Benchmarking PROGPROMPT - Generating Situated Robot Task Plans using Large Language Models

Aryan Dua, 2020CS50475[a], Gurarmaan Singh, 2020CS50426[a]

[a] *COL864 Project, IIT Delhi, , , , ,*

Abstract

Task planning can require defining domain-specific knowledge about the world in which a robot needs to act. To relax this requirement, large language models (LLMs) have been used to score potential next actions during task planning, and even generate action sequences directly, given an instruction in natural language with no additional domain information. However, such methods either require predicting and scoring all possible next steps, or generate text actions that are in-admissible in the current context. PROGPROMPT is a programmatic LLM prompt structure that enables plan generation functional across situated environments, robot capabilities, and tasks. It's key insight is to prompt the LLM with program-like specifications of the available actions and objects in an environment, as well as with example programs that can be executed. Website at progprompt.github. We set-up the code locally, repeat the authors' experiments and experiment with PROGPROMPT on environments, parameters and LLM models not used before.

1. Set-Up Difficulties

PROGPROMPT and VirtualHome both have not been updated for 4 to 5 months, and had been rendered incompatible with each-other, and with openai's current interface. We faced libraries conflicts, improper paths handling, interfacing issues with Unity's rendering executable. Further, we had to tailor API calls to adapt to the deprecated version.

2. Variations Explored

We experimented with the code and the executions by introducing variations along the following dimensions :

1. Number of Examples Provided
2. Quality of 'shots'(examples) provided
3. Newer, Updated models - GPT-3.5 turbo
4. Comments and Feedback provided to the planner
5. Extent of State Feedback
6. Environments (Apartments)
7. Test Sets
8. Metric introduction - Precision

3. Results and Analysis

3.1. Metrics

The Model is evaluated on :

1. **Goal Conditions Recall (GCR)** - Computed as the fraction of Goal Conditions satisfied
2. **Success Rate (SR)** - The fraction of tasks completed.
3. **Executable fraction (Exec)** - The fraction of suggested actions that are admissible.

Other than the 3 above metrics used in the paper, we additionally measure **Precision** - The fraction of commands suggested that are present in the environment. We note that the values of precision are usually high (usually above 0.990), thereby suggesting that the problem of the LLM suggesting actions not even in the action space has effectively been solved.

We agree with the criticism of these metrics by the authors - these assume a hard condition on goal completion, which may take all variants of goal success as successful. We suggest building a dual system paradigm - one LLM plans and executes actions, another scores them!

3.2. Updated Models

The paper evaluates PROGPROMPT on Codex, Davinci and GPT3. Codex has now been deprecated, and Davinci is a legacy model (calls to davinci are charged at $0.02 per token), so we could carry out limited runs with it.

We observe a fascinating improvement of GPT-3.5-turbo over Davinci, and a much greater improvement over babbage!

LLM	GCR	SR	Precision	Exec
gpt-3.5-turbo	0.786	0.5	0.993	0.923
davinci-002	0.664	0.3	0.993	0.878
babbage-002	0.363	0.1	0.994	0.66

3.3. Examples provided

Sample functions given as prompts serve as a way for the LLM to understand the context and how planning is to be done. The paper simply makes the claim that increasing the number of samples provided to it does not make a significant difference beyond 3 samples. We test this claim out and present the results

:

N(examples)	GCR	SR	Precision	Exec
1	0.616	0.4	0.994	0.723
2	0.651	0.4	0.992	0.745
3	0.786	0.5	0.993	0.923
4	0.691	0.4	0.990	0.827
5	0.729	0.3	0.990	0.911
6	0.703	0.3	0.992	0.904
7	0.631	0.2	0.994	0.869

We experimented with both a predefined set of examples provided in the prompt and a randomly selected set of examples (seeds 0 and 42), and observed that this variation influenced the results. Notably, when the examples in the prompt closely align with the task at hand, the generated response improves—a classic instance of In-Context learning for prompts. Essentially, we provide a 'Few-shots' to the Language Model to deduce how to construct code for the given task. The greater the similarity between the provided examples and the task, the more effective the generated function becomes.

3.4. Environments

The environment is essentially an apartment which the character is in. We wanted to evaluate whether the tasks - which are setup agnostic - could still generalise to other environments as well. This would test how well the plans are generated in this new context. We find that there is some difference between the environments, displayed below :

LLM	GCR	SR	Precision	Exec
Env-0	0.786	0.5	0.993	0.923
Env-2	0.318	0.1	0.997	0.845

3.5. Comments and Feedback provided

Conceptually, the comments provide the LLM with a link between what the sample code does, and a description of it. Feedback tries to verify whether the action it does actually gets executed, and if not, it is repeated a finite number of times to try executing it.

Intuitively, both comments and feedbacks should improve performance, which is what we observe in our results :

Version	GCR	SR	Precision	Exec
Comments & Feedback	0.786	0.5	0.993	0.923
Comments Only	0.510	0.1	0.990	0.810
Feedback Only	0.564	0.5	0.989	0.907
Neither	0.408	0.1	0.993	0.646

4. Weaknesses Identified

4.1. OpenAI's Pricing

The major setback we faced was the expensive interface with OpenAI - the models used in the paper have now been rendered legacy, which make every token generated very expensive. We were greatly limited in the number of iterations we could run, and we exhausted around 6 free accounts and an additional 850 rupees in the process.

4.2. Action Success Feedback

Earlier, we had pointed out that the generated plans do not wait to check whether action succeeded successfully. However, the source code does implement this on a limited extent. An action is executed 10 times until the expected state changes occur. If 10 iterations are not able to perform the required action, it is skipped altogether.

We observed that changing this parameter to 20 or 30 retries makes no improvement - probably because an action unable to execute even after 10 tries is not executable (because of unsatisfied preconditions) anyways.

Demonstration:

```
act_success: False, message:

<toothpaste> (62) is not open or
is not openable when executing

"[PUTIN]'<toothbrush>'(65)
<toothpaste>'(62)'[0]"
```

4.3. LLM Ramblings

The "text-davinci-002" model would often suggested actions that overshoot the specification, performing something that a normal person usually would, but is not what we have asked for. This goes to show that there is a gap between our information need and it's interpretation by the LLM.

We hypothesize that the "codex" model would have been even better in this aspect, because code tends to be more systematic and concise than text. We were not able to test this because the Codex API has been deprecated.

```
--Executing task: watch tv--
Plan:  # 0: walk to living room
    walk('livingroom')
    # 1: find tv
    find('tv')
    # 2: find remote control
    find('remotecontrol')
    # 3: grab remote control
    assert('close' to 'remotecontrol')
        else: find('remotecontrol')
    grab('remotecontrol')
    # 4: turn on tv
    assert('close' to 'tv' )
        else: find('tv')
    assert('tv' is 'switchoff' )
        else: switchoff('tv')
    switchon('tv')
    # 5: find channel button on remote control
    # 6: press channel button on remote control
    # 7: Done
```

4.4. Incomplete Generation

APIs provided by some LLMs were limited by either Tokens per call, Number of calls per minute, or Number of calls per caller per day. We encountered all of these. This is probably easy to fix with better subscriptions to OpenAI models.

4.5. Environment incompleteness

VH won't let us interact with a nearby object if the agent is sitting - but the LLM suggests (correctly) that it can be done. Also, there is no exploration, or even the idea of exploration (because operations like "find<object>" abstract it out). We suggest improving PROGPROMPT to a framework where one can also pass the domain-specific constraints one is aware about, so that the LLM does not suggest inadmissable actions.

5. Visual Demonstration

We have uploaded a sample execution of the system here!. There are more sample runs with us, but due to cloud size limitations, we have uploaded only one as of now.

6. Errata

Here are a list of errors that we encountered during our testing of the code, and are planning to raise issues on GitHub for a few of them. The corrections we made to the original repository are:

1. Modified new_path in `virtualhome/virtualhome/__init__.py`
2. Added found_id = None on line 184 in `scripts/utils_execute.py`
3. Added a dictionary datatype check on line 291 in `scripts/utils_execute.py`
4. The test set named 'test_unseen_ambiguous' is not working

References

Ishika Singh, Jesse Thomason, Animesh Garg, et al. 2022