Towards using LLMs to (co-)create agent(ic) systems

Michael Winikoff[1] [0000−0002−5545−7003]

Victoria University of Wellington michael.winikoff@vuw.ac.nz
https://michaelwinikoff.com

Abstract. This position statement argues that the time has come to consider using LLMs to help create (multi-)agent systems. It proposes research questions.

Keywords: Large Language Models (LLMs) · Software Development · Agent-Oriented Programming.

A few years ago the EMAS (Engineering Multi-Agent Systems) community began to consider the implications of LLMs. The focus was on how to *integrate* LLMs into agent systems [2]. However, since then the capabilities of LLMs have improved dramatically, and this position statement argues that **it is now time to consider a new role for LLMs: using them to help *(co-)create* (multi-)agent systems.**

LLMs[1] can generate code for popular programming languages. They can solve toy problems (e.g. towers of Hanoi) very well, but for more complex systems their effectiveness drops [1]. Somewhat surprisingly, LLMs can also generate code in some agent-oriented programming languages[2].

However, software development with LLMs is not just about generating code, but is a *partnership* between the LLM and the human developer. It is therefore important to consider what the LLM needs to be able to do to be a good partner, and what guidance can be provided to the human programmer about how to use the LLM.

To be an effective partner it is important for the LLM to be able to explain what it does, so that the human developer can understand what the LLM has done, why it has done this, how it operates, and what the LLM can and cannot do effectively. It is also important to be able to customise the LLM's operation, for example, indicating that certain parts of the code are test cases and should not be modified or deleted.

Turning to the human developer, their role changes when using an LLM as a coding partner. The human plays a crucial role in breaking down the problem, prompting the LLM, reviewing generated code, and specifying tests, as well as writing code themselves [4]. Guidance for developers therefore needs to address the overall process, as well as details on each of these aspects (e.g. when (not) to use the LLM, how to break down the problem, how to prompt). One particularly interesting question is how to specify requirements: what concepts and formats should be used?

[1] This paper is based primarily on experiences with Claude Sonnet 4, although ChatGPT and Gemini were also used.

[2] Claude Sonnet 4 indicates it can generate code in JASON, GOAL, 2APL, and JACK. However, when requested, it actually is also able to generate code in SARL, JAM and PRS. ChatGPT is able to generate JASON and SARL code (but not valid JACK code), and Gemini is able to generate JASON code, but not JACK or SARL.

We therefore pose the following key research questions:

- **How well can LLMs generate code in agent-oriented programming languages?** Are they less effective at this than at generating code in languages where there is much more code available for training? How can we assess this? Do we need to extend SWE-Bench [1] (or similar) with agent-oriented benchmarks?
- **How effective are LLMs as development partners?** How effectively can they explain things? How well can they be customised with prompting (e.g. "don't change this part of the code")?
- **How can developers best use LLMs to help with writing agent systems?** What guidance can we provide to developers about *when* to use LLMs (and when not to!), and *how* to use them? (e.g. what process to follow, what size and type of tasks to give to LLMs, how to prompt them, how to specify tasks, and what checking and testing to do with the LLM output)
- **How can requirements best be captured so that LLMs can generate (parts of) systems?** Should we be using requirements documentation, user stories [3], or something else? Since we are developing at a higher level, perhaps we should be using higher-level concepts such as values?

Additionally, the adoption of LLMs can reduce the barrier to using a new programming language, so may aid in the adoption of agent-oriented programming languages.

Finally, the emergence of "agentic AI" (sometimes "AI agent") presents a window of opportunity for the EMAS and AAMAS communities to showcase their work and its relevance to a wider community. We therefore close with a final key research question: **how can EMAS concepts (e.g. goals, plans, norms, commitments, organizations, valuess, ...) be used to engineer *reliable* and *safe* agentic software?**

Disclosure of Interests. The author has no competing interests.

References

1. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.R.: SWE-bench: Can language models resolve real-world github issues? In: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net (2024), https://openreview.net/forum?id=VTF8yNQM66
2. Ricci, A., Mariani, S., Zambonelli, F., Burattini, S., Castelfranchi, C.: The cognitive hourglass: Agent abstractions in the large models era. In: Dastani, M., Sichman, J.S., Alechina, N., Dignum, V. (eds.) Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024. pp. 2706–2711. International Foundation for Autonomous Agents and Multiagent Systems / ACM (2024). https://doi.org/10.5555/3635637.3663262
3. Rodriguez, S., Thangarajah, J., Winikoff, M.: User and system stories: An agile approach for managing requirements in AOSE. In: Dignum, F., Lomuscio, A., Endriss, U., Nowé, A. (eds.) AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021. pp. 1064–1072. ACM (2021), https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p1064.pdf
4. Ullrich, J., Koch, M., Vogelsang, A.: From requirements to code: Understanding developer practices in LLM-assisted software engineering (2025), https://arxiv.org/abs/2507.07548