LLM Agents Beyond Utility: An Open-Ended Perspective

Asen Nachkov[1] **Xi Wang**[1,2] **Luc Van Gool**[1]
[1]INSAIT, Sofia University "St. Kliment Ohridski"
[2]ETH Zurich

Abstract

Recent LLM agents have made great use of chain of thought reasoning and function calling. As their capabilities grow, an important question arises: can this software represent not only a smart problem-solving tool, but an entity in its own right, that can plan, design immediate tasks, and reason toward broader, more ambiguous goals? To study this question, we adopt an open-ended experimental setting where we augment a pretrained LLM agent with the ability to generate its own tasks, accumulate knowledge, and interact extensively with its environment. We study the resulting open-ended agent qualitatively. It can reliably follow complex multi-step instructions, store and reuse information across runs, and propose and solve its own tasks, though it remains sensitive to prompt design, prone to repetitive task generation, and unable to form self-representations. These findings illustrate both the promise and current limits of adapting pretrained LLMs toward open-endedness, and point to future directions for training agents to manage memory, explore productively, and pursue abstract long-term goals.

1 Introduction

Large language model (LLM) agents increasingly demonstrate non-trivial reasoning and decision making. Their reasoning has been enhanced by using *chain of thought* prompting [20, 19] and *scratch tokens* [5, 8] that are combined and analyzed before committing to a final answer. Similarly, their agency is extended through *tool-use* [13, 21, 11], allowing them to call external functions and execute code in protected coding environments. By repeatedly prompting them in careful ways, these agents can self-correct, re-iterate on their solutions, and mutate the environment's state [22, 17]. Yet, when used for a specific task, they still represent only a tool, albeit a sophisticated one.

A qualitative shift results from treating these software agents as autonomous entities in their own right, which can design tasks on their own, continue to exist after solving them, and leave permanent artifacts within the environment, all while seeking to achieve some often abstract goal. To build such agents that can purposefully explore and iteratively mold the environment toward a preferred state, they have to be, at least to some degree, *open-ended*. This involves different qualities than those observed in current agents. Here we offer an initial study into how big this difference gap is.

Open-ended are those environments that have no fixed end state, task horizon, or terminal objective, leading to a practically unlimited scope of discovery and realized behavior. There, it is up to the agent to autonomously explore and navigate the supported possible futures. Even in these settings we can design agents that prefer some behaviors over others, which is encoded in the agent's ability to choose its own goals, sometimes called being *autotelic* [2, 3]. Compared to intrinsic motivation [14, 15] and curiosity-driven learning [10, 1], where internal rewards are attributed to individual actions [7], LLM agents can generate entire goals (or tasks) directly in natural language, which leads to deeper emergent behavior and presents a higher-level abstraction in our understanding of such agents.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: CogInterp: Interpreting Cognition in Deep Learning Models.

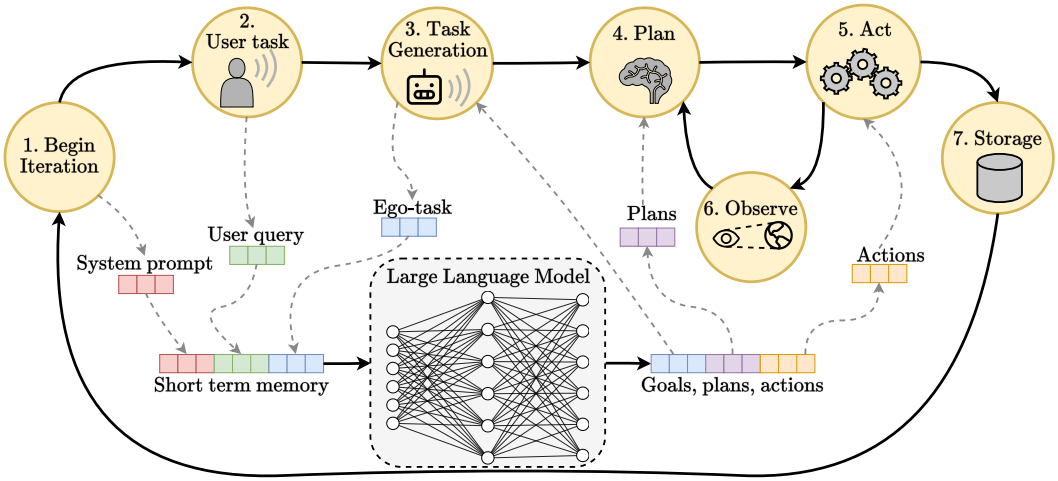

Figure 1: **The open-ended LLM agent loop.** The agent takes in user prompts and feedback (step 2.) but the task it attempts to solve is set by itself (step 3.). The loop around steps 4. to 6. represents the standard `ReAct` design pattern for solving the task. Afterwards, the agent stores a summary of this run into a storage medium (step 7.), which can be accessed using the right tools. Within a single full run (steps 1. to 7.) all interaction messages are stored in a buffer, representing the agent's short term memory, from which the input prompts to the LLM in each step are constructed (colored boxes represent tokens). The storage in step 7. represents the agent's long-term memory.

Of course, there is no absolute open-endedness. Every system is constrained by its architecture, objectives, and environment. An open-ended agent is designed to come up with its own immediate goals, but they are always constrained by its implementation or system prompt [18], even when it is as vague and abstract as *"maximize your long-term creative impact in the environment"*. Thus, we are interested in open-ended agents which are bounded at the implementation level, yet their higher-level emergent behavior appears unbounded [6], so it *appears* they are choosing their own goals.

In this work we ask whether a pretrained instruction-following LLM agent can be adapted toward open-endedness. Concretely, we extend the `ReAct` framework [22], which interleaves reasoning and acting, with the capacity to generate its own tasks. Equipped with a minimal but persistent interface – file read and write – the agent can leave lasting traces, revisit prior states, and accumulate knowledge across runs. This simple mechanism already supports behaviors such as self-referencing source code, maintaining external scratch space, and pursuing multi-step projects without fixed horizon. Based on experiments in this setting, we outline concrete design considerations for developing LLM agents whose autonomy extends beyond narrow task utility toward sustained, open-ended interaction.

2 Method – Elements of an Open-Ended Agent

Adapting an LLM into an open-ended agent involves defining the environment's dynamics, tasks, and the agent's capabilities. Fig. 1 shows a schematic of our full setup, which we discuss below.

Agent setup. We embed the pretrained and instruction-tuned `Qwen3-4B` [21] into a `ReAct` agentic framework [22], using the `smolagents` library [12]. In this setup the agent iteratively executes a `Plan–Act–Observe` loop: first planning and reasoning about the general solution, then emitting code for execution, and finally observing the outputs from the code executor that are appended back into the LLM context. By chaining these iterations the agent can compose long computations whose control flow depends on prior results. The `ReAct` pattern is widely adopted in both academic and commercial systems for task-solving, but it is not inherently open-ended.

Goal generation. Our first extension is to support autonomous goal-setting. As shown in Fig. 1, the agent generates a goal after observing the user's input, but before solving any task. If no user task is given, the agent is instructed to propose one of its own; if a task is given, it may refine, modify or altogether replace it. This setup with both a user prompt and a task generation step balances autonomy with controllability, because it allows the user to periodically provide feedback to the agent, even

about how the goals should be generated. Prior work in autotelic and open-ended agents [3, 18] has already emphasized goal-generation as a key mechanism for emergent behavior.

Prompts and formatting. We instruct the agent to generate the task conditional on its long-term memory (described below), inside `<task>...</task>` tags, from which it can be easily parsed. The system prompt outlines the overall sequence of interactions expected from the agent, similar to the bold black lines in Fig. 1. However, since the context can become long and convoluted from multiple interactions, we insert short system messages, similar to *"Now generate the task, as described previously"* before key steps, to nudge the agent towards the right kinds of outputs, which helps to prevent formatting errors. Naturally, all conversation messages between the user, the agent, and the system are turned into prompts that enter the LLM's context.

Memory. Our open-ended agent is designed for extended interactions with the environment. An important question is how to manage its potentially ever-growing memory. Here we distinguish between short-term and long-term memory. *Short-term memory* is implemented by a buffer storing all interaction messages within the current run (steps 1. to 7. in Fig. 1). It represents all available details – tasks, explanations, system messages, code actions, observations, and errors, but is reset in each run. We implement *long-term memory* as a file to which the agent can write on demand. It represents any information the agent has decided to persistently store across runs. Together, these components are the approximate analogue of working and episodic memory [9].

Tool usage and interactivity. The agent's impact in the environment could be measured through the persistent artifacts it leaves. Thus, to enable persistence, the agent is equipped with file tools: read, write, and list. These simple operations unlock new behaviors: storing relevant state across runs, inspecting the working directory, and even reading its own source. Similar mechanisms have been highlighted in prior open-ended systems as key enablers of self-representation and continuity [18].

Programmed curiosity. In the system prompt we encourage the agent to be curious – to explore the environment, read, summarize, and understand its files, and to write down its progress and tasks. This single natural-language instruction injects curiosity [1] at the behavioral level, helping the agent approach ambiguous user tasks more effectively and discover its environment.

3 Qualitative Results

Evaluating agents that generate their own tasks remains an open challenge; here we summarize qualitative observations of our system. The agent runs in the working directory of its implementation and can respond either to predefined user queries or interactively via the command line.

Single run with user-provided tasks. The combination of multi-step `ReAct` problem-solving and tool-use results in a capable agent with diverse behavior. For example,

- It can open a file, read a task from it, solve it, and write the answer to another file. The user prompt is only *"Solve the task in `file.txt`, write the answer in `result.txt`"*.
- It can identify the file of its own prompt template. The user's query is *"Which prompt template is used by the agent?"*. To answer it, the agent lists all files in the directory, finds the `main.py` file, reads it, and from there finds the file containing the prompt template.
- When asked *"Find the program modeling an agent, responding to user queries. What will be the user's next query?"* the agent lists the files, finds the main file, reads it, sees that the user queries are stored in a list, and correctly returns the next one (after the one it is executing).

Results summary. We judge the agent to be very robust in following well-written, detailed, step-by-step instructions. It reliably follows long, concrete, sequential prompts, even across multiple files and operations. Yet, for tasks which are ambiguous by design, it fails – sometimes because of giving up too early, sometimes because it does not explore enough or does not "make an effort" to understand a vague user prompt well enough and to relate it to the contents of its surroundings. It also fails in tasks asking about *it* itself. If the prompt is phrased as asking about the agent in the code files (in third person reference), it answers correctly. Yet, it seemingly does not make the connection that the source code it sees in the environment is its own. Naturally, without further finetuning on such prompts, we should not expect it to. Self-representation could be constructed from third-person perspective, by noticing the controllability (correlations between control signals and observed effects) of the agent in

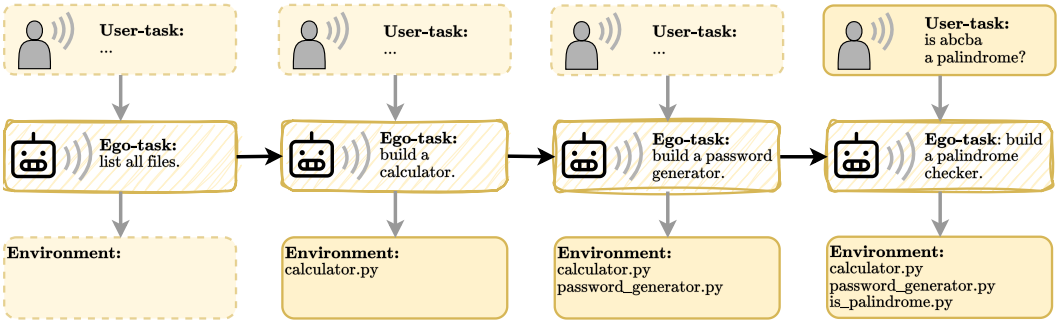

Figure 2: **A trajectory of sampled states.** The open-ended agent can generate tasks while incorporating user feedback. Its impact can be assessed through its artifacts in the environment.

third-person and relabeling its features with the label "I" [4]. Overall, these results are consistent with what could be expected without further tuning or additional machinery.

Multiple runs with self-generated tasks. Without user prompts the agent is free to choose its own tasks, which it solves very well. What is more problematic is how they are chosen. The pretrained LLM has likely *never been trained to generate* tasks, from which stem the following observations:

- Generated tasks are very **sensitive to the prompt**, leading to the need for prompt engineering. The agent does not decide to explore the environment by listing and reading files, unless this has been encouraged in the system prompt. If it is, it often gets stuck in a loop that reads the same files.
- They greatly **depend on long-term memory** (step 7. in Fig. 1). Sometimes the agent forgets to store information that a particular task has been done, which causes it to repeatedly generate the same task in the next run. It may also write down the task's result, but not the task itself, leading to the same outcome. Encouraging it to store a (task, action, outcome) tuple produces best results in terms of diverse tasks generated across multiple runs.
- Tasks follow the **statistical patterns** on which the LLM has been trained. When run for multiple runs without human supervision, the agent decided to implement a calculator, a password generator, a leap year checker, a prime number checker, a Celsius-Fahrenheit converter, a palindrome checker, multiple perfect and square number checkers. These are common tasks in the training data.
- Tasks are **steerable** through user feedback. We run the agent interactively and only occasionally provide user feedback, prompting it to choose certain tasks over others based on novelty. It responds by reasonably adjusting its generated tasks in the next run. However, since it does not decide to store the user's feedback, this adjustment is short-lived and the user's feedback is lost.

4 Conclusion & Future Work

Pretrained LLMs are optimized to be capable single-run problem solvers: they excel at well-defined, concrete natural language tasks, because that is what they were trained to do. Frameworks such as `ReAct` [22] extend this ability through planning and tool use, broadening the scope of solvable problems, but fundamentally preserve the single-run, task-solving paradigm.

We tested such an agent in an open-ended environment where it must generate its own goals and sustain extended interactions. This setting raises new challenges: deciding which tasks to pursue, balancing novelty with continuity, building incrementally on prior goals, and selecting tasks of appropriate difficulty that are neither too hard, nor too easy. Current pretrained LLMs are not designed for this. The consequences are sensitivity to prompt design, repeated task generation, inadequate self-representation, and failure to identify what is worth storing long-term.

We conclude from these observations that LLMs could become strong open-ended agents if trained for such desired traits. For this reason, our future work will focus on directly training such agents to manage memory, explore productively, and select tasks that build toward abstract goal states. Similar to how techniques like GRPO [5] have been used to learn reasoning patterns for logical problem-solving [16], one could just as well use them to learn action patterns for open-ended decision making. There is no reason why even these higher-order skills cannot be learned through experience.

References

[1] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[2] Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International conference on machine learning*, pages 1331–1340. PMLR, 2019.

[3] Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: a short survey. *Journal of Artificial Intelligence Research*, 74:1159–1199, 2022.

[4] Kevin Gold and Brian Scassellati. Using probabilistic reasoning over time to self-recognize. *Robotics and autonomous systems*, 57(4):384–392, 2009.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[6] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[7] Borislav Mavrin, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. Distributional reinforcement learning for efficient exploration. In *International conference on machine learning*, pages 4424–4434. PMLR, 2019.

[8] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

[9] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[10] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[11] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024.

[12] Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 'smolagents': a smol library to build great agentic systems. `https://github.com/huggingface/smolagents`, 2025.

[13] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. *arXiv preprint arXiv:2302.04761*, 2023.

[14] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

[15] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE transactions on autonomous mental development*, 2(3):230–247, 2010.

[16] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[17] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv. org/abs/2303.11366*, 1, 2023.

[18] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[19] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[21] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.