Letter

Agentic artificial intelligence for multistage physics experiments at a large-scale user facility particle accelerator

Thorsten Hellert,* Drew Bertwistle, Simon C. Leemann, Antonin Sulc, and Marco Venturini
Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA

(Received 21 September 2025; accepted 5 December 2025; published 16 January 2026)

We present a language-model-driven agentic artificial intelligence (AI) system to autonomously execute multistage physics experiments on a production synchrotron light source. Implemented at the Advanced Light Source particle accelerator, the system translates natural language user prompts into structured execution plans that combine archive data retrieval, control-system channel resolution, automated script generation, controlled machine interaction, and analysis. In a representative machine physics task, we show that preparation time was reduced by 2 orders of magnitude relative to manual scripting even for a system expert, while operator-standard safety constraints were strictly upheld. Core architectural features, plan-first orchestration, bounded tool access, and dynamic capability selection, enable transparent, auditable execution with fully reproducible artifacts. These results establish a blueprint for the safe integration of agentic AI into accelerator experiments and demanding machine physics studies, as well as routine operations, with direct portability across accelerators worldwide and, more broadly, to other large-scale scientific infrastructures.

DOI: 10.1103/jtqy-9jz1

Introduction. Particle accelerators such as the Advanced Light Source (ALS) [1] are among the most complex scientific instruments, enabling frontier research in material science [2,3], chemistry [4], and biology [5]. Their operation requires continuous oversight by teams with expertise spanning accelerator physics [6], rf systems [7], magnets [8], vacuum [9], diagnostics [10], and controls [11]. Because subsystem knowledge is distributed, operators frequently rely on domain specialists for troubleshooting, advanced tuning, or nonstandard experimental procedures. As user facilities, maximizing availability and protecting user beam are primary objectives; at the ALS, any beam interruption typically imposes a downtime of at least 30 min—and potentially several hours—with immediate consequences for dozens of concurrent experiments across more than 40 beamlines.

Many accelerator tasks extend beyond routine tuning, requiring custom scripts and deep subsystem knowledge. At the ALS, the control system exposes more than 230 000 process variables (PVs). Troubleshooting can be particularly demanding: Unexpected faults lack predefined solutions, forcing operators to identify relevant channels, retrieve archive data, and assemble ad hoc analysis under time pressure. Because each case is unique, preparation overhead and cognitive load are substantial, directly limiting machine availability and reducing scientific throughput across all beamlines. These challenges motivate the development of agentic systems that can translate user intent into structured, reproducible procedures while upholding strict facility safety constraints.

*Contact author: thellert@lbl.gov

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

Recent advances in language models (LMs) point to a class of agentic systems [12] capable of bridging the gap between complex infrastructures and intuitive human interfaces. Beyond fluent text generation, LMs have been shown to support structured reasoning [13], enabling them to decompose complex objectives into sequential steps. Building on this foundation, Toolformer [14] demonstrated that models can be trained to call external tools, while ReAct [15] introduced reasoning-acting loops that tightly couple deliberation with execution. These ideas have since been extended to multiagent orchestration [16], memory-augmented systems [17], and graph-based planning frameworks [18], highlighting the potential of agentic artificial intelligence (AI) to provide structured, inspectable execution. Yet, most demonstrations remain confined to simulated or low-stakes domains; several domain-focused systems illustrate this diversity: ChemCrow augments LMs with chemistry-specific tools [19], Co-scientist enables autonomous experimental planning in chemistry [20], and CRISPER-GPT applies agentic orchestration to gene-editing workflows [21]; and in synchrotron science, beamline prototypes such as VISION [22] have been explored, but reports of autonomous, multistage operation in production environments at large user facilities are still outstanding.

The constraints of accelerator facilities illustrate why even small mistakes, such as a mistuned rf parameter or an incorrect magnet setting, can cause extended downtime, beam loss, or hardware damage, with immediate impact not only on machine health but also on dozens of concurrent experiments across all beamlines. These high-stakes conditions underscore the need for interfaces that are both intuitive and auditable.

Early explorations have already applied LMs to accelerator operation in targeted ways. GAIA [23] introduced a prototype assistant at an R&D linac, demonstrating how an LM could interface with logbooks, trigger control routines, and

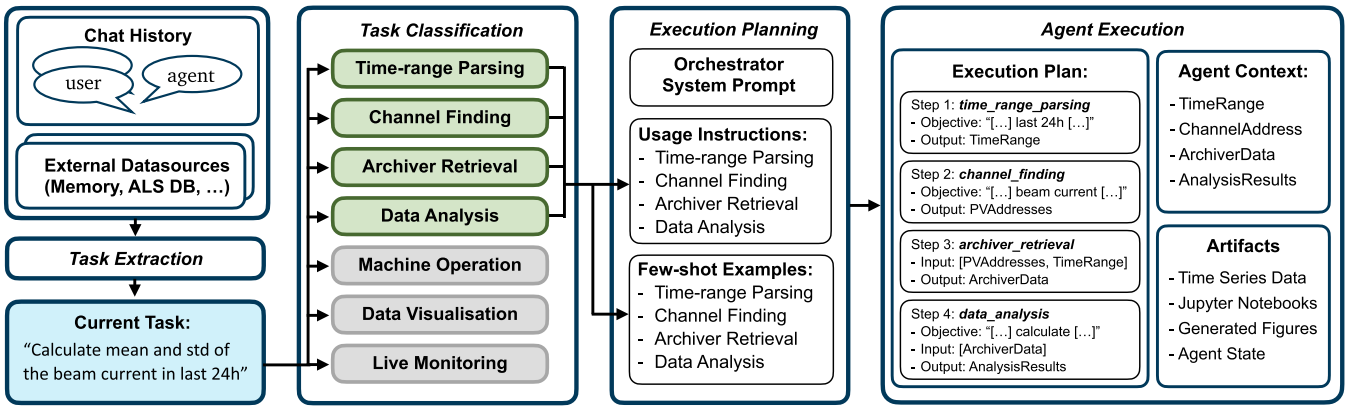

FIG. 1. Overview of the agentic workflow. Multiturn conversational input and external data sources are first processed into a structured task. Relevant capabilities are dynamically classified on each iteration of the interaction, and the description of the selected tools is passed to the execution planner. The planner generates a complete, inspectable execution plan with explicit dependencies, which is then carried out by the agent with context tracking and artifact management.

support operator workflows. Kaiser, *et al.* [24] presented a proof-of-principle study where an LM performed beamline optics optimization from natural language prompts, directly comparing its performance against established optimization methods. Sulc, *et al.* [25] outlined a broader vision for integrating AI systems into accelerator control. While these works highlight the promise of LM-based interfaces, they remained limited in scope as either conceptual roadmaps or single-task demonstrations.

Here, we advance beyond these prototypes by presenting the deployment of an LM-based multiagent system in a production synchrotron, the *Accelerator Assistant*. The system provides a natural language interface to an EPICS control environment [26] and extends beyond simple read/write access, enabling intuitive interaction with the control system through natural language: From a single user prompt, it can retrieve archive data [27], resolve PVs, generate and execute scripts, and analyze results. In a representative experiment (cf. *From query to experiment*), the system autonomously prepares and executes a complete multistage procedure, reducing preparation time by 2 orders of magnitude relative to manual scripting by experts while strictly maintaining operator-standard safety constraints.

These capabilities are enabled by several architectural advances. Plan-first orchestration captures every task as an inspectable execution plan with optional operator approval. Dynamic capability filtering ensures stable scaling across a large tool inventory, while structured artifacts, including JUPYTER NOTEBOOKS [28], json outputs, and logs, provide reproducibility and transparency. Validating these methods under the strict availability and safety requirements of the ALS demonstrates that agentic AI can be used safely in high-stakes environments. The approach offers a blueprint for broader integration of LM-driven systems into large-scale scientific facilities, with direct portability to other synchrotrons, accelerators, and complex scientific infrastructures.

ALS agentic control framework. The workflow of the agentic system follows a modular, capability-centric design that emphasizes separation of concerns following the OSPREY framework [29]. Each natural language fragment is first normalized into structured objects, ensuring that downstream components receive standardized inputs free of ambiguity. This modularity allows subsystems to be flexibly combined, enabling the framework to expand with capabilities without requiring retraining or modification of existing components. An overview of the complete workflow is shown in Fig. 1. All elements of the system, including source code, configuration files, and deployment scripts, are publicly available [30].

Users interact with the system either through a command line interface or via the OPEN WEBUI [31], accessible from every control room station as well as remotely through secure terminal access. Authentication is tied to individual user identities, enabling the framework to maintain personalized context and memory across sessions. Multiple conversations can be managed in parallel, allowing users to organize distinct tasks or experiments into separate threads. As illustrated in Fig. 2, input is routed through the Accelerator Assistant, which orchestrates connections to the PV database, archive service, and JUPYTER-based execution environments [32]. Model inference is performed either locally using OLLAMA [33] on an NVIDIA H100 node located within the control room network or externally via the CBORG [34] gateway; a lab-managed interface that routes requests to external providers such as CHATGPT [35], CLAUDE [36], or GEMINI [37]. This hybrid architecture balances secure, low-latency on-premises inference with access to state-of-the-art foundation models, while integration with EPICS enforces operator-standard safety constraints for direct interaction with accelerator hardware.

At each turn, conversational input is translated into a concise and well-structured natural language task description that isolates objectives and removes redundancy. External knowledge sources, such as personalized memory stores tied to user identities, documentation, and accelerator databases, are incorporated to ground terminology and context. The resulting specification provides a clear objective, giving downstream components an unambiguous basis for execution while remaining human-readable.

The system organizes its functional units into modular *capabilities*: self-contained tools for data retrieval, machine interaction, or analysis—that can be composed as needed for a given task. Capabilities are then classified for

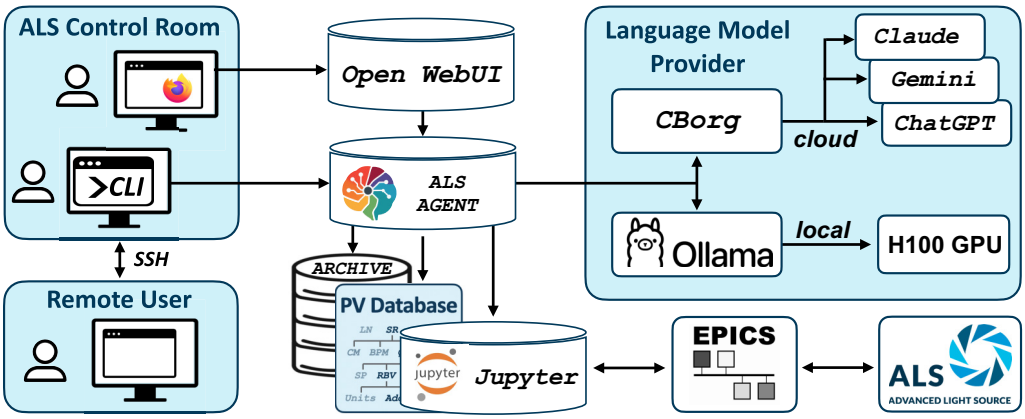

FIG. 2. System architecture of the Accelerator Assistant. Control room and remote users access the system via a web interface (OPEN WEBUI) or command line, which routes requests to the ALS Agent. The agent orchestrates connections to the PV database, archive data, and execution environments such as JUPYTER. Model inference is performed either locally using OLLAMA or through cloud providers via the CBORG gateway. Integration with EPICS enables safe interaction with accelerator hardware at the ALS.

relevance to the current task. Each capability is evaluated independently, with the classification posed as a binary decision using few-shot, capability-specific examples. Only those deemed relevant are passed forward, preventing prompt inflation and decoupling task complexity from the size of the overall capability inventory, which ensures efficient operation and allows the framework to scale as capabilities are added.

Execution proceeds via a plan-first orchestration strategy: Before any tool is called, the system generates a complete execution plan that encodes explicit input-output dependencies. This separation of planning and execution ensures that logic remains transparent, serializable, and subject to inspection or modification. Plans also provide natural checkpoints where safety gates can be enforced: Operators or automated validators may review inputs, outputs, and dependencies prior to the initiation of any action.

The execution environment is modular and containerized via PODMAN [38], ensuring reproducibility across both development workstations and production control room servers. Core components, including the agent, OPEN WEBUI, and JUPYTER services, run in isolated containers, allowing for consistent deployment, straightforward upgrades, and strict separation of privileges. Reliability features include checkpointing, structured error classification, and bounded retries with automatic replanning when required. Human-in-the-loop interrupts are supported, allowing users to inspect and approve plans, code, or memory operations before side effects occur. Every run produces structured artifacts (including logs, JSON outputs, and JUPYTER NOTEBOOKS) that provide a complete provenance trail, enabling reproducibility, auditing, and further development of workflows. In addition, the agent can materialize monitoring artifacts from natural language requests: for example, the prompt "Monitor the beam current and RF cavity temperature" generates a CS-STUDIO PHOEBUS DATA BROWSER file [39], a widely used control room toolkit with deep EPICS integration and the standard interface at the ALS, preconfigured to query the Archive Appliance for historical context while updating in real time. These autogenerated panels both eliminate PV lookup and data entry overhead and persist as reusable control room resources, yielding a practical speedup for routine observation and troubleshooting.

Although this work is demonstrated on the ALS, a mature and well-characterized storage ring, recent developments in the underlying OSPREY framework extend the approach to a broader range of accelerator environments. The integration of advanced agentic code-generation modules, such as the CLAUDE CODE SDK [40], allows each facility to provide example code, conventions, and safety patterns that guide the Python generator to follow local operational practices. In parallel, OSPREY's safety layer supports configurable PV boundary limits as well as write blacklists and whitelists, enabling facilities to define safe operating ranges without modifying the framework. These configuration-level mechanisms make the system adaptable to variations in actuator behavior, diagnostic conditions, and metadata quality, providing a practical path for deployment at facilities with different levels of maturity.

From query to experiment. To convey the capabilities of the Accelerator Assistant, we focus on a nonroutine but practically important machine physics task. Such procedures are complex enough to require custom scripting but occur too infrequently for dedicated solutions to exist, making them an ideal proving ground for agentic control. In this case, the task involves insertion devices (IDs) [41]: tunable undulator magnets whose gap settings strongly affect both machine optics and delivered photon beams.

As an illustration, the user made the following request:

> "Get the minimum and maximum value of all ID gap values in the last three days. Then write a script which moves each ID from maximum to minimum gap and back while measuring the vertical beam size at beamline 3.1. Sample the gap range with 30 points, wait 5 s after each new setpoint for the ID to settle and measure the beam size 5 times at 5 Hz. Return a hysteresis plot beam size vs gap."

This demonstration highlights both the scope and the impact of the system. While the beam-based measurement sequence itself necessarily requires about an hour, the preparation effort is reduced from what would typically take several

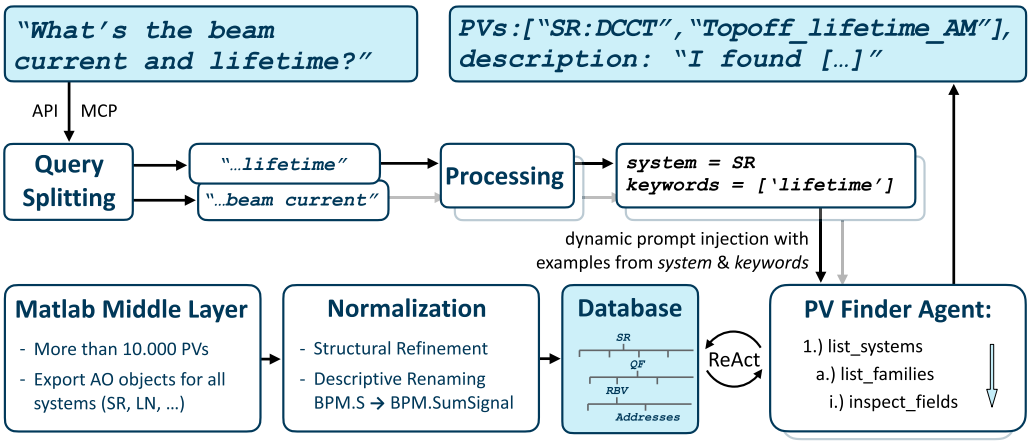

FIG. 3. Workflow of the PV Finder subsystem. A normalized export of the MATLAB Middle Layer Accelerator Object provides the data model, which the agent explores through a strictly bounded API. Natural language queries are split into atomic intents, preprocessed to extract systems and keywords, and resolved into specific EPICS PVs.

hours of manual scripting and debugging to only a few minutes from a single natural language prompt—a speedup of 2 orders of magnitude. This efficiency results from the system's ability to transform free-form user input into a structured execution plan that decomposes the request into modular steps.

The resulting execution plan organizes the task into a small number of safety-gated stages that can be inspected and, depending on configuration, approved either at every step or only for sensitive operations. In the present deployment at the ALS, the system has been configured to require operator approval for all write access to the control system. The stages comprise time-range normalization, PV resolution against the accelerator middle layer, archive retrieval, data analysis, controlled machine interaction, and visualization.

The workflow begins with time range parsing, a dedicated language processing task performed by a lightweight model to ensure low latency. Rather than relying on brittle pattern matching, the model flexibly interprets natural language fragments such as "last three days" and normalizes them into standardized `datetime` ranges that downstream services, such as archive queries, can directly process.

A central step is PV resolution, handled by the PV Finder subsystem (Fig. 3). A normalized export of the MATLAB Middle Layer (MML) [42] Accelerator Object provides the underlying data model, including approximately 10 000 key PVs across all accelerator subsystems. Because the MML is implemented at most synchrotron light sources worldwide, this foundation makes the approach naturally transferable to other storage rings, with only minor refinements and descriptive renaming required to maintain a consistent, interpretable database structure for the language model. User queries are split into atomic intents, preprocessed to extract target systems and keywords, and then resolved into specific PVs by a REACT-style agent restrained to a strictly bounded application programming interface (API). This tool-bounded exploration guarantees auditability while grounding ambiguous user terminology, such as "ID gap" or "beam size," into precise EPICS channel names.

Once PVs and time ranges are resolved, archive retrieval reduces to a straightforward API call to the archiver client. Input mapping is handled automatically by the orchestrator, returning time series data for all relevant IDs without user intervention.

Execution then proceeds through dynamically generated Python scripts (Fig. 4). To make this stage robust, code generation is decomposed into three successive model calls rather than a single direct request, which can be brittle and prone to overdesign. First, the model produces a high-level plan of what the script should achieve. Second, this plan and the user's objective are used to generate a structured JSON schema specifying the expected results. Finally, conditioned on both the plan and schema, the model produces the minimal Python code required to carry out the task. Scripts are executed inside containerized JUPYTER kernels with strict read/write policies, supporting two modes: read-only (analysis and visualization) and write-enabled (machine interaction), the latter requiring operator approval by configuration (default policy), with read-only analysis as the baseline mode. All code may be reviewed prior to execution, and every run produces structured artifacts (JUPYTER NOTEBOOKS, JSON results, and figures) for provenance and reproducibility.

In the aforementioned example, the Python executor is invoked three times. First, it computes the minimum and maximum gap values from archived data. Second, it generates and runs a scan script that sweeps ID gaps between these values while recording synchronized beam size measurements. Third, it visualizes the acquired data in the form of a hysteresis plot, confirming the absence of significant beam size hysteresis. Importantly, these are not just three generic code-generation calls but invocations of distinct, specialized capabilities for data analysis, machine operation, and data visualization. Each capability follows the same modular architecture and execution flow illustrated in Fig. 4, yet is guided by tailored prompts that reflect its specific domain. This modularity has proven essential for achieving robust performance in the control room, ensuring that natural language user requests can be translated into complex, end-to-end experimental procedures with full auditability.

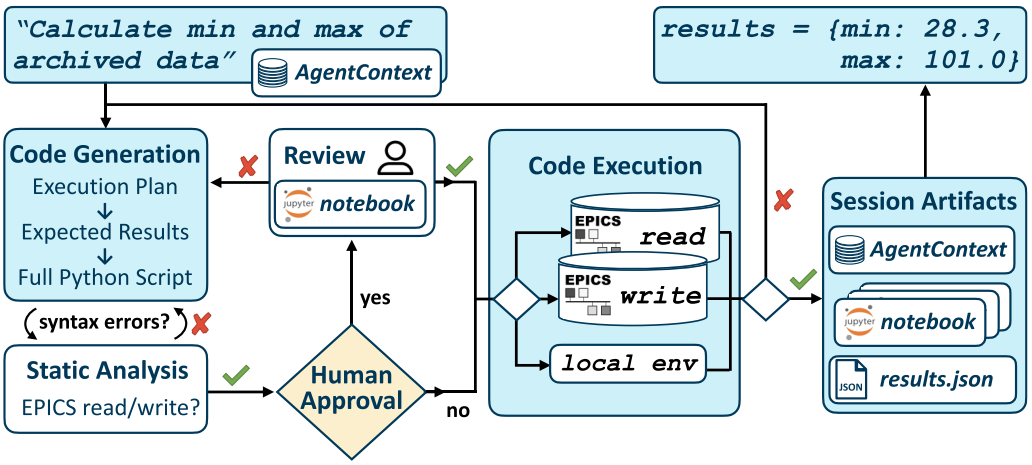

FIG. 4. Pipeline for controlled Python code execution in the Accelerator Assistant. Natural language tasks are translated into a plan, results schema, and then Python code, which can dynamically access the agent context, is statically analyzed, and may be reviewed by a human operator. Execution is typically confined to containerized JUPYTER kernels with strict read/write policies, and every run produces session artifacts (context, notebooks, JSON) for full reproducibility.

By chaining these modular capabilities, the system not only retrieves and analyzes archived data, but also orchestrates real-time machine operation. The result here is a series of consistent and publication-ready plots across all devices. An example is shown in Fig. 5, demonstrating the expected absence of hysteresis in the vertical beam size.

This experiment illustrates how the Accelerator Assistant converts natural language requests into fully executed physics measurements. Beyond this specific example, the same architecture applies to a wide range of machine physics tasks, providing a reproducible bridge from user intent to automated execution.

Conclusion. We have presented the deployment of an LM-based agentic system executing a multistage physics experiment on a production synchrotron light source storage ring. By integrating directly with the EPICS control system and Archive Appliance, the Accelerator Assistant demonstrates that natural language user requests can be converted into safe, auditable, and fully automated experimental procedures. The agent reduces preparation time by 2 orders of magnitude while preserving operator-standard safety constraints, and it produces reproducible artifacts that support inspection and trust.

Key architectural features, including plan-first orchestration, bounded tool access, and dynamic capability selection, ensure that the framework scales with growing functionality while remaining transparent and portable across accelerator facilities.

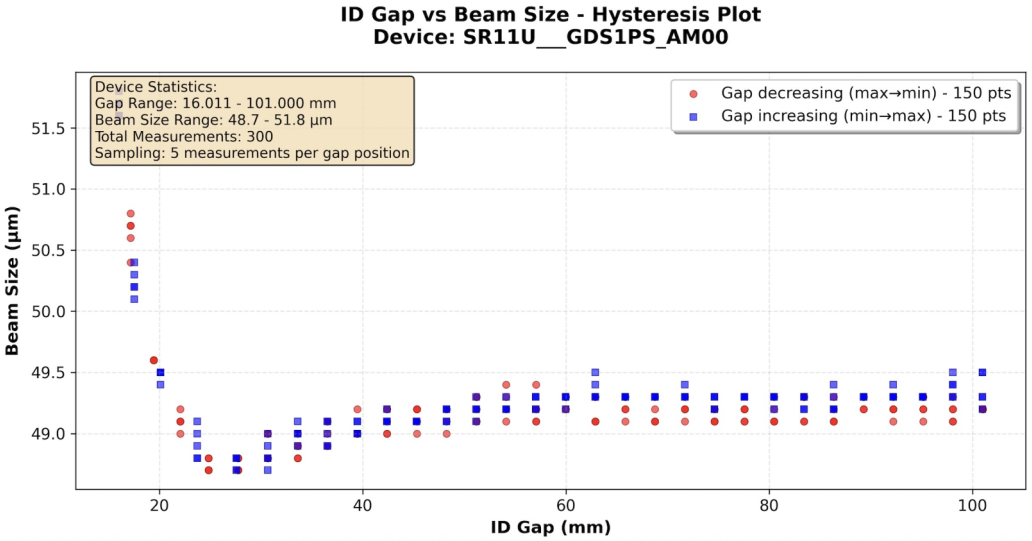

FIG. 5. Example output of the Accelerator Assistant: hysteresis measurement of ID gap vs vertical beam size at the ALS. The execution plan generated by the agent combined historical range extraction, automated script generation, and real-time machine control. The agent performed a 30-point bidirectional gap sweep with five repeated measurements per point, producing the plot shown here for one device. This figure illustrates the final output of the agentic workflow, where every step, from parsing natural language to data retrieval, machine control, and plotting, was generated and executed automatically.

These results establish a blueprint for the safe integration of agentic AI into scientific operations. Beyond synchrotrons, the demonstrated principles are broadly applicable to other large-scale experimental facilities, where automation, transparency, and reproducibility are equally critical.

Acknowledgments. This research leveraged the CBORG AI platform and resources provided by the IT Division at Lawrence Berkeley National Laboratory. We gratefully acknowledge Andrew Schmeder for his consistent responsiveness and support, which ensured that CBORG served as an invaluable resource for the development of this framework and natural language processing efforts in general at ALS. We are grateful to Alex Hexemer, Hiroshi Nishimura, Fernando Sannibale, and Tom Scarvie (LBNL) for stimulating discussions and continued support, and to Frank Mayet (DESY) for sharing insights from his pioneering GAIA prototype, which guided the early trajectory of agentic AI at ALS. We further acknowledge the use of AI tools during the preparation of this work. CURSOR, primarily with CLAUDE 4, was employed extensively during development. This work was supported by the Director of the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Data availability. The data that support the findings of this Letter are openly available [1].

[1] T. Hellert, B. Flugstad, C. Sun, C. Steier, E. Wallén, F. Sannibale, G. Portmann, H. Nishimura, J. Weber, M. Venturini, M. Dach, S. C. Leemann, S. Omolayo, S. Borra, T. Scarvie, and T. Ford, in *Proceedings of the 15th International Particle Accelerator Conference* (JACoW Publishing, Geneva, Switzerland, 2024), paper TUPG37.

[2] C. Chen, *et al.*, Strong electron-phonon coupling in magic-angle twisted bilayer graphene, Nature (London) **636**, 342 (2024).

[3] S. Tan *et al.* Spontaneous formation of robust two-dimensional perovskite phases, Science **388**, 639 (2025).

[4] S. K. Chandy, M. Lopez Luna, N. Z. Rustad, I. N. Zakaria, A. Siebert, S. Devlin, W. Li, M. Blum, and T. Head-Gordon, Ammonia synthesis under ambient conditions: Insights into water–nitrogen–magnetite interfaces, J. Am. Chem. Soc. **147**, 24538 (2025).

[5] C. Y. Ralston, *et al.*, ALS-ENABLE: Creating synergy and opportunity at the Advanced Light Source synchrotron structural biology beamlines, J. Synchrotron. Rad. **32**, 1059 (2025).

[6] H. Wiedemann, *Particle Accelerator Physics*, *Graduate Texts in Physics* (Springer International Publishing, Cham, 2015).

[7] H. Damerau, Radio-frequency (RF) systems, Technical Report, CAS Course: Introduction to Accelerator Physics (CERN Accelerator School).

[8] J. Tanabe, *Iron Dominated Electromagnets: Design, Fabrication, Assembly and Measurements* (World Scientific, Singapore, 2005), p. 354.

[9] O. B. Malyshev, *Vacuum in Particle Accelerators: Modelling, Design and Operation of Beam Vacuum Systems* (Wiley-VCH Verlag GmbH & Co, Weinheim, Germany, 2019).

[10] M. G. Minty and F. Zimmermann, *Measurement and Control of Charged Particle Beams*, *Particle Acceleration and Detection* (Springer, Berlin, 2003).

[11] J. J. DiStefano III, A. R. Stubberud, and I. J. Williams, *Schaums Outline of Feedback and Control Systems*, 2nd ed. (McGraw-Hill Professional, New York, 1997).

[12] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. (Prentice Hall, Hoboken, NJ, 2020).

[13] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2022), Vol. 35, pp. 24824–24837.

[14] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, Toolformer: Language models can teach themselves to use tools, Adv. Neural Inf. Process. Syst. **36**, 68539 (2023).

[15] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, ReAct: Synergizing reasoning and acting in language models, arXiv:2210.03629.

[16] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, AutoGen: Enabling next-gen LLM applications via multi-agent conversation, arXiv:2308.08155.

[17] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, and J. E. Gonzalez, MemGPT: Towards LLMs as operating systems, arXiv:2310.08560.

[18] LangGraph developers, LangGraph: A low-level orchestration framework for building resilient, stateful agents (2025), https://github.com/langchain-ai/langgraph.

[19] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller, Augmenting large-language models with chemistry tools, Nat. Mach. Intell. **6**, 525 (2024).

[20] D. A. Boiko, R. MacKnight, and G. Gomes, Autonomous chemical research with large language models, Nature (London) **624**, 570 (2023).

[21] Y. Qu, K. Huang, M. Yin, K. Zhan, D. Liu, D. Yin, H. C. Cousins, W. A. Johnson, X. Wang, M. Shah, R. B. Altman, D. Zhou, M. Wang, and L. Cong, CRISPR-GPT for agentic automation of gene-editing experiments, Nat. Biomed. Eng. (2025).

[22] S. Mathur, N. V. der Vleuten, K. G. Yager, and E. H. R. Tsai, VISION: A modular AI assistant for natural human-instrument interaction at scientific user facilities, Mach. Learn.: Sci. Technol. **6**, 025051 (2025).

[23] F. Mayet, GAIA: A general AI assistant for intelligent accelerator operations, arXiv:2405.01359.

[24] J. Kaiser, A. Lauscher, and A. Eichler, Large language models for human-machine collaborative particle accelerator tuning through natural language, Sci. Adv. **11**, eadr4173 (2025).

[25] A. Sulc, T. Hellert, R. Kammering, H. Houscher, and J. St. John, Towards agentic AI on particle accelerators, arXiv:2409.06336.

[26] L. R. Dalesio, J. O. Hill, M. Kraimer, S. Lewis, D. Murray, S. Hunt, W. Watson, M. Clausen, and J. Dalesio, The experimental physics and industrial control system architecture: Past, present, and future, Nucl. Instrum. Methods Phys. Res. Sect. A **352**, 179 (1994).

[27] M. Shankar, M. Davidsaver, M. Konrad, and L. Li, The EPICS archiver appliance, in *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)* (JACoW, Melbourne, Australia, 2015), pp. 761–764.

[28] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides and B. Schmidt (IOS Press, Amsterdam, Netherlands, 2016), pp. 87–90.

[29] T. Hellert, J. Montenegro, and A. Sulc, Osprey: A scalable framework for the orchestration of agentic systems, arXiv:2508.15066.

[30] ALS Assistant Developers, ALS assistant repo, (2025), https://github.com/als-apg/als_assistant.

[31] Open WebUI (2023), https://github.com/open-webui/open-webui.

[32] B. E. Granger and F. Pérez, Jupyter: Thinking and storytelling with code and data, Comput. Sci. Eng. **23**7 (2021).

[33] Ollama Team, Ollama: Get up and running with large language models (2023), https://github.com/ollama/ollama.

[34] Lawrence Berkeley National Laboratory, Science IT Group, CBORG AI portal, Lawrence Berkeley National Laboratory AI service platform (2024), https://cborg.lbl.gov/.

[35] OpenAI, ChatGPTt language model family (2022), https://openai.com/chatgpt, accessed 15 September 2025.

[36] Anthropic, Claude language model family (2023), https://docs.anthropic.com/en/docs/about-claude/models/overview.

[37] Google DeepMind, Gemini language model family (2023), https://deepmind.google/models/gemini/.

[38] Red Hat, Inc., Podman: A daemonless OCI-compliant container engine, version 4.x (2024), https://podman.io/.

[39] K. Shroff, T. Ashwarya, T. Ford, K.-U. Kasemir, R. Lange, and G. Weiss, Phoebus tools and services, in JACoW, Proc. ICALEPCS2023 (2023).

[40] Anthropic, Claude agent SDK for Python, GitHub, 2025, https://github.com/anthropics/claude-agent-sdk, documentation: https://docs.anthropic.com/en/docs/claude-code/sdk/sdk-python.

[41] Undulators, *Wigglers and Their Applications*, edited by H. Onuki and P. Elleaume (CRC Press, Boca Raton, 2003).

[42] G. Portmann, J. Corbett, and A. Terebilo, An accelerator control middle layer using MATLAB, Conf. Proc. C **0505161**, 4009 (2005).