Article in Press

https://doi.org/10.1038/s44260-026-00102-1

Computational models of complex systems meet AI agents

Check for updates

Ciro Cattuto[1] ✉ & Alessandro Vespignani[1,2]

Complex systems have structural properties that make them a promising domain for scientific workflows driven by AI agents— systems that combine large language models with tool use and iterative execution. However, most complex systems simulators and libraries do not expose their control points and outputs in a form that text-based AI agents can readily use. Realizing the potential of agent-driven workflows, therefore, requires equipping complex systems models with agent-friendly interfaces.

Background

Scientific research is being reshaped by the emergence of large language model (LLM)-based AI agents that combine natural language capabilities with the ability to use tools and iterate autonomously toward a goal[1]. Specifically, AI agents can interpret natural-language objectives, decompose them into sequences of actions, observe the results, and revise their plans accordingly, without explicit step-by-step guidance. This has led to AI systems that assist and actively participate in the scientific research process[2]. Early demonstrations have shown agents capable of end-to-end scientific workflows, spanning literature search, hypothesis generation, experiment design, and manuscript drafting[3,4].

Two research directions have attracted attention at the intersection of AI and complex systems. The first one uses LLMs as synthetic actors inside simulations, replacing human agents in social or behavioral models[5–8]. The second direction applies complex systems concepts to study LLM-based agents themselves[9]. Here we discuss another research use of AI agents that has received less attention: the agent as an external operator of a complex systems simulator, driving it toward scientific goals defined by a human researcher. In this perspective, agents may become the primary interface through which complex systems models are operated to conduct exploratory research and operational work. This calls for designing complex system simulators that LLM-based agents can effectively exploit. This perspective is analogous to past work on automated scientific investigation pipelines [King et al.[10]], active learning for discovery [Kusne et al.[11]] and the idea of a "robot scientist" [Burger et al.[12]] autonomously designing and carrying out experiments. Here, we identify gaps in complex systems simulators and discuss how to design them to better support hybrid human-AI workflows and delegation to LLM-based AI agents.

Complex systems science is a natural domain for AI agents

The opportunities opened by AI agents are not uniform across scientific fields. For structural reasons, complex systems science is a particularly favorable case for applying AI agents. The mechanistic rules that define complex systems operate at the microscopic scale, while the collective behaviors of interest emerge at the macroscopic scale. The gap between scales is a hard problem that is generally approached computationally. This involves simulating the model's mechanistic process at the individual-agent scale, observing the emergent behavior, adjusting parameters or rules, and then simulating the system again by sweeping the parameter space. Multiple computational models are frequently used to model the same phenomena: a network-based model of a contagion process or a system of ordinary differential equations for the same process offer different lenses for capturing the system's dynamics, and any finding is more definitive when they agree.

In addition, the most actionable complex systems science occurs when the system is perturbed. In most relevant, realistic situations, the space of possible perturbations is too large to explore analytically and too structured to sample at random. The general workflow is that empirical data inform the model, which guides the design of interventions/perturbations to reshape the system and generate new data. The scientist sits at the center of this loop, iterating the process to achieve the desired insights. While this workflow is challenging for researchers, it maps closely onto what an AI agent does well: iteration and ensemble logic, multiple models used in parallel, and deliberate search through a structured space are natural automation paths for AI agents. Matching complex systems science with LLM agents is therefore so natural that the interesting question is not whether they can assist this kind of science, but what becomes possible once they drive it.

[1]ISI Foundation, Turin, Italy. [2]Laboratory for the Modeling of Biological and Socio-technical Systems, Northeastern University, Boston, MA, USA.
✉e-mail: ciro.cattuto@isi.it

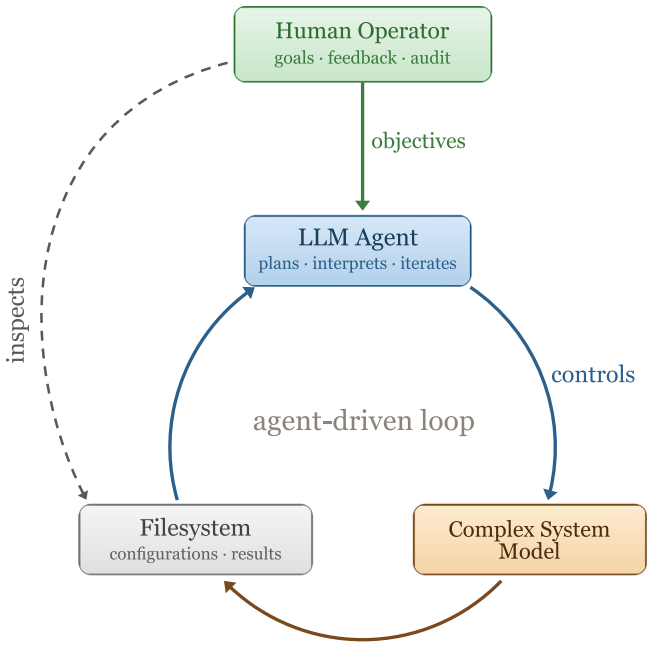


Fig. 1 | Agent-in-the-loop complex systems science workflows.

Agent frameworks in complex systems science

A major opportunity offered by AI agents is a redefinition of the complex systems scientist's role in the modeling workflow. Most complex systems simulators were built to keep the human in the loop, through user interfaces, data visualization, and interactive environments such as programming notebooks. An agent can take that role by controlling the model implementation, while the human operator sets the agent's higher-level goals. The agent runs the computational loop. The researcher holds scientific intent and epistemic responsibility by formulating research questions, interpreting results, and adapting/redirecting the research agenda based on findings. This lets the scientist focus on the scientific process rather than the mechanics of running simulations and parameter sweeps.

However, it is not just a matter of automation and speed. Agent-driven workflows differ qualitatively from established practices and open the door to new ways of organizing the study of complex systems. The most significant of these is the **systematic exploration of the intervention space**. Complex systems structure admits a large number of possible perturbations: which components to target, when to act, at what intensity, in what sequence. This space is too large to explore randomly and often too complex to characterize exhaustively by analytical means. An AI agent can navigate this space autonomously by formulating hypotheses, running targeted simulations to test them, and updating its strategy based on the results. Agents can detect patterns that a human might miss and autonomously plan a sequence of simulations by dynamically constructing the workflow. The user no longer needs to plan each intermediate step as the agent handles it autonomously. This is a machine-intelligence-driven mode of investigation that goes beyond the simple automation of existing practice.

Together, the above capabilities define novel simulation frameworks for complex systems, in which **the computational model is embedded in an agent-driven loop**: the agent plans successive simulation actions, interprets the model's output, and iterates toward objectives defined by the human operator (Fig. 1).

Design principles for agent-friendly interfaces

Current-generation AI agents, such as Claude Code or OpenAI Codex, are driven by Large Language Models (LLMs) that rely on text to interact with users and control the tools available to the agent[13,14]. Complex system simulators or libraries, on the other hand, have traditionally been designed to be driven either by users, with a focus on user-friendly interfaces, or programmatically by user-written programs and computational notebooks. An LLM-based agent using the simulator/library is often forced to inspect its source code and documentation to determine which models and configuration parameters are available; it needs to write one-off custom programs to drive the library; and it struggles to process the volume of output the simulator generates. For these reasons, delegating workflow control to an agent carries specific risks. Misspecified objectives or incorrect but plausible intermediate outputs can lead to a confidently wrong result[15]. These risks set a requirement for the agent workflow itself: the scientist must remain able to verify, direct, and take responsibility for what the agent does. Much of the interface design we outline here is shaped to address these limitations.

In the following, we delineate six core principles for designing or wrapping complex systems simulators and libraries with an interface layer (an "agent-friendly interface") that allows AI agents to control them better and leverage their capabilities in the context of complex, iterative workflows:

- **Discoverability/Introspection:** A model should be able to describe itself through structured queries that report its structure, parameters, and their valid ranges. Components, states, transitions, and available interventions should be discoverable, without the agent having to read source code or documentation. This is the foundation, since it allows an agent to plan a simulation run. It is also what lets an LLM-based agent bring the model's configuration into its context window.
- **Declarative control:** The agent should control the simulator through self-contained, structured configuration artifacts rather than through a sequence of imperative methods or API calls[2]. For configuration information, LLM-friendly formats such as JSON and YAML are particularly suitable. Configuration artifacts should be validatable before execution, allowing the agent to verify its plan before committing to a run. Domain-specific subtleties should be encoded in configuration fields rather than left to the agent's first-principles reasoning.
- **Statelessness:** Every operation should be a self-contained function of its inputs: a configuration file, previously saved output files, etc. Dynamically allocated objects, session state, and ordering constraints across tool calls should all be avoided. Agents have no reliable way to persist objects in memory across calls, and simulators that require them to model and manage their state are fundamentally agent-hostile.
- **Structured, queryable output:** Results should be written to the filesystem in a self-describing format, and the interface should provide tools for querying the raw output. The agent can use query tools to ask targeted questions (e.g., summary statistics, final state, scenario comparison) and obtain compact structured answers. It doesn't need to load or parse the full simulation output directly.
- **Human-Agent parity:** The interface should be equally legible and operable by a human researcher and an AI agent, with no need for a translation layer. A command-line interface (CLI) naturally fits this requirement. For example, a CLI that accepts YAML configurations and returns structured JSON puts the researcher and the agent on equal footing, as they can both use the same commands and inspect the same input and output files. Moreover, a CLI works equally well outside of an agent-driven loop: the operator can inspect what the agent did by replaying its commands in a terminal. This is an architectural commitment to transparency: a scientist who can reproduce agent actions can verify and trust the agent's workflow. Protocols such as the Model Context Protocol (MCP) have greater expressive power but interpose a translation layer that breaks this parity and often requires stateful servers[16].
- **Composability:** When multiple tools share the above design principles and operate on shared filesystem-backed artifacts, they can be chained into agent-orchestrated computational pipelines. Composability emerges when tools are built in accordance with the preceding principles. It is also what enables multi-model workflows: for example, an agent could run an ODE simulator, an ABM-based simulator, and a

calibration engine in sequence, passing outputs from one to the next as inputs.

The six principles illustrated above are design principles. They can be built incrementally into a complex systems simulator or library, verified by inspection, and adopted independently. In fact, in many cases, it is not necessary to modify the library or simulator at all: it is sufficient to wrap existing tools with a thin layer, designed according to the above principles, that implements CLI tooling, parameter registries, configuration templates, outputs' query tools, etc., creating an agent-friendly control surface.

However, a library or simulator that satisfies the above design principles without announcing them offers no reliable way for an agent to use the agent-friendly interface. It is therefore essential to provide, within the agent's context window, documentation of the interface in the form of an **agent contract**. In practice, this is a dedicated file (AGENT.md, by convention) that specifies the complete interface contract and is read by the agent at the beginning of the workflow. It documents the discovery and introspection mechanisms, the configuration format, the output data files, and the query commands. It ensures that the provided interface is stable and complete, and that it contains the domain-specific conventions an agent must know to use the library correctly. A simulator or library that provides an AGENT.md makes a verifiable commitment about its agent readiness. We argue that this should become a standard provision of agent-friendly scientific software. The agent contract itself should not be regarded as a static artifact. It can be iteratively improved, e.g., through agent-on-agent review. In this approach, one agent completes a suite of representative tasks using an agent framework that includes the AGENT.md contract. A second agent reviews the first agent's work by inspecting conversation logs, tool calls, and the generated outputs. This way, the reviewer agent may identify errors and ambiguities that prevented successful use of the simulator or library, and these findings can then be incorporated into a revised version of the AGENT.md contract, either manually or by the reviewer agent itself. The AGENT.md contract becomes the object of a continuous testing and refinement cycle analogous to software development. Just as usability testing reveals shortcomings in human-facing interfaces, agent-on-agent review exposes deficiencies in machine-facing interfaces.

Finally, current-generation AI agents can read source code, write and execute custom scripts, and directly call the underlying libraries' application programming interfaces (APIs). The agent-friendly interface we envision here does not remove these capabilities. Its aim is the robustness and reproducibility of the modeling workflow. When a task genuinely exceeds what the interface provides, the agent can inspect source code and write custom scripts. This escalation route ensures the agent interface never becomes a ceiling on what the agent can do, and frequent escalation may signal which capabilities the interface should cover.

An illustrative example

Here we illustrate the principles of section "Design principles for agent-friendly interfaces" in relation to Epydemix, a Python library for epidemic modeling[17]. The goal is to show that an agent-friendly interface can be built incrementally on top of a real, pre-existing complex systems library, enabling current-generation AI agents to drive end-to-end modeling workflows.

Epydemix provides stochastic compartmental models (SIR, SEIR, SIS, fully custom compartmental structures) with age-structured populations, interventions, and calibration to observed data. Its use cases exemplify all the properties discussed in Section "Complex systems science is a natural domain for AI agents": a vast but structured intervention space, a data-model-intervention loop, and a multi-scale architecture. Its original interface is typical of complex systems libraries, which are hard for an agent to drive. Models and their parameters live in Python code, so the only way to find out what is configurable is to read the source. A run is assembled by instantiating objects and calling methods in a script. There is no single declarative description of a run that an agent could write down, check, and then execute. Results come back as in-memory Python objects bound to a live session. An agent working through stateless tool calls has no way to retain them or return to them later. Finally, outputs are inspected by querying large binary output files with custom scripts, so there is no compact summary that an agent can inspect without loading and parsing full simulation trajectories. None of these are shortcomings of a library developed for human researchers, but they considerably limit automated use by agents.

The principles in Section "Design principles for agent-friendly interfaces" are meant to overcome the above hurdles and inform the design of specific artifacts that support an AI agent's use of the library. **Introspection** can be achieved by creating a *parameter registry* that the agent queries before planning a run: a machine-readable catalog of every configurable quantity, its type, valid range, and default value. **Declarative control** is achieved by introducing an LLM-friendly *configuration file* (e.g., in YAML format) that fully specifies a simulation, allowing the agent to confirm that its plan is coherent before committing to a run and avoiding the need to generate custom code for common scenarios. **Human–agent parity** leads to the design of a single, text-based control interface for both human users and agents. This can be achieved, for example, by creating a command-line interface (CLI) that wraps the existing Python API and allows an AI agent to drive the library by calling the CLI via its tool-use capabilities. The **Statelessness** principle requires that every CLI operation is a pure function of its input files, producing self-contained directories of result files (*bundles*) that also include a *manifest* file describing them, so that no session state needs to persist across CLI calls. **Structured output** can be realized by an *inspection/query tool* that returns summaries (e.g., ensemble statistics, epidemic peaks, scenario comparisons) so that the agent doesn't have to parse raw simulation output. **Composability** follows from defining an output bundle format: outputs are directly usable as inputs to other tools and because each bundle's manifest records its parent, provenance information is available and multi-step pipelines are traceable from information available in the filesystem alone. The **agent contract document** consists of an AGENT.md document that specifies the canonical epidemic modeling workflows supported by Epydemix, the format of configuration files, the available CLI commands to drive the library, the output format of each command, and conventions for reporting results and creating tables and figures. The agent contract is complemented by other files containing worked examples that the agent can use as a reference.

Overall, an agent-friendly interface can be delivered as a strictly additive layer on top of the Epydemix library[18]. Existing simulation routines and result objects behave exactly as before. Similarly, existing notebooks continue to work unchanged. This choice is essential because scientific libraries accumulate a community of users and downstream dependencies. In practice, a path to agent readiness that requires a substantial rewrite of the library is not viable.

This exercise provides a few general take-home messages. The six principles we outlined previously can be implemented in a concrete architecture that other complex systems libraries can adopt, without committing to any particular technology stack. We also show that wrapping existing libraries rather than rewriting them is an effective approach. The cost of making a library agent-ready is limited, the user community is not harmed, and the resulting agent-friendly interface supports LLM agents and other automated pipelines. Finally, the performance of an agent framework can be readily verified: the agent should be able to produce useful, reproducible results given only a natural-language scientific request and the AGENT.md contract. If a library meets this criterion on a representative set of tasks, it is agent-ready. Otherwise, the observed failures indicate which principles need strengthening. The complete example discussed here is available online (github.com/epistorm/epydemix/tree/agent-framework) and described in detail in Ref. Gozzi et al. [18].

The path forward

The complex systems community already has a mature ecosystem of validated simulators and libraries. What it lacks are interfaces that make those capabilities discoverable and controllable by AI agents. The priority is to

build such layers on top of the tools the community already trusts. However, designing these layers critically depends on domain-specific knowledge (what a model is for and where it breaks down); therefore, this is a task for the complex systems community of practice, not one that can be delegated to AI practitioners alone.

Two steps would accelerate progress on the gaps we have highlighted. One is the convergence on **shared conventions** for introspection, declarative configuration, structured output, and contract documents. This would enable agent-friendly tools across the ecosystem to compose with one another. The second is the **development of benchmarks** for agent-driven complex systems modeling tasks: community-developed tests that measure whether an agent, given only a natural-language goal and a contract document, can produce correct and reproducible results. The agent readiness of simulators and libraries would therefore become measurable, enabling the community to track its own progress.

The evolution towards agent-in-the-loop complex systems science is, we believe, inevitable. We think it is important that this transition happens on terms set by the complex systems community, preserving transparency, reproducibility, and scientific rigor.

Data availability

No datasets were generated or analyzed during the current study.

Received: 28 August 2026; Accepted: 5 October 2026;
Published online: 04 November 2026

References

1. Yao, S. et al. ReAct: synergizing reasoning and acting in language models. https://arxiv.org/abs/2210.03629 (ICLR, 2023).
2. Wang, H. et al. Scientific discovery in the age of artificial intelligence. *Nature* **620**, 47 https://doi.org/10.1038/s41586-023-06221-2 (2023).
3. Lu, C. et al. Towards end-to-end automation of AI research. *Nature* **651**, 914 https://doi.org/10.1038/s41586-026-10265-5 (2026).
4. Gottweis, J. et al. Accelerating scientific discovery with co-scientist. *Nature* https://doi.org/10.1038/s41586-026-10644-y (2026).
5. Park, J. S. et al. Generative agents: interactive simulacra of human behavior. In *Proc. 36th Annual ACM Symposium on User Interface Software and Technology* (ACM, 2023).
6. Gao, C. et al. Large language model Empowered agent-based modeling: a survey. *Humanit. Soc. Sci. Commun.* **11**, 1259 https://doi.org/10.1057/s41599-024-03611-3 (2024).
7. Lu, Y. et al. LLMs and generative ABMs for complex systems research. *Phys. Life Rev.* **51**, 284 https://doi.org/10.1016/j.plrev.2024.10.013 (2024).
8. Pedreschi, D. et al. Human-AI coevolution. *Artif. Intell.* **339**, 104244 https://doi.org/10.1016/j.artint.2024.104244 (2025).
9. Krakauer, D. C., Krakauer, J. W., & Mitchell, M. Large language models and emergence: a complex systems perspective. Preprint at https://arxiv.org/abs/2506.11135 (2025).
10. King, R. D. et al. The automation of science. *Science* **324**, 85 (2009).
11. Kusne, A. G. et al. On-the-fly closed-loop materials discovery via Bayesian active learning. *Nat. Commun.* **11**, 5966 (2020).
12. Burger, B. et al. A mobile robot chemist. *Nature* **583**, 237 https://www.nature.com/articles/s41586-020-2442-2 (2020).
13. Yang, J. et al. SWE-agent: agent-computer interfaces enable automated software engineering. NeurIPS 2024 https://doi.org/10.52202/079017-1601 (2024).
14. Wang, Y. et al. From imperative to declarative: towards LLM-friendly OS interfaces for boosted computer-use agents. In *Proc. Eurosys '26: Proceedings of the 21st European Conference on Computer Systems* https://doi.org/10.1145/3767295.3803576 (2025).
15. Messeri, L. and Crockett, M. J. AI and illusions of understanding in scientific research https://www.nature.com/articles/s41586-024-07146-0 (2024).
16. Pan, C. et al. "Experiences with model context protocol servers for science and high performance computing". Preprint at https://arxiv.org/abs/2508.18489 (2025)
17. Gozzi, N. et al. Epydemix: An open-source Python package for epidemic modeling with integrated approximate Bayesian calibration. *PLoS Comput. Biol.* **21**, e1013735 https://doi.org/10.1371/journal.pcbi.1013735 (2025).
18. Gozzi, N. et al. Driving epidemic models with AI agents: the Epidemix agent framework. Preprint at https://arxiv.org/abs/2609.28692 (2026).

Acknowledgements

C.C. and A.V. acknowledge partial support from the Lagrange Project of the ISI Foundation, funded by CRT Foundation. A.V. acknowledges support from the cooperative agreement CDC-RFA-FT-23-0069 from the CDC's Center for Forecasting and Outbreak Analytics. The findings and conclusions in this study are those of the authors and do not necessarily represent the official position of the funding agencies, the CDC, or the U.S. Department of Health and Human Services of the United States.

Author contributions

C.C. and A.V. contributed equally to the conceptualization of the work, development of the ideas and arguments presented in the manuscript, and writing and revision of the manuscript. Both authors reviewed and approved the final version.

Funding

C.C. and A.V. received support from the Lagrange Project funded by CRT Foundation. A.V. received support under CDC-RFA-FT-23-0069 from CDC/CFA.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to Ciro Cattuto.

Reprints and permissions information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2026

This Article in Press is shared early to give you faster access to new research. It is citable and carries a permanent DOI. The final edited version will replace it automatically.