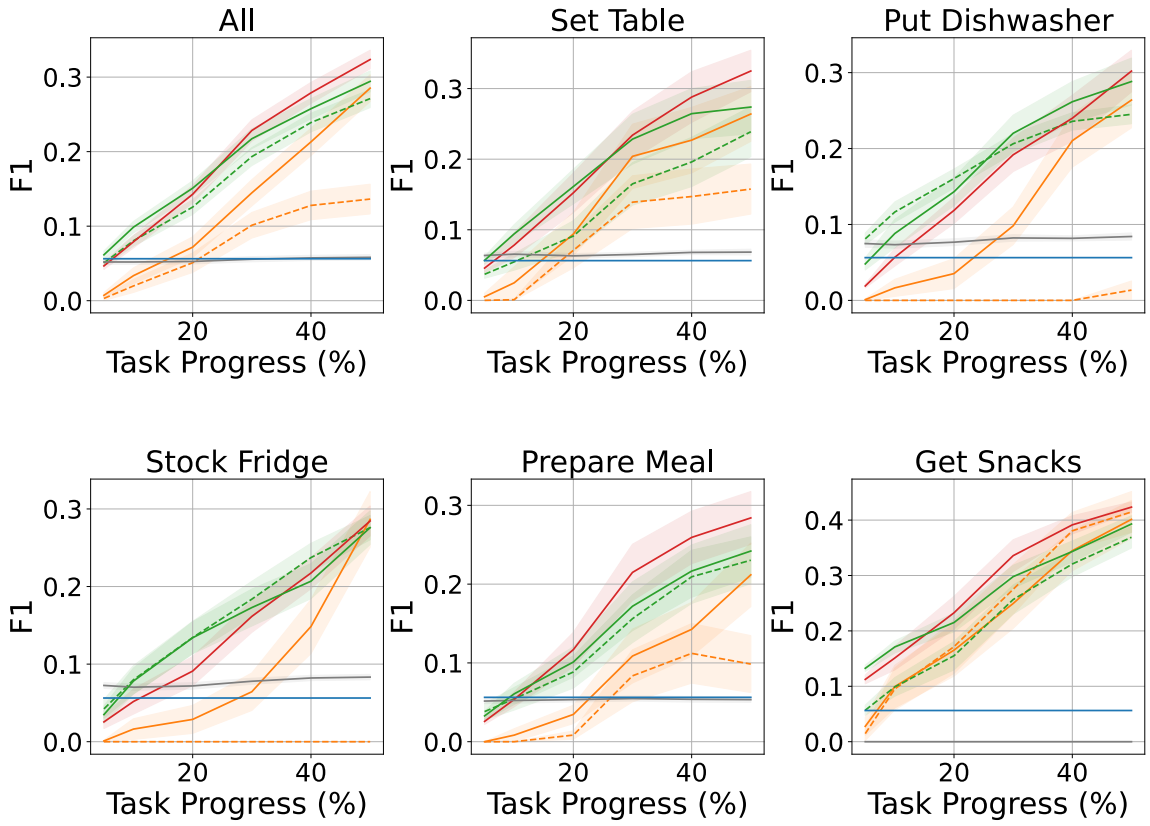


Figure C-4: F1-scores of the predicted goal over the course of a task. The x axis is normalized in proportion to the number of steps needed for the main agent to perform each task alone. The curves show the means and the shaded regions show the standard errors. We show the performance in the overall test set and in each type of task.

Moreover, we observe that $\mathbf{HP_{GPN}}$ performs much better than $\mathbf{HP_{GPN\text{-}S}}$ but **Ours** and $\mathbf{Ours_{GPN\text{-}S}}$ are comparable, especially in the two most difficult task types. This means that the helping performance of $\mathbf{Ours_{GPN\text{-}S}}$ can match with that of **Ours** in complex settings, even though the GPN in **Ours** is trained on a much larger training set and has a significantly higher F-1 score for the goal inference than GPN-S when used alone. This shows that NOPA can drastically increase the sample efficiency in complex settings thanks to its ability to integrate a goal proposal network trained with a small amount of data with other components of the model (in particular, the inverse planning-based particle filtering and the uncertainty-aware helping planner) and maintain a high performance. Since sample efficiency is critical for real world applications where human behavior data are scarce, this result suggests potentials for

applying NOPA to real world systems for online assistance.

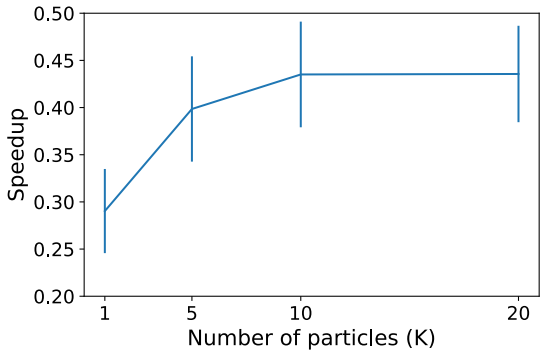


Figure C-5: The helping performance of our full model with different numbers of the particles (K).

To test how sensitive NOPA is to the number of the particles (K) used in the online goal inference, we compare the performance of NOPA with different numbers of particles. As shown in Figure C-5, the performance of the NOPA is not sensitive to K, as long as K is not too small.

C.5 Details of Human Experiments

C.5.1 Participants

We recruited 10 participants (mean age = 32.3; 4 female), all with college degrees. The participants had with no prior exposure to our system. Participants first completed a consent form and were then given a tutorial for interacting with the user interface[4]. They also had an opportunity to practice in an example task before the trials started.

C.5.2 More Details of the Statistical Testing

We measure whether there is a significant difference in the performance of NOPA under the two conditions (helping a main agent controlled by the built-in planner and helping a main agent controlled by human players). The residuals between both

[4] https://docs.google.com/document/d/1xJDmrx55RCaOq60B_CtZNABVFdfGccSOwMj6WRhMVhg/edit?usp=sharing

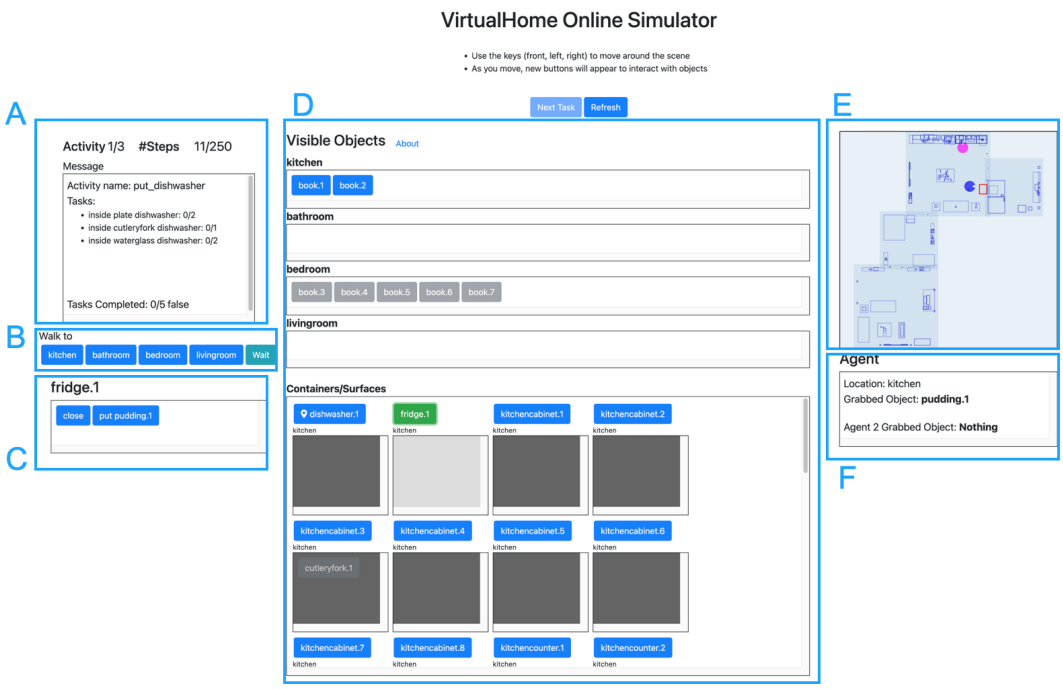


Figure C-6: Overview of the interface used to test helper agents assisting humans. We annotate different panels (A - F) to explain the layout and the relevant information displayed in each part of the interface.

samples are normally distributed, as reported by a D'Agostino-Pearson Test ($t(10) = 2.65, \rho = 0.27$). We thus perform a paired t-test between the speedups of NOPA under the two conditions. The result shows that there is no significant difference in the performance of NOPA under these two conditions ($t(9) = 0.87, \rho = 0.40$).

C.5.3 Interface

Figure C-6 shows a screenshot of the interface for the human experiment. Players are given a description of the goal (the predicates and their counts) and the progress, as indicated by the panel A. At any time, they can see all the objects in the environment, represented as buttons with object names (panels B, C, and D). Buttons are only enabled when objects can be interacted with. Green buttons indicate that the agent is close to the corresponding objects. In the top-right panel (E), users can see a floor plan of the apartment depicting the locations and the facing directions of the agents. The main agent is shown in blue and the helper agent is shown in magenta. Users

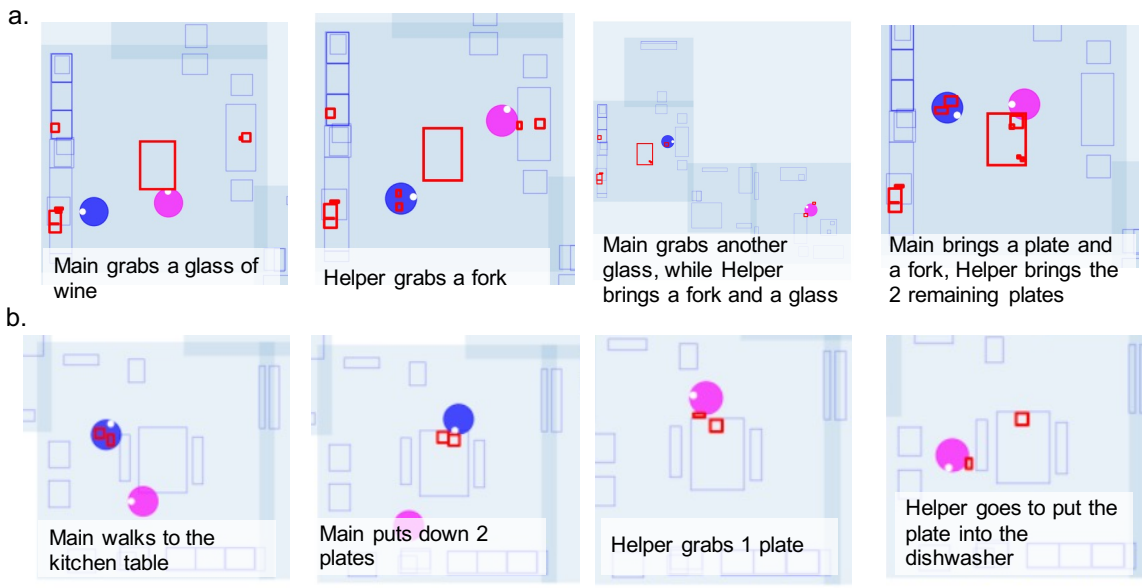


Figure C-7: Examples of helping plans in the AI helping human condition. Here we show the floor plans; the locations and the facing directions of the main agent (blue) and the helper agent (magenta); and the relevant objects (red bounding boxes). (**a**) An example of successful assistance by NOPA (the goal is to set up a kitchen table for 3 persons); (**b**) a failure example of $\mathbf{HP_{RG}}$ caused by conflicting goals (the helper agent tries to put plates to the dishwasher while the main agent tries to put plates to the kitchen table).

can also check what objects agents have in their hands from panel F.

We adopt this symbolic design instead of directly providing camera views to human players to ensure that they can easily see the full state, matching with the full observability setting of the challenge.

C.5.4 Visual Interface

While all human experiments are conducted using a symbolic representation of the environment, mimicking the setting of the planning-based agents, we also provide, for future use, an interface to perform activities with visual observations. Figure C-8 shows an overview of the interface. Users are provided with a first-person view of an agent in VirtualHome which is streamed in real time via WebRTC [87]. Users can control agents by clicking on objects within the field of view and choosing which action they want to do with the selected object. This setting allows for more realistic assistive

scenarios, where humans need to move around a scene to find required objects.

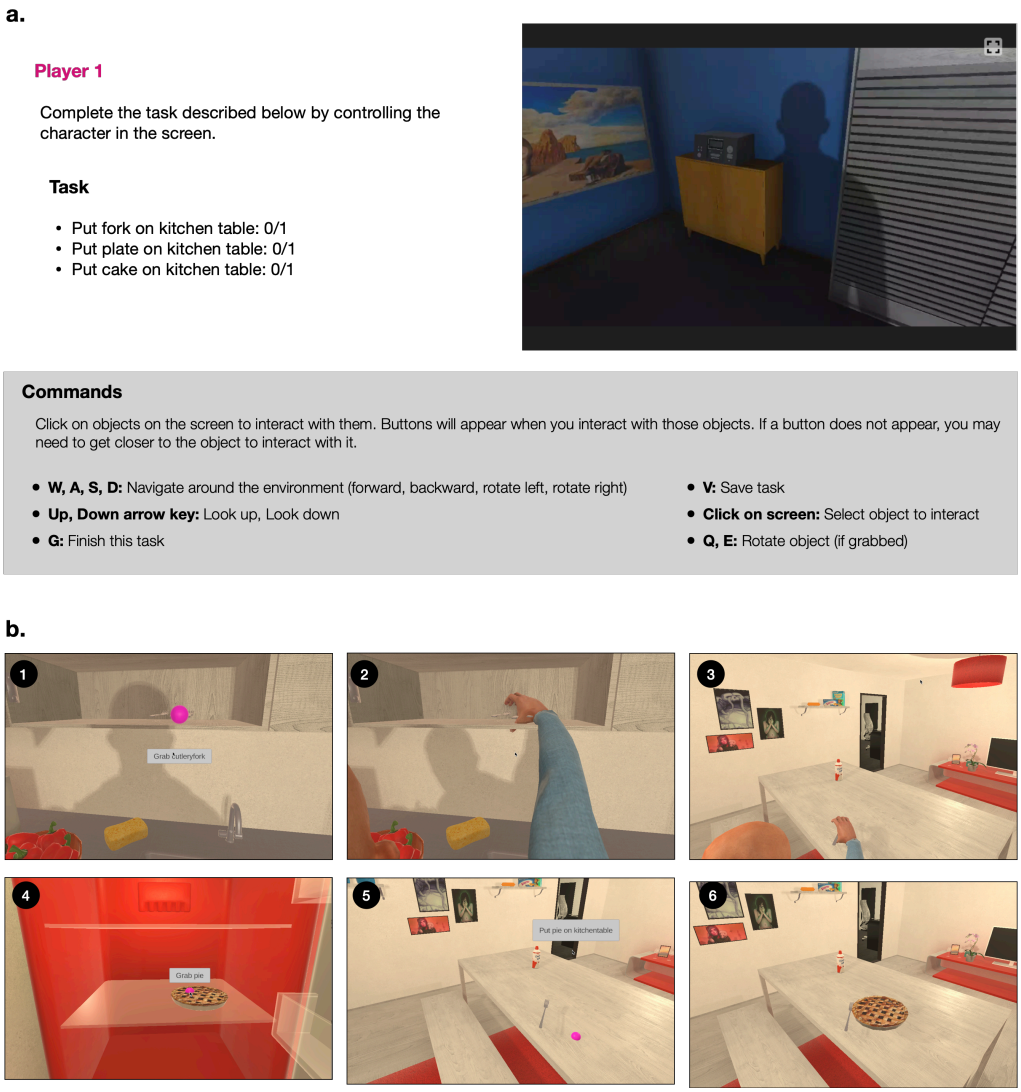


Figure C-8: Overview of the Visual Interface to collect interaction data (**a**) and visualization of an episode (**b**). Users connect to the simulator using a web server, and receive a video stream corresponding to the first-person view of an agent in the environment (**a**). When they click on an object, a set of buttons appears, allowing to chose an action to perform on the object (**b**).

C.5.5 Qualitative Results

Figure C-7 depicts typical examples between a helper agent and a main agent controlled by a human player, including an example of successful assistance by NOPA and a failure example of $\mathbf{HP}_{\mathbf{RG}}$ where the helper agent attempts to undo what main agent has achieved due to conflicting goals.