Integrating ProgPrompt and VirtualHome for Autonomous Robot Task Planning

Abstract

Large Language Models (LLM) can encode large amounts of semantic information covering many processes in everyday life. Such knowledge can be useful for robots that aim to act according to high-level instructions expressed in natural language. However, a major weakness of language models is that they cannot know exactly how real-world phenomena work. For example, asking for a language model to describe dinner might result in a plausible description, but not for an agent such as a robot that needs to perform this task in a specific environment.

This work explores the integration of the ProgPrompt method and the VirtualHome simulation environment to improve the ability of autonomous robotic agents to navigate unknown environments and perform tasks. ProgPrompt leverages LLMs for hierarchical task planning, while VirtualHome provides a simulated home environment for testing and training. The synergy between these technologies offers promising developments in autonomous robotics, especially in task execution and environmental interaction.

Introduction

Recent advances in the training of large language models have led to the emergence of systems that can generate complex text based on prompts, answer questions, and even dialog on a wide variety of topics [1]. These models draw large amounts of information from texts extracted from the web, and we might wonder whether the knowledge of everyday tasks encoded in such models could be used by robots to perform complex tasks in the real world. But how can embodied agents acquire and use the knowledge of LLMs for physically based tasks? This question poses a major challenge [2].

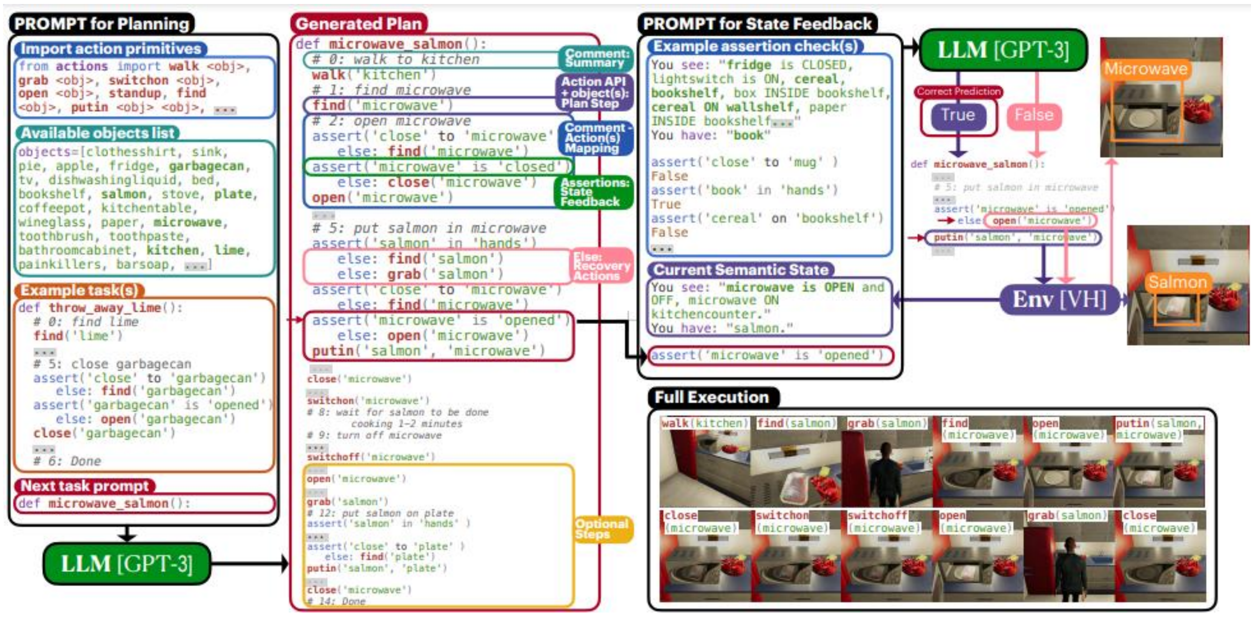
With prompt engineering, an LLM can decompose high-level instructions into subtasks, but it cannot do so without the context of the robot's capabilities and what it can do given the current state of the robot and the environment [3]. Starting from this example, explore how to extract information from LLMs to enable an embodied agent, such as a robot, to follow high-level textual instructions.

Autonomous robots operating in unstructured environments require sophisticated task planning and environmental understanding. Traditional methods often demand extensive domain knowledge and predefined action sequences. The emergence of LLMs has introduced new avenues for dynamic task planning. ProgPrompt [4] utilizes LLMs to generate executable task plans, and when combined with the VirtualHome [5] simulation environment, it provides a comprehensive framework for developing and testing autonomous agents.

ProgPrompt Methodology

ProgPrompt is a prompting technique that guides LLMs to generate structured task plans using programming language constructs. By providing the LLM with a programmatic context, including available actions, environmental objects, and example task sequences, ProgPrompt enables the generation of executable plans tailored to specific environments and tasks. This approach contrasts with end-to-end learning models by explicitly defining reasoning and planning processes, enhancing interpretability and adaptability.

ProgPrompt employs a programmatic structure to guide the LLM in generating task plans. By framing the task planning problem as a programming task, the LLM is prompted to produce Python code that outlines the sequence of actions the robot should perform [4]. This approach leverages the structured syntax and semantics of programming languages to constrain the LLM's output, ensuring that the generated plans are both syntactically correct and semantically meaningful within the robot's operational context. The use of Python as the target language allows for the direct execution of the generated plans, facilitating seamless integration with the robot's control systems.



```
from actions import walk <obj>,
grab <obj>, switchon <obj>,
open <obj>, standup, find
<obj>, putin <obj> <obj>, ...
```

```
objects=[clothesshirt, sink,
pie, apple, fridge, garbagecan,
tv, dishwashingliquid, bed,
bookshelf, salmon, stove, plate,
coffeepot, kitchentable,
wineglass, paper, microwave,
toothbrush, toothpaste,
bathroomcabinet, kitchen, lime,
painkillers, barsoap, ...]
```

```
def throw_away_lime():
  # 0: find lime
  find('lime')
  ...
  # 5: close garbagecan
  assert('close' to 'garbagecan')
     else: find('garbagecan')
  assert('garbagecan' is 'opened')
     else: open('garbagecan')
  close('garbagecan')
  ...
  # 6: Done
```

```
def microwave_salmon():
```

```
def microwave_salmon():
  # 0: walk to kitchen
  walk('kitchen')
  # 1: find microwave
  find('microwave')
  # 2: open microwave
  assert('close' to 'microwave')
     else: find('microwave')
  assert('microwave' is 'closed')
     else: close('microwave')
  open('microwave')
  ...
  # 5: put salmon in microwave
  assert('salmon' in 'hands')
     else: find('salmon')
     else: grab('salmon')
  assert('close' to 'microwave')
     else: find('microwave')
  assert('microwave' is 'opened')
     else: open('microwave')
  putin('salmon', 'microwave')
  ...
  close('microwave')
  ...
  switchon('microwave')
  # 8: wait for salmon to be done
        cooking 1-2 minutes
  # 9: turn off microwave
  ...
  switchoff('microwave')
  ...
  open('microwave')
  ...
  grab('salmon')
  # 12: put salmon on plate
  assert('salmon' in 'hands' )
  ...
  assert('close' to 'plate' )
     else: find('plate')
  putin('salmon', 'plate')
  ...
  close('microwave')
  # 14: Done
```

```
You see: "fridge is CLOSED,
lightswitch is ON, cereal,
bookshelf, box INSIDE bookshelf,
cereal ON wallshelf, paper
INSIDE bookshelf..."
You have: "book"

assert('close' to 'mug' )
False
assert('book' in 'hands')
True
assert('cereal' on 'bookshelf')
False
...
```

```
You see: "microwave is OPEN and
OFF, microwave ON
kitchencounter."
You have: "salmon."
```

```
assert('microwave' is 'opened')
```

```
def microwave_salmon():
  ...
  # 5: put salmon in microwave
  ...
  assert('microwave' is 'opened')
   else: open('microwave')
  putin('salmon', 'microwave')
  ...
```

Figure 1. Progprompt Task Planning

Experiments were conducted to evaluate the effectiveness of ProgPrompt in generating feasible and efficient task plans for robotic agents. The results show that ProgPrompt successfully reduces the need for extensive domain knowledge specification by enabling LLM to generate contextually appropriate action sequences directly from natural language instructions. The generated plans are executable and compatible with the robot's capabilities and environmental constraints, demonstrating the practical applicability of the proposed approach.

ProgPrompt offers a novel approach to robot task planning by leveraging Big Language Models to generate structured, executable task plans in the form of Python programs. This method addresses the limitations of existing approaches by reducing the need to specify extensive domain knowledge and ensuring that the generated plans are executable within the operational context of the robot. Experimental results confirm the effectiveness of ProgPrompt in generating reliable and efficient task plans and highlight its potential for practical applications in robotic systems.

VirtualHome Simulation Environment

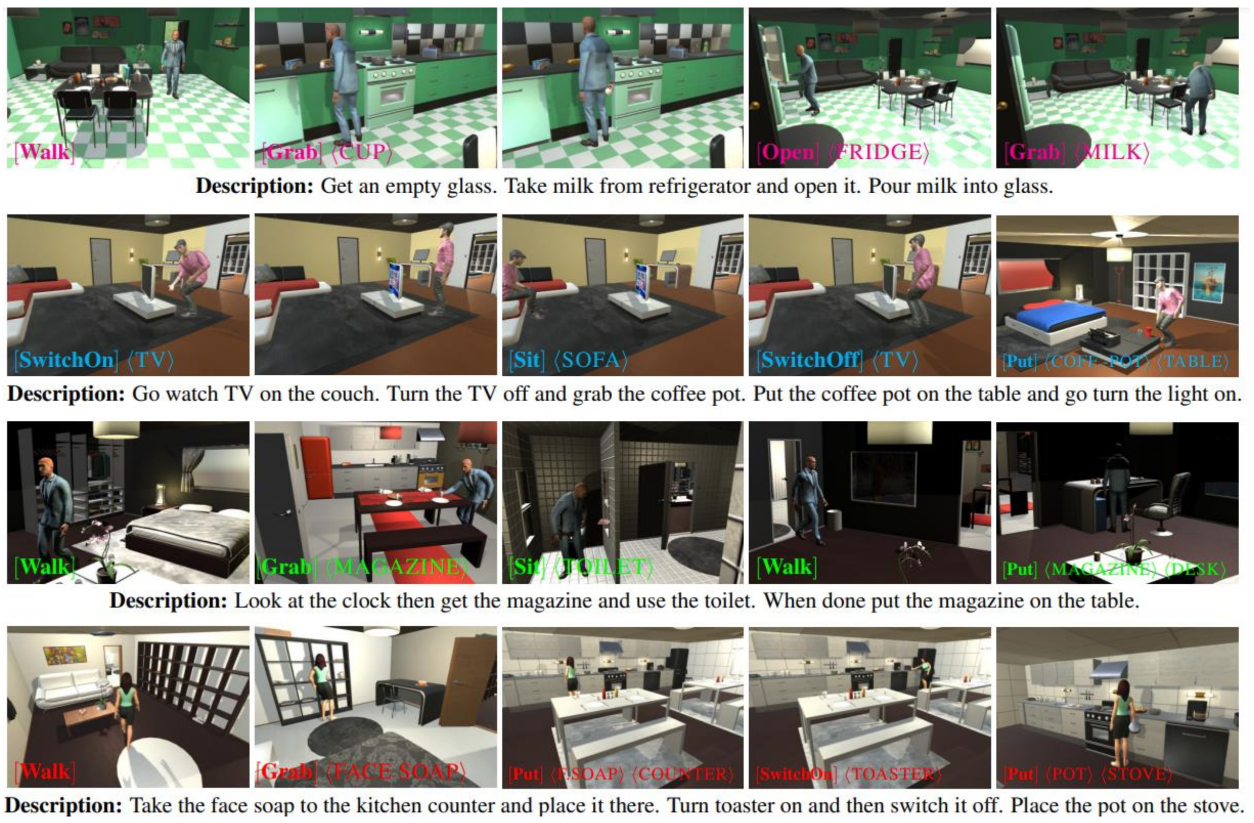
VirtualHome is a multi-agent platform designed to simulate complex household activities through the use of programs—sequences of atomic actions and interactions [5]. By representing activities as programs, VirtualHome provides a clear and executable framework for modeling tasks within a household environment. The platform facilitates the generation of large-scale activity datasets with rich ground-truth information, supporting the training and evaluation of models in areas such as video understanding and task planning. This abstraction outlines the key components of VirtualHome, including its program-based representation, data collection methodology, and simulation capabilities.

Modeling complex household activities presents significant challenges due to the intricate sequences of actions and interactions involved. VirtualHome addresses these challenges by utilizing programs as high-level representations of tasks, offering a non-ambiguous and executable format that artificial agents can follow. This approach not only aids in the clear definition of tasks but also enables agents to perform them within a simulated environment.

In VirtualHome, activities are represented as programs composed of sequences of atomic actions. Each action corresponds to a fundamental interaction within the household environment, such as picking up an object, opening a door, or turning on an appliance. This structured representation allows for the precise modeling of complex tasks by breaking them down into manageable components. Programs provide a clear blueprint for agents to execute tasks, ensuring consistency and reproducibility in simulations.

To build a comprehensive database of household activities, VirtualHome employs a crowd-sourcing approach. Participants use a game-like interface, inspired by educational tools for teaching programming to children, to create programs that represent various household tasks. This method enables the collection of a diverse set of activity programs, capturing a wide range of interactions and sequences that occur in typical home settings. The resulting dataset serves as a valuable resource for training models to understand and generate human-like activity sequences.

VirtualHome is implemented using the Unity3D game engine, providing a realistic and interactive 3D environment for agents to perform tasks. The platform includes implementations of common atomic interactions, allowing agents to manipulate objects, navigate spaces, and interact with various household items. By executing the collected programs within this simulated environment, VirtualHome can generate extensive activity video datasets with detailed ground-truth annotations. These datasets are instrumental for developing and evaluating models in video understanding, activity recognition, and task planning.



Description: Get an empty glass. Take milk from refrigerator and open it. Pour milk 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 and 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 2. Executing generated programs from descriptions in VirtualHome

Integration of ProgPrompt and VirtualHome

Combining ProgPrompt with VirtualHome involves using the former to generate task plans that the latter can simulate. The process begins with defining the environment's state and available actions within VirtualHome. ProgPrompt then utilizes this information to generate a sequence of actions, which are executed by agents in the simulated environment. This integration allows for the evaluation of task plans in realistic settings, facilitating iterative improvements in both planning and execution.

VirtualHome source codes were downloaded from the repo in this link and installation was done by following the instructions. (https://github.com/xavierpuigf/virtualhome)

A virtual env was created with Conda and ProgPrompt source code was downloaded and installed from this repo. (https://github.com/NVlabs/progprompt-vh)

The following example python code connects to the VirtualHome simulation environment and simulates the agent taking a salmon from the fridge and putting it in the microwave.

```
import time

from simulation.unity_simulator import comm_unity

YOUR_FILE_NAME = "/Users/suatbayir/progprompt-vh/virtualhome/macos_exec.v2.2.3.app"

comm = comm_unity.UnityCommunication(file_name=YOUR_FILE_NAME)

# Start the first environment

comm.reset(0)

# Get an image of the first camera

success, image = comm.camera_image([0])


# Reset the environment

comm.reset(0)

comm.add_character('Chars/Female2')

s, g = comm.environment_graph()


# Get nodes for salmon and microwave

salmon_id = [node['id'] for node in g['nodes'] if node['class_name'] == 'salmon'][0]

microwave_id = [node['id'] for node in g['nodes'] if node['class_name'] == 'microwave'][0]


# Put salmon in microwave

script = [

    '<char0> [walk] <salmon> ({})'.format(salmon_id),

    '<char0> [grab] <salmon> ({})'.format(salmon_id),

    '<char0> [open] <microwave> ({})'.format(microwave_id),

    '<char0> [putin] <salmon> ({}) <microwave> ({})'.format(salmon_id, microwave_id),

    '<char0> [close] <microwave> ({})'.format(microwave_id)

]

comm.render_script(script, recording=True, frame_rate=10)
```

References

[1] Wu, S., Fei, H., Qu, L., Ji, W., & Chua, T. S. (2023). Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*.

[2] Dorbala, V. S., Mullen, J. F., & Manocha, D. (2023). Can an embodied agent find your “cat-shaped mug”? llm-based zero-shot object navigation. *IEEE Robotics and Automation Letters*, *9*(5), 4083-4090.

[3] Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022). Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

[4] Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., ... & Garg, A. (2023, May). Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 11523-11530). IEEE.

[5] Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., & Torralba, A. (2018). Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8494-8502).