MEASURING AI AGENT AUTONOMY: TOWARDS A SCALABLE APPROACH WITH CODE INSPECTION

Peter Cihon *
GitHub
San Francisco, CA, USA
petercihon@gmail.com

Merlin Stein *
University of Oxford
Oxford, UK
merlin.stein@bsg.ox.ac.uk

Gagan Bansal
Microsoft
Redmond, WA, USA

Sam Manning
Centre for the Governance of AI
Oxford, UK

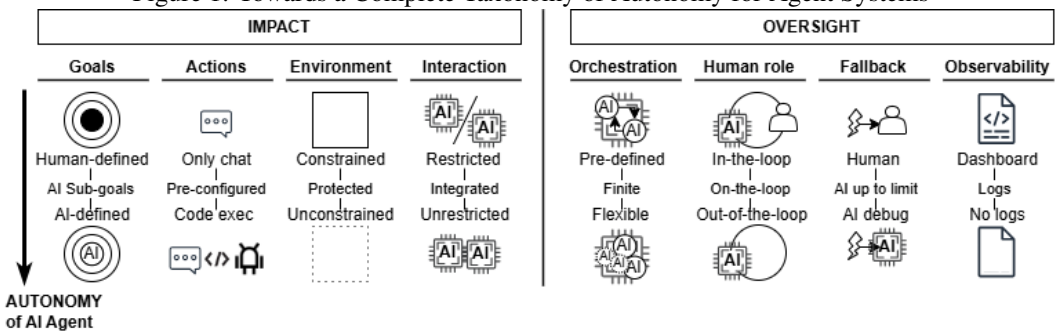
Kevin Xu
GitHub
San Francisco, CA, USA

*Equal contribution

ABSTRACT

AI agents are AI systems that can achieve complex goals autonomously. Assessing the level of agent autonomy is crucial for understanding both their potential benefits and risks. Current assessments of autonomy often focus on specific risks and rely on run-time evaluations – observations of agent actions during operation. We introduce a code-based assessment of autonomy that eliminates the need to run an AI agent to perform specific tasks, thereby reducing the costs and risks associated with run-time evaluations. Using this code-based framework, the orchestration code used to run an AI agent can be scored according to a taxonomy that assesses attributes of autonomy: impact and oversight. We demonstrate this approach with the AutoGen framework and select applications.

Figure 1: Towards a Complete Taxonomy of Autonomy for Agent Systems



1 INTRODUCTION

Language model research and product attention focuses on creating Artificial Intelligence systems capable of flexibly planning and acting to influence environments over time ('AI agents') (Wang et al., 2024; Kapoor et al., 2024). In many cases, these systems orchestrate language models and their outputs to perform complex chains of thought and action for tasks ranging from software development to vacation planning. The responsible development and deployment of AI agents present many open questions today (Shavit et al., 2023; Chan et al., 2024; Gabriel et al., 2024; Wu et al., 2023).

AI agents are designed to function autonomously. Autonomous systems can present risks of harm that have drawn policy scrutiny globally (Cihon, 2024; Chan et al., 2023). Early evaluations of AI

Published arXiv:2502.15212v1 21 Feb 2025 and in NeurIPS Socially Responsible Language Modelling Research (SoLaR) Workshop 2024.

agent capabilities have focused on general capability benchmarks (Liu et al., 2023; Mialon et al., 2023; Jimenez et al., 2024) and measuring specific threat models from frontier-capable models integrated into agent scaffolding (METR, 2024; UK AI Safety Institute, 2024).

Measuring the level of autonomy of an AI agent is helpful for a number of reasons.

- **Risk Assessment**: Measuring agent autonomy helps identify the range and consequence of decisions the agent might make, which is essential for assessing risks it might pose to those that interact with it.
- **Operational Control and Safe Deployment Strategies**: Understanding agent autonomy helps users and organizations set appropriate safety and control mechanisms, including escalation paths or fallback options. This understanding may influence deployment strategies such as phased roll outs, testing in restricted environments, or setting guardrails on certain AI agent behaviors prior to scaling up deployment.
- **Accountability, Liability, and Governance**: By measuring autonomy levels, organizations can more effectively allocate accountability and clarify liability across different actors, including users and AI agent developers.

This paper develops an initial method that could potentially be used to measure the autonomy of AI agents at scale. Complementing capability evaluations on running systems, our approach uses code inspection to assess how developers have structured the agent to function autonomously. Taking inspiration from 'autonomy levels' in other fields (SAE, 2021; Yang et al., 2017), our work aims to support research in developing empirically grounded measures of agent autonomy to inform policy and governance.

Code-based evaluations offer several advantages over run-time evaluations but also come with certain limitations (see Table 1). For policymakers, AI system developers, and evaluators, these evaluations can provide a resource-efficient, potentially scalable way to assess AI autonomy without the risks associated with run-time evaluations. This approach enables consistent, scaled evaluations that can help guide decisions, from establishing system security requirements to setting standards for safe AI deployment. Additionally, code inspection supports monitoring the development of autonomous AI agents in open source ecosystems and provides a framework for assessing proprietary applications when code access is available.

Table 1: Code-Based Evaluations in Comparison to Run-Time Evaluations for Assessing Autonomy

Benefits of Code-Based Evaluations	Limitations of Code-Based Evaluations
Reduced Risk of Harm During Evaluation: Allows evaluation of autonomy features without running the AI agent, reducing potential for unintended, harmful behaviors during testing.	**Limited Insight into Emergent and Context-Specific Behaviors:** Cannot capture unexpected or adaptive behaviors that only emerge in real-world or complex environments, which may sometimes be observable with run-time evaluations.
Lower Evaluation Costs: Avoids the need for resource-intensive run-time environments, leading to faster and more cost-effective autonomy assessments.	**Challenges in Understanding User Interaction:** Misses real-time adjustments agents may make in response to user inputs, which can reveal nuances in autonomy that aren't evident from code alone.
Early Detection of Risk Factors: Enables identification of autonomy risks early in development, allowing preemptive adjustments before deployment.	**Potential for Overlooking Latent Dependencies and Resource Usage:** Dependencies on external data sources or network availability can affect an AI agent autonomy under real operational conditions – differences that may be more apparent at run-time.
Enables Scalable and Broad Evaluation: Facilitates quick assessment across large numbers of agents or versions, enhancing comparability and consistency.	**Absence of Real-World Feedback Loops:** Code inspections cannot capture how agent actions may affect or alter the environment over time, creating feedback loops that might shift agent autonomy.

Functioning agent systems make use of language models and 'orchestration' or 'scaffolding' software frameworks that routes model inputs and outputs (Davidson et al., 2023). We focus on these software frameworks, seeking to answer (1) how frameworks, like AutoGen and LangChain, are

designed for autonomy and (2) how do downstream developers implement these frameworks into applications for autonomy? As an initial proof of concept, we analyze AutoGen (Wu et al., 2023) and some of its downstream applications. In subsequent work, this analysis could be scaled to possibly cover the wider universe of open source AI agent frameworks and applications.

Figure 1 summarizes our resulting conceptual framework of agent autonomy. This work makes three contributions: (1) a review of how system autonomy levels are operationalized in policy and AI (Section 2), (2) a taxonomy to assess autonomy levels of AI agents, based on AutoGen (Section 3), and (3) a pilot code inspection rating the autonomy of select AI Agent applications (Section 4). Section 5 concludes with implications, limitations and future work.

2 Approaches for assessing autonomy of systems

Autonomy has been defined and operationalized differently across contexts. Autonomy can include conceptions of self-actualization (constitutive) and capacity for interactions (behavioral) (Froese et al., 2007). We focus on a form of the latter: *decision autonomy*, which refers to the ability of an AI system to implement decisions without human oversight (Walsh et al., 2021) [1] It is this concept that centers the impact of actions, environments and human oversight – which are central for understanding and addressing risks from deployments.

Regulatory initiatives worldwide have started to define autonomy with a focus on *decision autonomy*. The EU AI Act considers AI systems with varying degrees of autonomy, meaning that "they have some degree of independence of actions from human involvement and of capabilities to operate without human intervention" (EU, 2024). Similarly, NIST (2005) guidance defines a system as fully autonomous if it is "expected to accomplish its mission, within a defined scope, without human intervention" and autonomy as levels characterized "by factors including mission complexity and environmental difficulty." This guidance may be applicable across domains including manufacturing and national security (Huang, 2007).

In other industries, autonomy of systems is assessed along levels. For instance, autonomous driving levels are distinguished by the boundedness of the environment (*operational design domain*), the responsibility for advanced actions (*dynamic driving tasks*) and the degree of human oversight (*fallback*) (SAE, 2021). Similarly, degrees of autonomy of medical robots or aviation depend on the environment, difficulty of tasks and human oversight (Yang et al., 2017; Anderson et al., 2018).

For AI systems, researchers have operationalized levels of autonomy with empirical evaluations on tasks requiring increasing autonomy (benchmarked to humans taking minutes, hours, days or months to complete it) (Kinniment et al., 2023; Morris et al., 2023), according to levels of human oversight (in-, on-, and off-the-loop) (Simmler & Frischknecht, 2021), autonomy of AI agents to determine suitable outputs and actions to take in the environment (LangChain, 2023; Li et al., 2024). Protocols have also been developed to evaluate an AI system's capability to pose autonomy-related risks (METR, 2024). Appendix A.1 provides an overview of levels of autonomy of different approaches.

In summary, across regulatory definitions, standards in other industries, and initial AI-specific characterisations, degrees of autonomy are differentiated based on (1) possible **impact** as a result of possible actions and environments, and (2) **oversight** in relation to orchestrating interactions and fallbacks within the system or between the system and the human. In the next section, we will operationalize an assessment of these attributes with AutoGen, an AI agent framework.

3 Assessing autonomy of AutoGen

AutoGen is a popular[2] open source software framework for building AI agent systems using language models (Wu et al., 2023). AutoGen supports multi-agent conversations and tool-use to achieve arbitrary goals with varying levels of autonomy. It includes features that involve human users in the

[1] This work is focused on informing governance of deployed AI systems. Thus, this paper builds on regulatory definitions of decision autonomy, leaving aside broader forms of autonomy or 'agenticness' (Chan et al., 2023; Ezenkwu & Starkey, 2019).

[2] At the time of writing, the AutoGen repository had over 35000 stars, 5100 forks, and over 2300 dependents on GitHub. Please see this note on governance changes in the open source project.

agent workflow, although downstream developers have ultimate control over how to configure their applications. Building on the literature above, we present a taxonomy of autonomy that assesses the specific features of the AutoGen framework for their relevance to system impact and oversight. For this preliminary work, we identify three general levels of autonomy associated with each attribute. For AutoGen, we focus on five most prominent attributes in Table 2. These attributes do not include others that are informed by the model layer and deployment monitoring, notably the granularity of goals set or pursued by the agent, inter-agent interaction, and fallbacks in deployment failure cases (See Figure 1).

Table 2: AutoGen-focused Taxonomy of Agent System Autonomy

Autonomy	**Impact**		**Oversight**		
	Actions	Environment	Orchestration	Human-in-the-loop	Observability
	What actions can the system take?	*In what environment does the system operate?*	*How are agent interactions orchestrated?*	*How are humans kept in the loop?*	*How to monitor the system?*
Lower	None (conversation only)	Constrained (limits on internet access, memory, and compute)	Agent interaction is pre-determined	Agent(s) always consult human user	Dashboards or user-focused explanations
Middle	Pre-configured tools for specific actions	Protected (Docker container deployment prevents agent from modifying its deployment environment directly)	Agent interaction is flexible but bounded (agent interactions are finite)	Agent(s) consult user for termination condition	Logs
Higher	Code execution permits arbitrary actions	Unconstrained (access to both internet and local deployment)	Agent interaction is unbounded	Agent(s) never consult user	No logs

IMPACT concerns the overall consequences of what an agent system can do. The deployment of agent systems in critical or consequential use cases influences their impact. At the framework level (as opposed to specific application-level uses), impact is shaped by the system's configuration, including the environment it can access and the range of actions it can take.

Actions are the direct capabilities an agent has to influence an environment. AutoGen enables language models to take actions using tools, which may be more or less restricted. At its most constrained, AutoGen can facilitate agent conversation without any action. Other implementations can make use of registered tools, which can enable bounded actions (such as obtaining web content using a specified search engine, for example). When frameworks enable code execution, the potential for impactful actions increases significantly. This capability allows agents to execute any software-defined action, ranging from interacting with online resources to controlling physical actuators.

Deployment environments define the operational context and constraints for agent systems. By imposing access limits, they can reduce the potential impact of an agent's actions. The AutoGen framework has two default configurations: local deployment or deployment to a Docker container. In the former, tools or executed code can alter the machine where the system is deployed, introducing computer security risks. The AutoGen framework default for code execution is the latter. By

constraining the system's environment to a Docker container, it cannot directly affect its deployment environment, even while it can access the broader internet (by default). Docker containers can be configured to limit network access, memory and compute resources.

In sum, impact is a function of the interplay between the actions an agent can take and the (constrained) environment within which it operates. For example, a Docker-deployed system without access to tools represents a low-impact implementation, whereas a locally deployed system with unrestricted code execution exemplifies a high-impact configuration.

OVERSIGHT concerns how developers and users may monitor and direct what an agent does. Design decisions at both the framework and application levels will impact how a user exercises oversight. At the framework level, these can be considered in three attributes: how agents are orchestrated, how the user is kept in the loop, and observability of the system during operation.

Orchestration of agents within a system determines the flow of their interactions. The AutoGen framework supports single agent chains of tasks and multi-agent interactions. Both can be predetermined in practice, where a developer designs a fixed flow for the agent system. Alternatively, agent interactions can be orchestrated flexibly, changing with each system run. Such flexibility is often bounded in practice: agent and multi-agent interaction (GroupChat) parameters govern an interaction, namely how it ends: for agents, how many auto-replies may be given and how many interactions total an agent may make, for GroupChat, how many rounds of interactions among all the agents. An interaction can be unbounded, where it does not have a set termination. At the extreme, AutoBuild[3] is an implementation that can permit the unbounded creation of sub-agents to complete a task.

Human-in-the-loop is an express feature of the AutoGen framework. Each agent has a parameter 'human_input_mode' which can take three values: never, terminate, and always. With 'always,' an agent will seek input from a human following every step, though it may continue after a delay insofar as the agent was initialized with an 'auto-reply'; else, the agent will be blocked awaiting human input. For 'terminate', the agent will continue until given the terminate command. With 'never', the agent will never wait for human input once initialized. 'Never' may be common in systems with a Society of Mind agent or otherwise Nested Chat: for example, an user-agent may interact with the human while running numerous sub-agents without direct human input.[4] In practice, this may complicate analyzing oversight by simply looking for the presence of a human in the loop, when a system could do so while retaining complex and possibly opaque autonomous agent interactions. Understanding the needs and competence of intended users can support a better assessment of humans-in-the-loop, although this is beyond the scope of our code inspection method.

Observability means that a user can see what an agent system is doing. The AutoGen framework supports logging, documenting each time an agent within the system is invoked, the input, output, time, cost, and associated identifiers. Logs may be of use to developers looking to debug an application they are building. However, if the logs are not used further to create explanations or dashboards, end-users may not have meaningful observability into agent actions.

See Table 2 for a summary. These attributes are not exhaustive: other aspects of agent frameworks, downstream application development, and use influence autonomy of systems in practice. Additional considerations are raised at the model layer and/or deployment time, including the specificty of goals pursued by the agent, its interactions with other agents, and means of falling-back in failure cases. These relate to an agent taking initiative (while considering costs and benefits of interruptions, user effort, and the chance of getting meaningful user input (Horvitz, 1999)). Our approach is practical rather than comprehensive: to facilitate empirical evaluation of applications being built with agent frameworks. We offer a case study assessment for AutoGen to provide an illustrative example of how the autonomy taxonomy can be applied to agent frameworks more broadly.

[3] https://microsoft.github.io/autogen/blog/2023/11/26/Agent-AutoBuild/

[4] https://microsoft.github.io/autogen/docs/reference/agentchat/contrib/society_of_mind_agent/

Table 3: Scoring Autonomy of Selected AutoGen Applications

Autonomy	Impact		Oversight		
	Actions	Environment	Orchestration	Human-in-the-loop	Observability
Lower	–	5	2, 5	5, 10	2, 4, 7, 8, 9
Middle	5, 6	4, 6	1, 3, 4, 6, 8	8	1, 3, 6, 5, 10
Higher	1, 2, 3, 4, 7, 8, 9, 10	1, 2, 3, 7, 8, 9, 10	7, 9, 10	1, 2, 3, 4, 6, 7, 9	–

Key: 1: AutoGen Studio, 2: Composio, 3: Sibyl System, 4: Dream Team, 5: GraphRag_Ollama, 6: AutoTx, 7: Letta, 8: h2oGPT, 9: Langflow, 10: GPT-Academic

4 Assessing autonomy of AutoGen applications

We operationalize the autonomy taxonomy by inspecting the source code for AutoGen applications. Appendix A.2 describes code flags used for scoring. Ten AutoGen applications[5] were scored by three researchers. Consensus results are shown in Table 3 and further detail in Appendix A.2.

The analysis shows that a differentiated categorization and scoring of agent autonomy with code inspections is tractable. The three raters demonstrated substantial inter-rater agreement (Fleiss' $k = 0.64$) across all evaluation categories, indicating consistent application of the rating criteria; additional agreement measures reported in Appendix A.2.

The variations between AutoGen applications are notable. Flexible platform applications like AutoGen Studio or Composio allow users to build specific applications with multiple degrees of autonomy. For example, this includes code executions, inside or outside pre-specified environments like Docker containers. Applications range from pre-configured machine learning agent with multiple sub-agents who build and critique code, to financial research agent and editing agent teams.

Specific applications like Dream Team for development with quality assurance or GraphRAG for data-based reasoning are more constrained for most autonomy attributes. Potentially for their reliability for a specific purpose, they operate in bounded Docker environments. However, both applications still spin up one subagent that requires a human-in-the-loop, as a fallback or main input, and the remaining – executing – subagents human-off-the-loop.

This pilot has limitations, partly due to our small sample and partly due to the nature of code inspections. Inter-rater agreement varied by attribute: orchestration ($k = 0.67$), human-in-the-loop ($k = 0.65$), environment ($k = 0.60$), observability ($k = 0.47$), and action ($k = 0.30$), with the latter three being below the substantial agreement threshold. Actions may be enabled not only by the AutoGen framework but by downstream developer choices to make custom tools or use additional frameworks, complicating review. Observability proved challenging to consistently differentiate between maintaining logs that may be useful to developers and meaningful awareness for users that may limit autonomy in practice. Consistent assessment of the constrained environment also proved challenging, where in some cases wider access may be possible through API calls to other systems. Inter-rater agreements aide, additional limitations apply to the human-in-the-loop attribute. Is an agent that requires human approval only after trying multiple different information retrieval tools more or less autonomous than an agent that requires human approval after each mouse click? It might depend on more granular assessments of the kind of actions before which human approval is required.

No agent system received lower scores on actions and observability. The first may be explained by the common interest in developing agent systems to take actions, i.e., to do more than simple chat interactions that are also possible with AutoGen. The open-source nature and focus on developers of

[5]Top 5 dependent repositories identified from AutoGen repository by stars, and 5 uniquely significant repositories like AutoGen Studio.

the inspected agent systems, might lead to a focus on transparency and consequently logs. Closed-source systems might still offer logs to developers, but not to users.

5 Conclusion and future work

This paper piloted a code-inspection approach to assessing agent systems for their level of autonomy. With further development as identified below, this approach holds promise for agent assessments at scale. Code inspection rates AI agent applications without running them, which is resource-efficient, reduces risks associated with run-time evaluations, and can enable uniform, scaled assessments that can inform stakeholder decisions – from model developers to policymakers. It also supports risk assessment to inform what level of system security might be important before running them. This method could be used to monitor the development of autonomous agent systems in the open source ecosystem and provide a framework to assess proprietary applications given code access.

Although final results will require additional scale, this preliminary exercise identifies the relevance of defaults in shaping responsible behavior in the AI development value chain. The AutoGen framework constructed default set-ups for human oversight that were readily used by downstream developers. Most default set-ups were used consistently. There are exceptions - the Composio repository consistently overrode defaults to use agent code execution outside a protected docker environment. Scaling assessment can inform governance proposals to set responsible and effective defaults.

Future work can refine the code-inspection approach and subsequently scale it for all open source AutoGen applications using AI-assisted grading methods (Eloundou et al., 2024). Autonomy levels can be developed further to reflect internal dependencies among the impact and oversight attributes and to better reflect literature from other fields that uses holistic and more numerous levels of autonomy. Future work could rate autonomy levels on additional categories, such as goal setting, interactions between agent systems and fallback specifications (see Figure 1), though multiple measures complementing code inspection may be needed for such attributes arising at the model layer or at inference time. Code inspection results for select systems can be compared to inference-time assessments using evaluation harnesses (UK AI Safety Institute, 2024). Subsequent work can differentiate between agent applications and platforms, and generalize this approach to assess additional frameworks.

Acknowledgments

The views expressed in this paper are those of the authors alone and do not necessarily reflect the official policy or position of their employers. Thanks to participants and organizers of the NeurIPS SoLaR 2024 workshop, Alan Chan, Shrey Jain, Owen Larter, and three anonymous reviewers for feedback on earlier drafts. Any errors are the authors' alone.

References

Eric Anderson, Timothy Fannin, and Brent Nelson. Levels of aviation autonomy. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pp. 1–8. IEEE, 2018.

Alan Chan, Rebecca Salganik, Alva Markelius, Chris Pang, Nitarshan Rajkumar, Dmitrii Krasheninnikov, Lauro Langosco, Zhonghao He, Yawen Duan, Micah Carroll, et al. Harms from increasingly agentic algorithmic systems. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pp. 651–666, 2023.

Alan Chan, Carson Ezell, Max Kaufmann, Kevin Wei, Lewis Hammond, Herbie Bradley, Emma Bluemke, Nitarshan Rajkumar, David Krueger, Noam Kolt, et al. Visibility into AI agents. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, pp. 958–973, 2024.

Peter Cihon. Chilling autonomy: Policy enforcement for human oversight of ai agents. In *41st International Conference on Machine Learning, Workshop on Generative AI and Law*, 2024. URL `https://blog.genlaw.org/pdfs/genlaw_icml2024/79.pdf`.

Tom Davidson, Jean-Stanislas Denain, Pablo Villalobos, and Guillem Bas. Ai capabilities can be significantly improved without expensive retraining. *arXiv preprint arXiv:2312.07413*, 2023.

Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. Gpts are gpts: Labor market impact potential of llms. *Science*, 384(6702):1306–1308, 2024. doi: 10.1126/science.adj0998. URL https://www.science.org/doi/abs/10.1126/science.adj0998.

EU. Artificial Intelligence Act. Official Journal of the European Union, 2024. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L_202401689&qid=1726146160101. Accessed: 2024-09-12.

Chinedu Pascal Ezenkwu and Andrew Starkey. Machine autonomy: Definition, approaches, challenges and research gaps. In *Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 1*, pp. 335–358. Springer, 2019.

Tom Froese, Nathaniel Virgo, and Eduardo Izquierdo. Autonomy: a review and a reappraisal. In *Advances in Artificial Life: 9th European Conference, ECAL 2007, Lisbon, Portugal, September 10-14, 2007. Proceedings 9*, pp. 455–464. Springer, 2007.

Iason Gabriel, Arianna Manzini, Geoff Keeling, Lisa Anne Hendricks, Verena Rieser, Hasan Iqbal, Nenad Tomašev, Ira Ktena, Zachary Kenton, Mikel Rodriguez, et al. The ethics of advanced AI assistants. *arXiv preprint arXiv:2404.16244*, 2024.

Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 159–166, 1999.

Hui-Min Huang. Autonomy levels for unmanned systems (alfus) framework: safety and application issues. In *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, PerMIS '07, pp. 48–53, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595938541. doi: 10.1145/1660877.1660883. URL https://doi.org/10.1145/1660877.1660883.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Megan Kinniment, Lucas Jun Koba Sato, Haoxing Du, Brian Goodrich, Max Hasin, Lawrence Chan, Luke Harold Miles, Tao R Lin, Hjalmar Wijk, Joel Burget, et al. Evaluating language-model agents on realistic autonomous tasks. *arXiv preprint arXiv:2312.11671*, 2023.

LangChain. What is an agent? *LangChain Blog*, 2023. URL https://blog.langchain.dev/what-is-an-agent/. Accessed: 2024-09-09.

Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2023. URL https://arxiv.org/abs/2308.03688.

METR. Autonomy evaluations protocol guide, 2024. URL https://metr.github.io/autonomy-evals-guide/example-protocol/. Accessed: 2024-09-12.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. Levels of agi: Operationalizing progress on the path to agi. *arXiv preprint arXiv:2311.02462*, 2023.

NIST. Autonomy levels for unmanned systems (alfus) framework. Technical report, National Institute of Standards and Technology (NIST), 2005. URL https://www.nist.gov/system/files/documents/el/isd/ks/NISTSP_1011_ver_1-1.pdf. Accessed: 2024-09-09.

SAE. Iso/sae pas 22736: Definitions for terms related to driving automation systems for on-road motor vehicles. Technical report, International Organization for Standardization (ISO) and SAE International, 2021. URL https://cdn.standards.iteh.ai/samples/73766/63c7c9dd67c147a1a7067be549d9653d/ISO-SAE-PRF-PAS-22736.pdf. Accessed: 2024-09-09.

Yonadav Shavit, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O'Keefe, Rosie Campbell, Teddy Lee, Pamela Mishkin, Tyna Eloundou, Alan Hickey, et al. Practices for governing agentic AI systems. 2023. URL https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf.

Monika Simmler and Ruth Frischknecht. A taxonomy of human–machine collaboration: capturing automation and technical autonomy. *Ai & Society*, 36(1):239–250, 2021.

UK AI Safety Institute. Agents, 2024. URL https://ukgovernmentbeis.github.io/inspect_ai/agents.html.

Kenneth R Walsh, Sathiadev Mahesh, and Cherie C Trumbach. Autonomy in ai systems. *The Journal of Technology Studies*, 47(1):38–47, 2021.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

Guang-Zhong Yang, James Cambias, Kevin Cleary, Eric Daimler, James Drake, Pierre E Dupont, Nobuhiko Hata, Peter Kazanzides, Sylvain Martel, Rajni V Patel, et al. Medical robotics—regulatory, ethical, and legal considerations for increasing levels of autonomy, 2017.

A Appendix

A.1 Levels of autonomy

Table 4 provides a comparative overview of autonomy levels from AI and robotics literatures.

Table 4: Taxonomy of Autonomy Levels in Various Domains

Level	**Autonomous vehicles** (SAE, 2021)	**Medical robots**(Yang et al., 2017)	**AI: Human oversight** (Simmler & Frischknecht, 2021)	**AI: Action space** (LangChain, 2023)	**AI: Task evaluations** (METR, 2024)
0	No automation	No autonomy	No autonomy	Code	–
1	Assistance: Steering-support	Assistance	Human decision: Decision-support AI	LLM call	Few minutes
2	Partial automation: Acceleration- and steering-support	Task autonomy: Human maintains discrete control	Human in the loop: Human approves action	Chain: AI-generated outputs at multiple steps, human specifying action and available actions	Several minutes (Implement simple programs)
3	Conditional automation: Autonomous driving, human fallback	Conditional autonomy: Human selects plan, oversees execution	Human on the loop: Human does not veto action	Router: AI-generated steps and (single) action decision with human-determined available actions	Under an hour or few hours (Debugging etc.)
4	High automation: Fallback by system	High autonomy: System makes decisions under human oversight	Human off the loop: AI takes action and then informs human	State machine: AI-generated steps and action decisions with human-determined available actions	Day long (Replicate ML papers)
5	Full automation: Unlimited environment	Full automation	Human out of the loop: AI takes action independently	Autonomous: State machine with AI-generated steps and action decisions with unbounded actions	Week long or month long (Identify vulnerabilities in network, exploit them)

A.2 Application code inspection

Table 5 shows the code flags used to conduct the code inspections. Full scoring for the ten repositories can be found at: `https://docs.google.com/spreadsheets/d/1f7Ft24a54QapZLdAce6yIVvoyYFixarJijE39MxzMGU`.

Table 5: Code flags for scoring autonomy of selected AutoGen applications

Autonomy	**Impact**		**Oversight**		
	Actions	Environment	Orchestration	Human-in-the-loop	Observability
Main code flag	`code_execution_config`	`use_docker` & `browser_config`	`max_rounds` (GroupChat) or `max_consecutive_auto_reply`	`human_input_mode`	search for `log`, `display`, and `reply_func`
Lower	`=False`	`=True` & not set	`<=1` or never called	`=ALWAYS`	`display_...` is configured
Middle	`=False` & system message with "execute the function"	`=True` & set	`>1`	`=TERMINATE` or mixed	Caching, logging, or tracing is invoked
Higher	Absence of `=False`	`=False`	Not set	`=NEVER`	No logging or similar