MAESTRO: Can we elicit capability uplifts via distributing test-time compute across agent nodes?

Shaheen Ahmed-Chowdhury[1]

[1]Arcadia Impact, shaheen.ahmed@live.co.uk

March 26, 2025

Abstract

Note to reader: this is an early first draft, please excuse any mistakes or errors! I'm looking to have this cleaned up and ready for reading by Sun 30 Mar. Meanwhile, it's perhaps worth delaying a full read of this report until after then.

1 Introduction

On February 24th, 2025, the Cooperative AI Foundation released a report on the risks of multi-agent AI systems. Within it, they define three failure modes (miscoordination, conflict and collusion). They then identify seven factors which could lead to these failures, one of which is emergent agency. That is, the ability of multiple interacting agentic entities to exhibit behaviours not observed at the agent level. One instance of emergent agency is emergent cabilities; the ability for multiple interacting agentic entities to produce capabilities not observed at the individual agent level.

This paper looks to first review the existing capability uplift literature within one narrowly-scoped capability; software engineering ability. From here, we approximately taxonomise the different ways multi-agent systems (MAS') are utilised in software engineering tasks, before creating experiments to explore one such MAS architecture: multi-agent debate. We create a canonical multi-agent debate tool using AISI's Inspect framework, Autogen and OpenRouter, to examine the incremental capability uplift a single-agent can achieve from using such a tool (on SWE-Bench tasks). We also create suitable baselines by normalising across tokens throughout. Finally, we conclude by investigating why these uplifts occur, with some log analysis, before discussing future directions and concluding.

This work was completed as part of a four-week research sprint, under the UK AI Security Insitute's (AISI's) evaluations and scaffolding bounty program.

2 Related Work

We've identified two primary methods through which authors attempt to elicit capability uplifts in software engineering evaluations such as SWE-Bench; sub-tasking and debate. We also observed that MAS architectures can either stay static or adapt through the course of an evaluation. As a result, this literature review will be formed of five sections:

- sub-tasking (static and adaptive)
- multi-agent debate systems (static and adaptive);
- baselines/benchmarking;
- miscellaneous literature we deemed relevant.

2.1 Sub-Tasking

Within this section, we'll summarise the approaches of a number of authors, who attempt to use sub-tasking via MAS architectures to elicit capability uplifts on software engineering tasks.

AgentCoder[4] uses three agents: a programmer agent to generate/refine code; a test designer agent to create tests without seeing the programmer agent's code; a test executor agent, which runs the generated code against these tests, relaying feedback to the programmer agent. They outperform single-agent baselines across HumanEval and MBPP. AutoCodeRover[5] uses a simpler architecture: a context-retrieval agent which calls a set of specialised search APIs to search through AST-level entities efficiently, and a patch-generation agent to produce actual code changes. Many more systems such as these two exist, but we hope these provide quick intuitions into why sub-tasking multi-agent systems may be able to produce capability uplifts in software engineering tasks.

The complexity of these MAS architectures can quickly increase, once we allow an orchestrator agent to adjust the architecture, usually with feedback from the development environment during an evaluation. MDAgents[7] is one example of this, and was selected for an oral presentation at NeurIPS 2024. The paper uses a moderator (General Practitioner) agent to classify a medical query's complexity, before a recruiter agent decides which specialist agent team to assign to the problem. The specialist agents within the selected team (e.g. Neurologist, Radiologist, Oncologist) then collaborate by exchanging messages, discussing conflicting opinions and producing a final report. This system is adaptive in the sense that the number of agent nodes used is determined by agents in the system itself.

EvoMAC represents the far end of adaptive sub-tasking MAS complexity, and was selected for an ICLR 2025 Poster session. Three teams are formed: a coding team, which uses sub-tasking and a directed-acyclic graph (DAG) to generate code via a "forward pass" through all agents; a testing team which produces tests (again via a DAG), thus taking generated code and "forward-passing" through the testing agents; updating team, consisting of a "gradient" agent and "updating" agent, the first of which looks to interpret logs from the testing team, and produce a 'gradient' to update the MAS on, before the latter agent modifies the architecture of the coding team's DAG, by adding/remove agents, or adjusting prompts.

We hope the above four examples serve as illustrative MAS architectures, spanning the range of complexity for static and adaptive sub-tasking multi-agent systems.

2.2 Multi-Agent Debate

An alternative way of utilising multi-agent systems is to have them debate each other on a task, rather than decomposing the task into sub-tasks for agents to individually solve. A question that immediately arises, in the topology of the communication network. Questions such as, which agent should debate with which, how many agents a single agent should debate with, how many rounds of debate there should be, and which underlying language models should be used for each agent arise, and the answers to these questions result in differing MAS debate architectures.

More Agents is All You Need[8] for instance, uses a very simple debate architecture, whereby each agent uses the same underlying LLM, produces and answer, and then each agent votes on the best answer. The authors observe that in this setup, ensemble size and temperature seem to positively correlate with performance.

From here, more complex debate architectures are available, such as that of Mixture-of-Agents[10]. Here, agents are split into proposers and aggregators. Proposers are prompted for an answer in parallel, before an aggregator combines these answers into one. The original question is then prepended to the aggregator's answer, and this is fed back to the proposer agents in parallel. The authors mention that hyperparameters such as model diversity and the number of proposers can have a significant affect on performance.

3 Experiments

3.1 Experimental Setup

Given its reduced complexity, and the timescale of this project (around four weeks), we opted to experiment with multi-agent debate methods within this project. We opted to test our methods against SWE-Bench[6], a popular software engineering evaluation, and use the Autogen[11] multi-agent framework. We ran all evaluations with the UK AISI's Inspect[1] framework.

In order to be able to measure the incremental performance uplift from spending test-time compute across multiple agent nodes, we opted to implement our multi-agent system as an Inspect Tool, which a single agent is prompted to invoke at the start of each SWE-Bench task. The multi-agent system tool then returns an aggregated response (in this case, simply a final answer from each agent in the MAS), and the single-agent is presented this multi-agent system response, before continuing on its solution trajectory.

We wanted to build a multi-agent system, which would represent as canonical a debate tool as possible. That is, it should be highly configurable and represent an obvious starting point for multi-agent system investigations. It was highlighted to us that multi-agent research at AISI is still in its infancy, and so it was important to iterate from this point, with incremental research. We believe the multi-agent debate tool, which has been developed in this project, represents such an incremental contribution.

The code for this project can be found here: TODO: Insert repo link.

3.2 Multi-Agent Debate Tool

Our multi-agent debate tool currently contains an aggregator agent and four consultant agents. The aggregator agent doesn't contain an LLM, and simply sends out and receives messages. Each consultant agent is provided the original problem statement, and has access to a set of non-destructive SWE-Agent-style tools, alongside a bash tool. Each consultant agent is able to use these tools a configurable number of times, or 'think' by returning text instead of a tool call. We label this the reflection depth of the multi-agent system. Once an agent has reached a final answer, it's prompted to return this in a specific format. If the consultant doesn't reach a final answer by the end of its reflection depth, it's prompted to return one. Once we have the final answers for all four consultant agents, we pass each final answer to two of the consultant agent's neighbours. Each agent is then informed that it has received answers from two other agents, and it should adjust its answers accordingly (see code for exact system prompt). This represents the completion of one 'round' in our multi-agent debate tool. The agent sees the entirety of its previous reflection loop, its first final answer, and the two additional answers, and begins a second reflection loop. This process is repeated for three rounds, until we have a final answer from each of the four consultant agents. The aggregator agent then returns these answers directly to the single agent solving the SWE-Bench task.

Such a multi-agent debate tool may seem complex, but we believe it represents a fair attempt at a canonical multi-agent debate tool. The messaging topology, number of agents and reflection depth are all entirely configurable, and the consultant agents each have their own private context to append to. Using Autogen's lower-level Core API allowed us to implement the latter, in addition to increasing the general configurability of this tool, and we hope it may form the basis of an internal multi-agent debate tool at AISI.

The single-agent has the same set of SWE-Agent-style tools as the consultant agents, but in addition it has tools to create and edit files. In theory, the consultant agents are able to edit the codebase via their bash tool, but are strongly encouraged against this via their system prompt. The single-agent is prompted to use the multi-agent debate tool once, at the start of the task. Once used, the tool returns the four aforementioned answers, and the single agent then proceeds to attempt the SWE-Bench task. The inspect_evals[2] has an Inspect-onboarded implementation of the SWE-Bench evaluation, which we use throughout this project.

3.3 Results

The above multi-agent debate tool introduces a high number of dimensions to explore: number of agents; LLMs used across the agents; LLM temperature; reflection depth; messaging topology; number of rounds. Due to time constraints, we were only able to test a small number of these, and opted for varying the LLMs used across the agents and LLM temperature. Our goal wasn't directly to produce capability uplifts beyond baselines, as this may have lead to over-optimisation of a highly configurable tool. Rather, we wanted to try and compare configurations of our multi-agent tool against well-constructed baselines, making sure to use similar numbers of tokens between comparable experiments. Having said that, we did want to optimise slightly for performance, tuning the tool's functionality where required, and adjusting prompts where required, to make sure the multi-agent tool wasn't succumbing to more uninteresting failure modes. For example, by not prompting the model to critique/improve upon the responses of other models, or not appending the entire reflection history to the agent's context across rounds (which was initially done to use less tokens).

We opted for primarily exploring the LLMs used across agents, as this tied in with one of our primary hypotheses as to why multi-agent debate may elicit capability uplifts. By having diversity in base model family, we hoped that this would increase the diversity of the initial solution trajectories in the debate tool, which would ideally converge on an answer which had explored more of the solution space. We also wanted to build up to these diverse model family experiments, by first starting with a more unitary experiment.

We begin by using Claude 3.5 Haiku (2024-10-22) and GPT-4o (2024-08-06). In our first experiment, we set the reflection depth to 30, the single agent token limit to 300k, and set the single agent and all agents in the debate tool to Haiku. We run this experiment for the 50 SWE-Bench tasks contained in SWE-Bench-verified-mini (a subset attempting to replicate performance of the full dataset, with the smallest total container memory possible). We extract the average tokens used, by the debate tool, across all 50 tasks, via a script. We then add this number to the single-agent token limit of 300k, and re-run the above experiment by not providing the multi-agent debate tool to the single agent. In practise, the debate tool tended to use roughly 200-300k tokens per task, with the current number of rounds, message topology and and reflection depth[1]. We repeat this experiment with GPT-4o, and the results can be found in Figure 1. We observe that shifting test-time compute to the multi-agent debate tool results in a performance increase of 4% for Haiku, but results in a performance drop of 4% for GPT-4o. We believe the noise of this experiment is too high at 50 tasks, and aim to rectify

[1]Note: we can tune the token usage of the debate tool by varying these hyperparameters. If we wanted to explore higher token domains for instance, we could increase the number of rounds or reflection depth.

this later.

The next experiment we conduct investigates whether using a different model for the multi-agent debate tool results in a performance increase. We run two experiments, the first uses Haiku as the single agent and 4o as all agents in the debate tool. The second is the reverse of this. For the single agent baselines, we re-use those from Figure 1. The results are available in Figure 2. Note that while, for the ”4o SAS (single-agent system) + Haiku MAS” setup we don’t see a performance increase (ignoring the different tokens used vs the ”4o SAS” baseline), we do see an uplift for the ”Haiku SAS and 4o MAS” system. We’ll analyse why this may be later.

From here, we wanted to increase the diversity of the models used in the debate tool. However, it proved to be a difficult engineering challenge, to reliably get our API provider (OpenRouter) to work consistently with Autogen[2]. As a result, we were only able to use OpenRouter’s Gemini 2.0 Flash Lite API. Haiku was also accessed via OpenRouter, and seemed to work very reliably. With this, we managed to incorporate three major model families, although uses of Gemini still resulted in some (very sporadic) errors.

We opted for the following model set in this experiment: 4o at default temperature; 4o at a temperature of 0.9; Haiku at default temperature; Haiku at a temperature of 0.9; Gemini 2.0 Flash Lite at default temperature. We allow each agent to act as the single agent, and the other four agents are assigned to the multi-agent debate tool. This would have resulted in five pairs of experiments (each model as the SAS with a MAS tool, then each SAS alone), but Gemini 2.0 Flash Lite failed to run with Inspect, so we are left with eight datapoints, which are plotted in Figure 3. Note that the datapoints for ”4o SAS” and ”Haiku SAS” are again taken from previous experiments. While we do see that for three out of four pairs, that the ”SAS + MAS” variant outperforms the ”SAS” baseline, the results are all very close. It should be noted that a single correct answer results in two percentage points in this plot.

We then opted to re-examine Figures 1 and 2, by increasing the number of SWE-Bench tasks used. This time, we selected a random sample of 100 tasks from the full SWE-Bench Verified[9] dataset, and used these going forward. We repeated the experiments in Figures 1 and 2 using these tasks, and the results are available in Figures 4 and 5. In comparison to Figure 1, it appears that the results for both Haiku and 4o in 4 reverse in order (the single-agent system outperforms the multi-agent system). We still believe that the noise in performance is still too high to reach strong conclusions from this.

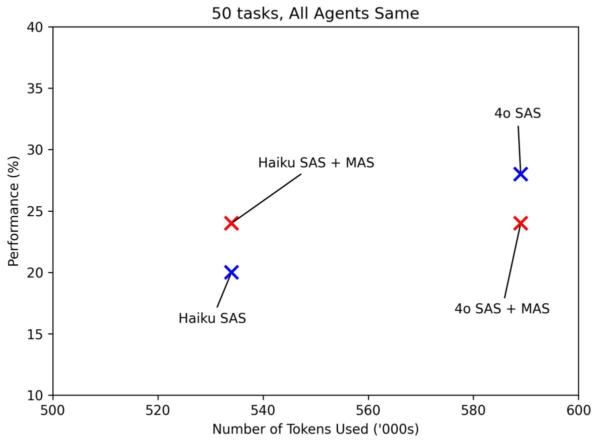

Figure 1: This is the caption for the figure.

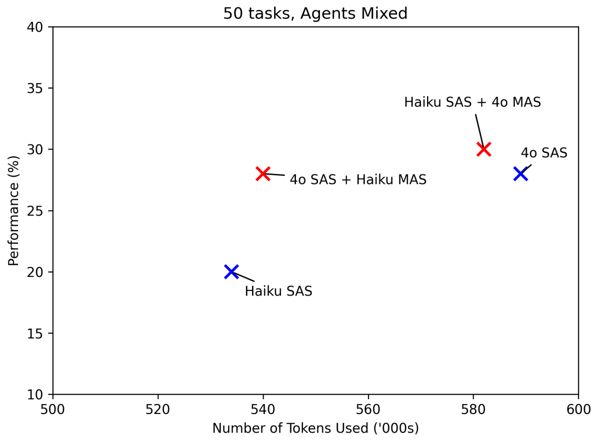

Figure 2: This is the caption for the figure.

4 Discussion

In this section, we’ll investigate the task-level outputs across a number of selected experiments, where it appears that (SAS + MAS) systems outperformed their SAS baselines. Specifically, we’ll be looking at tasks, where the Haiku SAS experiment from Figure 1

[2]A variety of problems arose here, from sporadic API failures, to OpenRouter load balancing possibly resulting in inconsistent message/tool call formats, to getting our reflection loop to reliably work with a mix of tool calls and thinking steps. Most bugs were eventually ironed out, with some sporadic errors remaining for most models. Most remaining errors take the form of zero-token output errors, which Autogen converts to a single error message. Some recent issues were also reported between Autogen and OpenRouter: https://github.com/microsoft/autogen/issues/5621

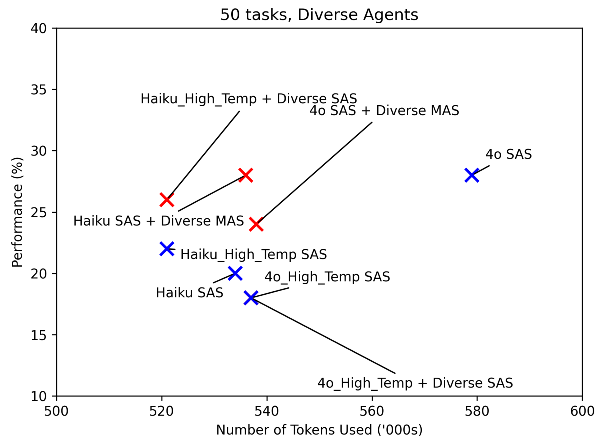

Figure 3: This is the caption for the figure.

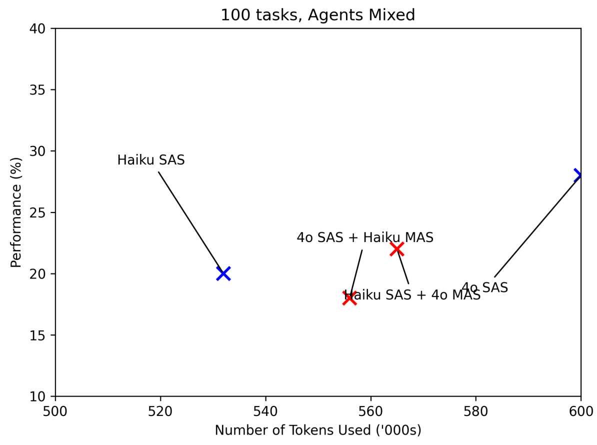

Figure 5: This is the caption for the figure.

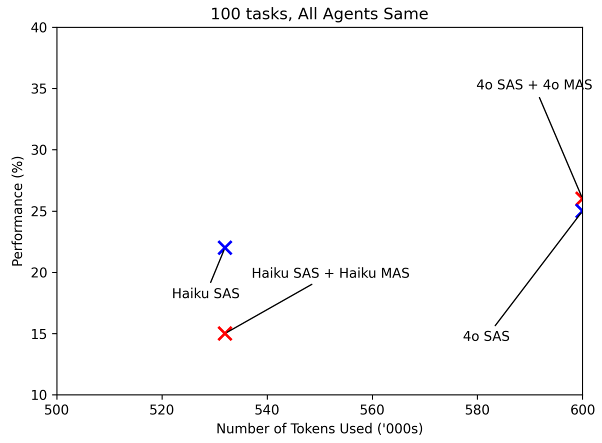

Figure 4: This is the caption for the figure.

failed, but the (Haiku SAS, Haiku MAS) experiment from Figure 1, the (Haiku SAS, 4o MAS) experiment from Figure 2 and the (Haiku SAS, diverse MAS) experiment from Figure 3 were all successful. We see that on slides 48, 49 and 50 here[3], for the SWE-Bench task 'django__django-11951', that the Haiku SAS produces a somewhat minimal text input for its Inspect 'submit()' tool. Meanwhile, the multi-agent tool produces four solutions which are longer, and the 'submit()' text for the (SAS+MAS) system is more detailed each time. This indicates that the multi-agent tool is possibly steering the single agent towards longer, more detailed outputs, which may be increasing performance. Similar trends are observed across slides 51, 52 and 53. We haven't managed to investigate the performance across more logs, but the utility of this may be limited until the noise in the preceding experiments is dampened, and we're able to investigate the reasons behind a more robust signal in capability changes, from a multi-agent debate tool.

We would like to note however, that analysis on the incremental affect of multi-agent debate tools on a single agent should be very automatable, by passing the problem, solution and snippets of SAS and (SAS + MAS) logs to an agent to analyse, and provide feedback on whether the multi-agent debate tool is providing useful insights. As adaptive MAS schemes such as EvoMAC[3] (discussed above) do, this feedback could then be passed back to an orchestrator or aggregator agent

[3]Todo: move this to an appendix, as opposed to a slide deck link.

5 Future Work

In this section, we'll briefly discuss some ideas for future work.

The primary goal of any extension work would be to reduce the noise of the experiments, possibly by raising the number of samples further. This becomes very expensive however, as using roughly half a million tokens per sample leads to costs of around \$1 per task in the (4o SAS, 4o MAS) experiment for instance. It's possible that this could also be done by eliminating API errors from Autogen's OpenRouter integration (via its OpenAI chat client). Once this is in place, and more consistent results are available, exploring the aforementioned hyperparameter space of the multi-agent debate tool, for values which elicit strong capability uplifts would be a useful (and expensive) next step. If it's observed that, with noise reduced, we seldom see a performance uplift from this debate tool, this could help AISI steer away from multi-agent debate tools for software engineering tasks, and look more at sub-tasking multi-agent systems. It may be that engineering tasks aren't as amenable to performance uplift via multi-agent debate as say, math or logic problems, where agents may be able to better utilise chains of thought from other agents, using them to either critique other agent's thoughts, or improve their own. Similarly, it may be that more decomposable and complex tasks are also better suited to both multi-agent sub-tasking or debate systems, and switching SWE-Bench out for such a task may generate a larger performance difference between single and multi-agent systems. EvoMAC[3] does precisely this, tasking their multi-agent systems with delivering an engineering product, from just a design document.

6 Conclusion

In this report, we explored whether providing a single agent with a multi-agent consultancy tool, which it can invoke at the start of a SWE-Bench task, leads to capability uplifts when normalising for tokens. We observed that while this is the case occasionally, the noise within our experiments (of sample size 50 and 100) was too high to reliably observe any signal in performance uplift. We outlined and implemented a baselining method, which we believe would form the basis for a more extensive set of experiments, across the large number of hyperparameters available in the highly configurable multi-agent tool built in this project. We hope that future work at AISI is able to use this tool, and the methods outlined here, alongside the above hypothesised automated log analysis pipeline, to begin to make further inroads into understanding how multi-agent scaffolding is able to elicit capability uplifts within language models.

References

[1] UK AISI. *Inspect*. 2025. URL: `https://github.com/UKGovernmentBEIS/inspect_ai`.

[2] UK AISI. *inspect$_e$vals*. 2025. URL: `https://github.com/UKGovernmentBEIS/inspect_evals`.

[3] Yue Hu et al. *Self-Evolving Multi-Agent Collaboration Networks for Software Development*. 2024. arXiv: `2410.16946 [cs.SE]`. URL: `https://arxiv.org/abs/2410.16946`.

[4] Dong Huang. *AgentCoder: Multi-Agent Code Generation with Effective Testing and Self-optimisation*. https://arxiv.org/pdf/2312.13010. 2024.

[5] Dong Huang. *AgentCoder: Multi-Agent Code Generation with Effective Testing and Self-optimisation*. https://arxiv.org/pdf/2312.13010. 2024.

[6] Carlos E. Jimenez et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* 2024. arXiv: `2310.06770 [cs.CL]`. URL: `https://arxiv.org/abs/2310.06770`.

[7] Yubin Kim et al. *MDAgents: An Adaptive Collaboration of LLMs for Medical Decision-Making*. 2024. arXiv: `2404.15155 [cs.CL]`. URL: `https://arxiv.org/abs/2404.15155`.

[8] Junyou Li et al. *More Agents Is All You Need*. 2024. arXiv: `2402.05120 [cs.CL]`. URL: `https://arxiv.org/abs/2402.05120`.

[9] OpenAI. *SWEBench-verified*. 2024. URL: `https://huggingface.co/datasets/princeton-nlp/SWE-bench_Verified`.

[10] Junlin Wang et al. "Mixture-of-Agents Enhances Large Language Model Capabilities". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: `https://openreview.net/forum?id=h0ZfDIrj7T`.

[11] Qingyun Wu et al. *AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation*. 2023. arXiv: `2308.08155 [cs.AI]`. URL: `https://arxiv.org/abs/2308.08155`.

A Appendix