AutoGPT

Tom Rochette <tom.rochette@coreteks.org>

August 30, 2025 — 861fb9d0

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- Multi-agent system where each agent has a specialization/role. By doing so we can make the agents communicate with one another instead of just feeding back the same agent with its own output.
- Integration of RLHF (Reinforcement Learning with Human Feedback) to allow the user to provide feedback to the agent(s) in order to improve the quality of the output.
- Automating prompt improvement by prompting ChatGPT to improve an initial prompt.

1 Overview

In this article we explore the idea of using ChatGPT to automate the achievement of high-level goals. We will refer to this as AutoGPT.

2 Notes

- Like any project, before leaving AutoGPT to its own devices, it is important to set both a time and monetary budget. This is to prevent it from running indefinitely and to prevent it from running up a large bill.
- Like any project, as a project manager, it is important to have regular checks to ensure that the project is on track. Leaving AutoGPT unattended may result in ChatGPT being stuck in a prompt-response loop or having ChatGPT working on something completely unrelated.

3 Task planning

When submitting an initial request to achieve a goal, the first steps will be to generate a high-level plan. From this high-level plan we will need to call ChatGPT again numerous times in order to refine the steps until we reach a point at which programs can be written and executed.

Much like during project decomposition, we will need to break down the high-level plan into smaller and smaller steps. Some of those steps will have dependencies between themselves, while others will be independent.

Establishing task dependencies automatically may not be trivial. A simple but inefficient approach would be to consider the steps of a plan generated by ChatGPT to be sequentially dependent on each other, meaning that no parallelization may be possible. A more complex approach could be to ask ChatGPT to generate a dependency graph for the plan it generates. This would allow us to parallelize the tasks. The plan may however not necessarily be executable (i.e., have dependencies missing). We could ask for each task to indicate its inputs and outputs. This would allow us to generate a dependency graph and to parallelize the tasks according to their dependencies. This would also ensure that each task is essential.

4 Task prioritization

Given a task graph, we can let libraries such as dask or ray take care of running the task when their dependencies are met.

Given a limited amount of compute we may however want to prioritize what AutoGPT should be working on. Prioritization in this scenario would require us defining a metric to decide on which tasks to work first. A common approach is to estimate the amount of effort necessary to complete the task and the amount of value produced by the completion of the task (i.e., the return on investment).

5 Task distribution

Given that ChatGPT will generally produce a list of tasks to produce in order to achieve a goal, it is possible to distribute those tasks to different agents. This is similar to how a project manager would distribute tasks to different people. This is also similar to how a machine learning engineer would distribute tasks to different machine learning models.

Given that current ChatGPT models take between 10 and 20 seconds to respond, high task parallelism will make better use of our rate limit. Assuming we are using the gpt-3.5-turbo model which has a response time of 10s per request (for 512 output tokens and a temperature of 0.7) and we have 3500 RPM, a single agent will run 6 queries per minute. We would be able to run around 580 agents in parallel.

Assuming AutoGPT is implemented using Python, we could make use of dask or ray to distribute tasks to different agents.

Note that most of the client's work is simply to submit requests to ChatGPT, wait for an answer, do some minimal processing, and submit the next request. As we get into the low level tasks (i.e., executing programs generated by ChatGPT), more time will be spent on by the client to execute those programs. As such, we may be spending less time sending requests to ChatGPT and more time actually doing "work".

Depending on the type of work involved, the amount of compute necessary will vary.

6 Execution environment

A local environment could be implemented using docker or microk8s (or similar) as its foundation. The host computer would be responsible starting and stopping docker containers according to the needs of the project. A production environment could be implemented using Kubernetes.

In both situations, a client is responsible for submitting a goal, which will result in the AutoGPT cluster to start working on the goal.

Each running container contains a copy of the code necessary to run a worker, which is mostly going to be responsible for calling a python script with a set of arguments. The result of the script execution will be sent back to the client for further processing.

7 Self-reflection/Continual improvement

At regular interval, inject prompts asking AutoGPT to reflect on its own progress. This will allow us to detect if AutoGPT is stuck in a prompt-response loop or if it is working on something completely unrelated.

Self-reflection should also be used to evaluate whether the generated plan is still relevant and the most efficient way to achieve the goal. If it isn't, then we should revise the plan and adapt.

8 Unsorted

- Generate structure within the content it is producing (i.e., create databases, tables, schemas, datasets, etc.)
- Task deduplication

8.1 See also

- AnthropomorphGPT

8.2 References

- https://en.wikipedia.org/wiki/OODA_loop
- https://en.wikipedia.org/wiki/PDCA
- https://en.wikipedia.org/wiki/Continual_improvement_process
- https://github.com/Significant-Gravitas/Auto-GPT
- https://github.com/reworkd/AgentGPT
- https://github.com/yoheinakajima/babyagi