Mini Review | Open Access

AI Agent

Hu *et al. AI Agent* **2026**, *2*, 21 **DOI:**10.20517/aiagent.2026.38

Computational materials agents: from task demonstrations to executable scientific workflows

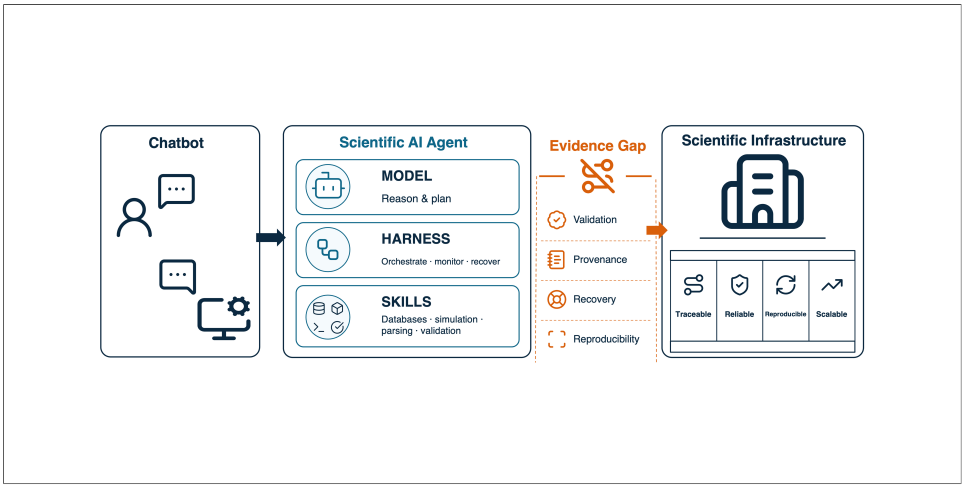
Chenhui Hu[1], **Zhenbin Wang**[2,3]

Keywords:
Artificial intelligence agents, computational materials, large language models, benchmarks

Citation: Hu, C.; Wang, Z. Computational materials agents: from task demonstrations to executable scientific workflows. *AI Agent* **2026**, *2*, 21. https://dx.doi.org/10.20517/aiagent.2026.38

Received: 30 Jun 2026
First Decision: 28 Jul 2026
Revised: 4 Aug 2026
Accepted: 20 Aug 2026
Published: 4 Sep 2026

Academic Editor:
Jianjun Liu
Copy Editor:
Tong Wang
Production Editor:
Tong Wang



Abstract

Computational materials modeling connects physical theory, atomistic mechanisms, and materials design, but its value depends on workflows that are specified, executed, checked, and interpreted with scientific rigor. Agents built on large language models are beginning to link materials knowledge with databases, simulation software, workflow controllers, and feedback mechanisms, enabling greater automation of computational research. However, most reported successes remain concentrated in scaffolded settings for density functional theory, molecular dynamics, and related atomistic simulations, where tasks, engines, validators, and target outputs are predefined. This review analyzes computational materials agents across three interconnected dimensions: architecture, executable workflow practices, and evaluation evidence. We examine how current systems translate materials questions into computational tasks, identify the settings in which reliability and recovery have been demonstrated, and explain why runnable calculations should be distinguished from scientifically justified conclusions. By clarifying the evidence required for traceable, reproducible, and reusable workflows, this review aims to guide the development of computational materials agents from task demonstrations to practical scientific tools.

[1]Hong Kong Institute of AI for Science, City University of Hong Kong, Hong Kong 999077, China.
[2]Department of Materials Science and Engineering, City University of Hong Kong, Hong Kong 999077, China.
[3]School of Energy and Environment, City University of Hong Kong, Hong Kong 999077, China.

Correspondence to: Dr. Zhenbin Wang, Department of Materials Science and Engineering, City University of Hong Kong, Hong Kong 999077, China. E-mail: zwan22@cityu.edu.hk

INTRODUCTION

Large language models (LLMs) have changed how scientific information is accessed, summarized, and recombined, but question answering alone is insufficient for scientific work. Many research tasks require systems that interact with external resources and adjust their actions in response to observations. Artificial intelligence (AI) agents provide this capability by coupling a model with planning, action, observation, and feedback[1]. In ReAct-like settings, reasoning and action are interleaved in repeated think-act-observe cycles. Tool-use systems[2] further show that models can operate through application programming interfaces (APIs), calculators, databases, or instruments.

Computational modeling of materials and molecular systems is central to modern materials research because it connects physical theory, atomistic mechanisms, and measurable materials properties[3]. First-principles calculations, molecular dynamics (MD), thermodynamic modeling, and high-throughput screening allow researchers to test hypotheses, evaluate candidate materials, interpret experiments, and guide design decisions before or alongside laboratory work[4-6]. However, such studies rarely consist of a single calculation. They require choices about structures, models, parameters, convergence criteria, software environments, post-processing methods, and physical interpretation. The reliability of a computational result therefore depends not only on its numerical output but also on whether the full workflow is traceable, reproducible, and scientifically justified. These requirements make computational materials modeling a natural but demanding setting for agentic systems. Recent studies have begun exploring agentic coordination in this domain[7-9]. Established computational workflow systems already support standardized execution, provenance tracking, job management, and reproducible calculations within predefined procedures[10-12]. However, designing and adapting these procedures still requires substantial domain expertise and practitioner judgment. LLM-based agents can complement such systems by lowering technical barriers through natural-language interaction, enabling more flexible coordination when procedures are not fully specified, and linking computational workflows to automated experimental platforms.

Against this background, studies of LLM-based materials agents have increased markedly since 2023 [Figure 1]. Reported systems span literature extraction[13], candidate design[14,15], synthesis planning[16], autonomous experimentation[17,18], closed-loop discovery[19], and computational modeling[20-23]. Within this landscape, computational materials agents have focused primarily on density functional theory (DFT), MD, and related atomistic simulations. These systems organize materials knowledge, simulation software, databases, and feedback signals around executable calculations, allowing materials questions to be translated into computational tasks. The central question is whether they can support routine research by setting up, executing, checking, and reporting calculations that produce usable computational evidence.

In this review, we examine computational materials agents as executable scientific workflow systems rather than as isolated language or tool-use demonstrations. We first describe their architectures through a model-harness-skill framework, then trace their operation across computational workflow stages, and finally analyze how current evaluation practices shape claims about reliability, reuse, and scientific correctness. This organization clarifies what current agents can already support, where the evidence remains limited, and what is needed to move from task demonstrations toward reusable computational materials workflows.

ARCHITECTURES FOR COMPUTATIONAL MATERIALS AGENTS

Common requests in computational materials research, such as structure optimization, band-structure calculation, or adsorption energy evaluation, require agents to choose methods, prepare structures and inputs, run domain software, and inspect logs and outputs. An agent architecture must therefore connect scientific intent with executable skills while keeping files, errors, checks, and decisions available for inspection and revision.

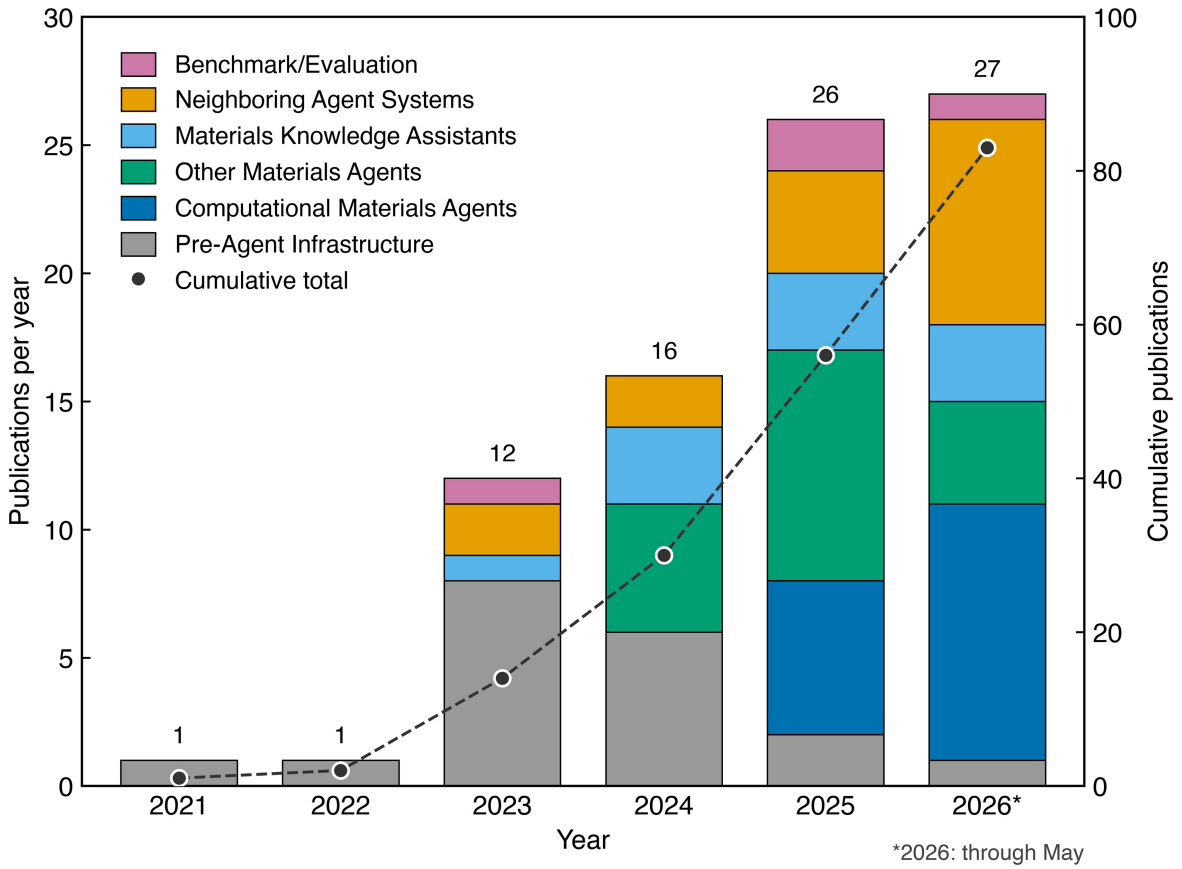


Figure 1. Curated publication landscape of materials-agent studies by year and category. Stacked bars show the annual number of included works from 2021 through May 2026. Works are grouped into benchmark/evaluation studies, neighboring agent systems, pre-agent infrastructure, materials knowledge assistants, other materials agents, and computational materials agents. Open circles connected by a dotted line show the cumulative number of works. The distribution highlights a shift from early pre-agent infrastructure toward rapidly expanding agent-based systems after 2023. Computational materials agents have become a major component of recent studies. Studies were identified by searching arXiv and Google Scholar using combinations of the terms "materials agent", "computational materials agent", "atomistic simulation agent", and "materials agent benchmark". Retrieved records were screened for direct relevance to materials-agent systems, executable scientific workflows, or agent evaluation. Reviews and perspectives were excluded. Duplicate preprint and journal versions were counted only once, and each included study was assigned to a single primary category based on its principal contribution.

Drawing on general formulations of LLM reasoning and tool use[1,2] and recurring functional components in the computational chemistry and materials-agent systems reviewed here[24,25], we organize the architectural discussion around a coupled model-harness-skill framework [Figure 2A]. The model provides reasoning and generation capabilities. The harness turns model outputs into controlled workflows by managing task decomposition, tool routing, workflow state, and validation or stopping rules. Skills are callable procedures or tool wrappers through which the agent acts on external resources, files, software, or data. Although these roles are often coupled in practice, distinguishing them clarifies their functions. The harness determines when and in what order actions are taken, whereas skills provide the executable channels for those actions.

Materials-specific design can be implemented at any layer of this architecture. At the model layer, a system may use a materials-specialized language model or a broader materials foundation model to improve the use of domain terminology and support reasoning about structures, compositions, and properties[26-28]. LLaMat illustrates model-layer specialization. It combines continued pretraining on tens of billions of materials-science tokens from millions of publications and crystallographic records with subsequent instruction and

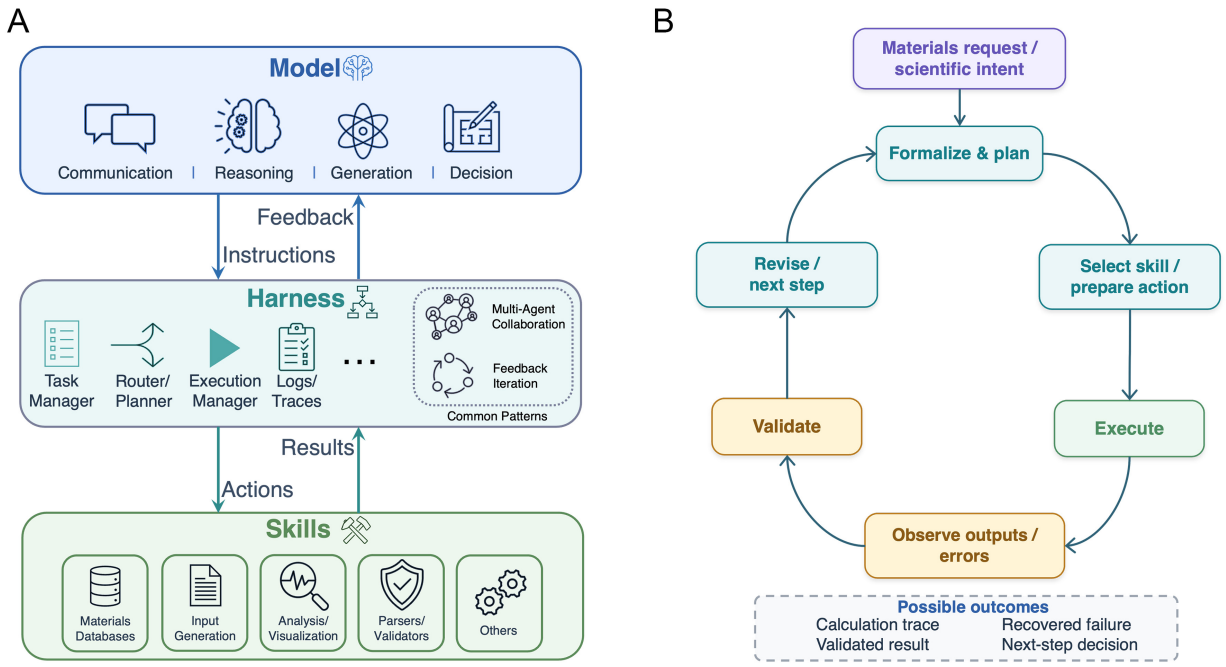


Figure 2. Model-harness-skill framework and executable workflow of computational materials agents. (A) Model-harness-skill framework. The model supports reasoning and generation, the harness controls the workflow, and domain skills execute actions through materials databases, structure tools, simulation codes, parsers, validators, and analysis tools; (B) Executable workflow loop. A materials request is converted into a computational task. Domain software executes the task, whose outputs and error messages are then evaluated. Feedback determines whether the task is revised, accepted, or stopped. The figure was assembled by the authors using Microsoft PowerPoint and the generic graphical symbols were selected from the built-in Microsoft 365 icon library.

task fine-tuning on a large materials question-answering corpus[27]. Such specialization can improve performance on materials-related tasks, including language understanding, extraction, and generation, but it is resource-intensive and not uniformly beneficial. The study also reports adaptation rigidity in extensively pretrained base models[27]. The benefit of a materials-specialized LLM therefore depends on the base model, training procedure, task family, and evaluation setting.

Given these limitations of model-layer specialization, many current agent systems place materials-specific knowledge in the harness and skills. In such systems, domain knowledge is embodied in task templates, software-specific validators, retry rules, input and output handlers, and interfaces to simulation codes or computing resources. These functions are often implemented together rather than separated into distinct harness and skill layers. TritonDFT[20] maps DFT requests to Quantum ESPRESSO (QE) subproblems and executable calls, parses and evaluates the outputs, and retries with revised parameters. MDAgent2[21] organizes Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) code generation around syntax and potential-file checks, execution feedback, and regeneration; MatClaw[25] uses Python code as the action interface for materials libraries and remote high-performance computing (HPC) workflows, with memory and source-code retrieval supporting repair during long tasks.

Harness-level specialization can also be implemented through role-based organization in multi-agent systems. Agents with different responsibilities handle distinct parts of the calculation[29]. The multi-agent atomistic-simulation framework of Vriza *et al.* provides a concrete example[30]. An administrator agent routes a user request through structure-generation, potential-search, LAMMPS-input, HPC-execution, property-specific, and results-analysis agents. In the gold (Au) workflow, these roles generate the face-centered cubic (FCC) structure and locate and validate an embedded-atom method (EAM) potential. They then prepare and submit LAMMPS jobs to an HPC cluster and parse lattice-constant and cohesive-energy outputs.

A complementary approach keeps the coordinator general and places most domain specificity in skill packages. OpenClaw-based computational-chemistry skills[24] illustrate this pattern: OpenClaw maintains task state, reads execution feedback, and decides which skills to invoke, whereas the skills supply domain-specific procedures. In the methane-oxidation case study[24], skills for molecular conversion, packing, LAMMPS/Deep Potential Molecular Dynamics (DeePMD) execution, job dispatch, and trajectory analysis assemble a reactive MD workflow spanning molecular preparation through reaction-pathway analysis. This separation facilitates reuse because the coordinator can remain general while the skills encapsulate executable domain procedures, validation logic, and recovery behavior.

In summary, coupled workflow controllers can be efficient and relatively easy to validate within a fixed software stack, but they may be less transferable when the engine, file conventions, or target property changes. Multi-agent role separation makes domain responsibilities and handoffs visible, but it also adds coordination overhead and more pathways for errors to propagate between roles. General coordinators with domain skills support reuse, yet their performance depends on whether the skills expose sufficient procedural detail, validation hooks, and failure information for the harness to control them. For computational materials agents, the architecture determines which parts of a calculation can be inspected, replaced, or reused, and whether the workflow can recover from failure.

COMPUTATIONAL WORKFLOW PRACTICES

Building on the architectures discussed above, recent computational materials agents can perform a range of tasks[31-33]. Although their software stacks, target tasks, and execution environments differ, they often follow the same basic process: they translate a user request into a calculation plan, implement the plan as runnable inputs or scripts, execute them with domain software, and parse or check the resulting outputs [Figure 2B].

A request for a band gap, diffusion barrier, adsorption energy, elastic constant, or thermodynamic quantity may leave unspecified the structure source, level of approximation, convergence target, reference state, allowable computational cost, and acceptable error. To reduce this uncertainty, a user's scientific request should be formalized in terms of the materials system, target property, method family, constraints, and success criteria before input generation, execution, and checking. The user specifies some choices; others are drawn from templates, defaults, or software-specific recommendations. TritonDFT[20] makes this step concrete by mapping the request to DFT task classes such as self-consistent calculation, variable-cell relaxation, band structure, or density of states. Through a knowledge-graph-based recommendation layer, GENIUS[23] interprets a natural-language request in light of QE parameter details and constraints, then uses this interpretation to initialize the workflow before input generation. QUASAR[22] emphasizes the planning aspect of this stage: its strategist interprets a user-defined research objective, decomposes it into scientifically grounded subtasks, and uses a second planning pass to identify missing domain-specific steps before execution. User-adjustable granularity and accuracy settings further determine the depth of task decomposition and the stringency of the calculation protocol.

During executable setup, the agent uses skills and tool interfaces to convert the formalized task into runnable files or software calls. TritonDFT[20] links DFT task classes to QE workflow functions and executables: self-consistent calculations, variable-cell relaxations, and band-mode calculations use pw.x, whereas band-structure and density-of-states post-processing steps use bands.x and dos.x, respectively. In GENIUS[23], QE input generation begins with the template returned by the recommendation system; the language model then selects from the suggested parameters based on the user prompt and additional context. For MD, MDAgent2[21] uses a Writer LLM to draft a LAMMPS script and an output-file contract before tool-based checks. This contract provides a concrete basis for validating the files produced by LAMMPS. These examples also show why executable setup remains environment-dependent: many successful demonstrations

rely on predefined engines, prompt constraints, local file availability, and benchmark task design. If the environment changes, a generated input may be syntactically plausible yet still fail because a pseudopotential is missing, a potential file is misnamed, a package version interprets a keyword differently, an HPC queue is unavailable, or the directory state differs from the expected state. Executable setup should therefore record sufficient information about software versions, file locations, auxiliary models, and resource assumptions so that another user or the agent in a subsequent run can determine what was executed.

After execution, agents must extract usable results from raw simulation outputs. TritonDFT[20] parses QE outputs to obtain structures, symmetry information, energies, band structures, and density-of-states results. QUASAR[22] places greater emphasis on preserving the run record: archived runs retain raw outputs, parsed summaries, plots, reports, and associated metadata. Beyond extraction, output handling requires organizing simulation products into task-level records and determining whether the results answer the scientific question. Current agents are most reliable at extraction and organization when the output formats are known. Interpretation depends more strongly on physical context, reference choices, uncertainty, and comparisons with prior knowledge. A numerical result can support different claims depending on the calculation route and the scientific question; the result is useful only when interpreted in that context.

To keep a calculation or simulation moving through the workflow, checks and repairs are required at multiple points. Generated inputs or scripts must be tested against domain software, execution logs must be inspected, and parsed outputs must be checked against task criteria. In MDAgent2[21], the drafted LAMMPS script is revised through tool-mediated syntax and potential-file checks: the syntax tool identifies code-level errors, whereas the potential-file tool verifies whether the specified LAMMPS potential file is locally available or recommends the top-k most similar alternatives. The Writer LLM uses this feedback to regenerate code until it passes the checks or an iteration limit is reached. In GENIUS[23], the generated QE input is syntactically validated by running QE; failed validation triggers automated error handling to diagnose crash messages and repair the protocol. In these examples, repair begins only after a tool, validator, or result criterion identifies a local error. Some repairs restore input validity or keep an engine running; others address non-convergence, missing outputs, or results that fail a task criterion. A correction loop can improve workflow continuity, but it does not by itself validate the physical model, convergence protocol, or interpretation. Repair loops therefore show both whether the computation can proceed and which types of failure the agent can detect and address.

These checks become workflow-level decision points when their signals guide retries, acceptance, stopping, or replanning. In TritonDFT[20], unsatisfactory results can trigger parameter revisions and reruns; in QUASAR[22], evaluator feedback can return a task to the Operator for a limited retry before it is accepted or marked as unresolved. AutoDFT extends this pattern in its Vienna Ab initio Simulation Package (VASP) workflows[34]: a dual-path monitor, a recovery agent, and a step reflector diagnose failed or physically implausible steps, revise inputs or plans, and limit the number of recovery attempts. These mechanisms can detect missing files, invalid scripts, startup failures, convergence problems, or mismatches with predefined task criteria.

Not every detected failure should trigger the same response. A formatting error may require only a local edit, whereas an unsuitable method or workflow route may require replanning. A retry may be inexpensive during input generation but costly once a production DFT calculation or long MD simulation has been launched. In some cases, stopping is preferable to continuing with an unsupported calculation. In discovery settings, these decisions must also account for branch value, computational budget, and the expected information gain from an additional calculation. Current demonstrations include early forms of such decision-making, generally implemented through predefined validators or limited retry policies.

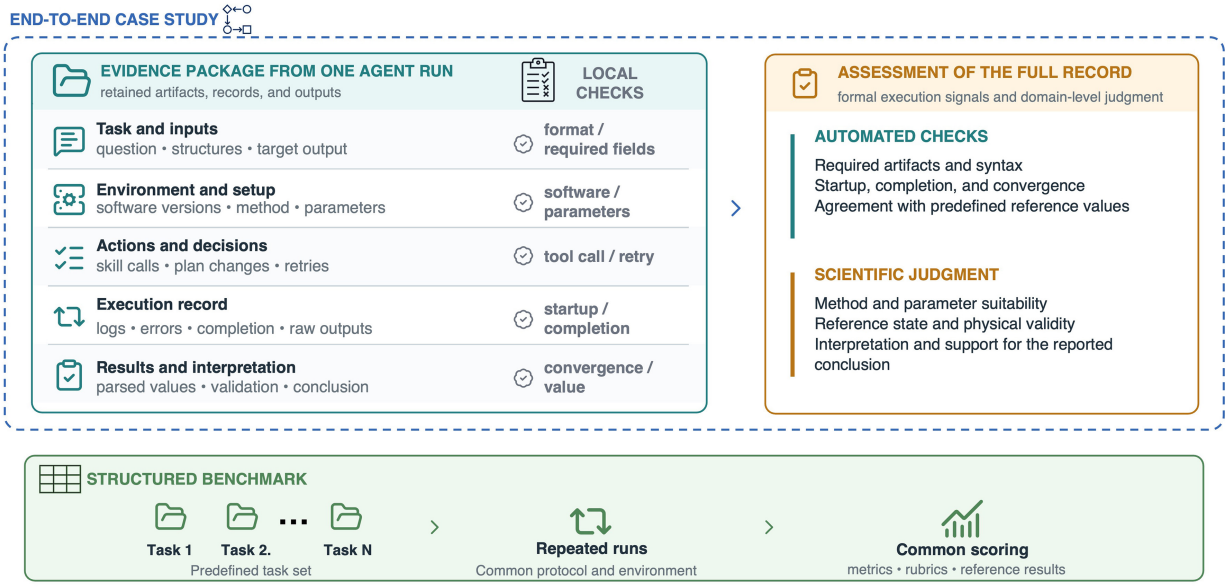


Figure 3. Common evaluation forms for computational materials agents. Local component checks assess formal properties of individual artifacts or execution steps. An end-to-end case study places these checks within the complete record of a selected agent run, including the task and inputs, software setup, actions and decisions, execution record, and interpreted results. Structured benchmarks evaluate performance across a predefined task set through repeated runs conducted under a common protocol and environment, with scoring based on metrics, rubrics, or reference results. The figure was assembled by the authors using Microsoft PowerPoint. Icons were adapted from Lucide Icons and used under the ISC License and, where applicable, the MIT License for Feather-derived icons.

Most reported successes still come from scaffolded DFT, MD, and atomistic tasks with well-specified software environments, file conventions, validation points, and target outputs[20,21,34,35]. These studies show that agents can progress through several stages of a computational workflow rather than merely make isolated code or tool calls. However, these demonstrations remain limited in scope because many otherwise challenging aspects of open-ended computation are predefined: the engine is known, the task type is constrained, the expected output is identifiable, and the failure modes are often detectable by formal checks.

BENCHMARKING AND EVALUATION EVIDENCE

Reported workflow successes raise a further evaluation question: what claims can be supported by a correct answer, a valid tool call[36], a runnable input file, a completed run, or a scientifically justified conclusion? To compare heterogeneous studies, we classify current evaluation practices into three forms: local component checks, end-to-end case studies, and structured benchmarks. Figure 3 illustrates their relationship: local checks assess specific elements of an agent run, case studies examine the complete record of a selected workflow, and structured benchmarks evaluate performance across predefined tasks and repeated runs under common conditions. Across all three forms, the strength of the resulting claims depends on the completeness of the workflow record and on whether formal execution checks are complemented by scientific assessment.

Early LLM benchmarks, such as MaScQA[37] and MSQA[38], commonly evaluated the accuracy of models' answers to materials-science questions[39]. As the field progressed toward materials-agent systems, MatTools extended this evaluation paradigm from question answering to materials tool use and executable code[40]. More recent studies align more closely with simulation practice by incorporating runtime feedback, short correction loops, and selected workflow steps under predefined conditions. For example, MDAgent2[21] evaluates LAMMPS script generation together with execution feedback, syntax and potential-file checks, limited regeneration loops, and expert scoring.

Such local component checks support specific but limited claims. They can show that a model knows relevant terminology, can select a tool, can generate a script that passes a known check, or can revise an artifact after a visible error. These capacities are necessary for workflow agents; weak performance at this level would make longer workflows fragile. However, local success does not show how errors accumulate, whether intermediate assumptions are preserved, or whether an apparently correct step remains valid when embedded in a longer calculation. For computational materials agents, these checks mainly test individual workflow components and therefore cannot establish the reliability of a complete scientific workflow.

To evaluate behavior across complete workflows, several studies rely on case-study evidence. QUASAR[22], for example, uses three tiers of selected atomistic-simulation tasks to test single-task execution, literature-grounded workflow orchestration, and performance on more open-ended simulation problems. Because these studies report end-to-end cases and release some run records, they support assessments beyond isolated tool-level success: planning, execution, error response, and reporting can be evaluated within a complete workflow trajectory. Related studies of agents for DFT[33], MD[21,41], and other atomistic simulations take two approaches: they evaluate selected examples against expert judgments or reference results, or pair workflow demonstrations with tests of retrieval, input preparation, code generation, or evaluator components.

The value of such case studies is that they preserve the workflow episode intact, allowing integration failures to become visible when a later step depends on earlier choices or local resources. A limitation is that the cases are selected: they demonstrate connected execution in the reported settings but do not establish whether the same behavior generalizes across other task families, materials systems, software environments, or failure modes.

More structured evaluations specify the task family, run conditions, or scoring rules before execution. For DFT-focused materials agents, TritonDFT's DFTBENCH[20] asks the agent to complete QE workflows for 68 crystalline materials across four task types: variable-cell relaxation, static calculations, band-structure calculations, and density-of-states calculations. It then evaluates completion, accuracy-cost trade-offs, and parallelization choices against expert-derived references. GENIUS[23] targets a different point in the workflow: its 295 researcher-written prompts test whether a natural-language DFT request can be converted into a QE input setup that survives a 60-second validation run. Recovery is assessed through the automated error-handling loop rather than on the basis of converged production calculations or the correctness of final properties. El Agente Sólido[42] emphasizes repeatability and expert judgment through seven types of solid-state exercises, Level 1 and Level 2 prompt variants, repeated trials, reported results, and rubrics developed by computational chemists. These designs make each study more controlled than an open demonstration, but the evaluation settings still differ substantially in task definitions, software stacks, prompt conditions, scoring rubrics, artifact completeness, and expert involvement.

The unit of assessment also varies across studies. Some evaluations score input validity or code execution, whereas others assess completed workflows, agreement with reference properties, code quality as judged by experts, or case-study completion. AutoMat extends this line of evaluation beyond tool use by assessing claim-level reproduction[43]. It tests whether an agent can reconstruct the relevant computational procedure, execute it, and determine whether the resulting evidence supports a specific computational claim from a materials paper. These units cannot be compared directly: a valid input, a successful run, a value that matches a reference, and a reproduced computational claim support different types of conclusions. Evaluations should therefore specify their unit of assessment before comparing scores.

Across these forms of evaluation, automated methods usually assess local, formalized signals[40,44]. Syntax validity, required-file generation, startup success, completion status, convergence flags, and agreement with predefined reference values can be checked at scale, supporting reproducible evaluation. These checks are necessary, but they do not cover all aspects of scientific assessment. AutoMat shows why this distinction matters in practice: a run can generate outputs yet still fail as a reproduction if the results do not support the target claim or if the calculation procedure deviates from the intended method[43]. For example, in the AUTOMAT-0007 task, the agent completed the workflow and achieved an accuracy of 0.8894, close to the reported value of approximately 0.89. However, the agent used an incorrect temperature feature and evaluated a different data subset; therefore, the apparent numerical agreement did not constitute a reproduction of the intended scientific claim. Expert or reference-based judgment is still needed when the claim depends on the choice of method, the physical adequacy of the model, reference-state selection, the reasonableness of parameter choices, or interpretation. Rubrics can make this judgment more reproducible[42,45,46], especially when they specify what evidence should be present and which errors matter most; however, they also encode task-specific disciplinary assumptions. Evaluation protocols should therefore state which components are checked automatically, which require expert or reference-based judgment, and which remain outside the benchmark.

Clear task definitions and scoring rules are still insufficient when the workflow record is incomplete. Success on individual checks or tool-use steps does not establish robustness across complete workflows, and local repair loops do not show whether recovery remains reliable in longer workflows, across changing software environments and task types, or when unexpected failures occur. Intermediate artifacts are therefore important: without procedures, files, logs, and failure records, it is difficult to determine whether a failure stems from agent behavior, missing evidence, unavailable files, or an underspecified task. AutoMat's failure analysis supports this concern: many failed reproductions were attributed to incomplete procedures, methodological deviations, or execution fragility, indicating process-level failures rather than simply incorrect final answers[43]. Related agent studies increasingly release portions of such records[20,22,47], but papers and repositories still often leave the process only partly visible. Benchmarks should therefore treat artifact completeness as part of the evaluation and state whether they assess a runnable artifact, a checked computational result, or an inspectable workflow record.

These visibility issues point back to the same evaluation problem: each form of evidence supports a different type of claim. Local component checks support claims about components; end-to-end case studies support claims about selected workflow episodes; structured benchmarks support claims about performance within a defined task distribution; automated checks support claims about formalized signals; expert rubrics support broader judgments of scientific adequacy when their criteria are explicit. Conflating these evidence types makes computational materials agents appear more reliable than their evaluations warrant. Keeping the evidence types distinct makes reported progress easier to interpret.

CONCLUSION AND OUTLOOK

In materials research, computation often connects theoretical ideas with experimental or design decisions. It maps structures, mechanisms, theoretical assumptions, and experimental questions onto quantities that can be compared, tested, or used for screening, while experimental constraints and observations determine which calculations are meaningful. In this context, computational materials agents extend language assistance into executable research workflows.

Computational materials agents can help make this connection operational by selecting a calculation route, preparing executable inputs, running domain-specific software, checking whether outputs are usable, and preserving the context needed for subsequent scientific decisions. Usable outputs can then inform those

decisions. A relaxed structure may be reused in a higher-level calculation; an adsorption energy may alter a screening decision; and a failed recovery may leave a branch underexplored. Agents must therefore make clear the basis of each computed value, the remaining uncertainty, and the actions the result can justify.

Reported systems already show that agents can carry a computational materials task through input generation, software execution, output parsing, and limited repair. To date, these capabilities have mostly been demonstrated in settings with well-specified tasks and software configurations. Progress toward routine research use requires benchmarks that go beyond showing that a single example can run and instead test whether an agent remains reliable across task families, software environments, materials systems, and failure modes. Agent systems must also preserve the calculation route, assumptions, checks, recovery history, and uncertainty needed for other researchers to interpret or reuse the result.

The broader value of computational materials agents will depend on whether they enable computational workflows that are traceable, verifiable, and reusable. Their outputs need to be evaluated against relevant scientific standards and integrated responsibly into materials design, hypothesis testing, or experimental decision-making. Under these conditions, executable agent workflows may become a practical part of everyday materials research.

DECLARATIONS

Authors' contributions

Contributed to the conception and design of the review: Hu, C.; Wang, Z.
Conducted the literature review and analysis and drafted the manuscript: Hu, C.
Supervised the work, contributed to interpretation, and revised the manuscript: Wang, Z.
Both authors read and approved the final manuscript.

Availability of data and materials

Not applicable.

AI and AI-assisted tools statement

Not applicable.

Financial support and sponsorship

This work was supported by the City University of Hong Kong Start-up Grant (No. 9020004), with additional support from the Hong Kong Institute of AI for Science (No. 9360163).

Conflicts of interest

Both authors declared that there are no conflicts of interest.

Ethical approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

REFERENCES

1. Yao, S.; Zhao, J.; Yu, D.; et al. ReAct: synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. https://openreview.net/forum?id=WE_vluYUL-X. (accessed 2026-09-01).
2. Schick, T.; Dwivedi-Yu, J.; Dessi, R.; et al. Toolformer: language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36*, New Orleans, Louisiana, USA, December 10-16, 2023; Neural Information Processing Systems Foundation, Inc. (NeurIPS): San Diego, California, USA, 2023; pp 68539-51. DOI

3. Curtarolo, S.; Hart, G. L. W.; Nardelli, M. B.; Mingo, N.; Sanvito, S.; Levy, O. The high-throughput highway to computational materials design. *Nat. Mater.* **2013**, *12*, 191-201. DOI PubMed
4. Lejaeghere, K.; Bihlmayer, G.; Björkman, T.; et al. Reproducibility in density functional theory calculations of solids. *Science* **2016**, *351*, aad3000. DOI PubMed
5. Liu, Z. K. Thermodynamics and its prediction and CALPHAD modeling: review, state of the art, and perspectives. *Calphad* **2023**, *82*, 102580. DOI
6. Meng, Z.; Hao, C.; Li, X. Bridging machine learning and water electrolysis: Concepts, methods, and perspectives. *Mater. Today. Energy.* **2026**, *57*, 102232. DOI
7. Van, M. H.; Verma, P.; Zhao, C.; Wu, X. A survey of AI for materials science: foundation models, LLM agents, datasets, and tools. *ACM. Comput. Surv.* **2026**, *58*, 1-37. DOI
8. Wei, J.; Yang, Y.; Zhang, X.; et al. From AI for science to agentic science: a survey on autonomous scientific discovery. *arXiv* **2025**, arXiv:2508.14111. Available online: https://doi.org/10.48550/arXiv.2508.14111 (accessed 1 September 2026).
9. Oliveira, O. N.; Christino, L.; Oliveira, M. C. F.; Paulovich, F. V. Artificial intelligence agents for materials sciences. *J. Chem. Inf. Model.* **2023**, *63*, 7605-9. DOI PubMed
10. Calderon, C. E.; Plata, J. J.; Toher, C.; et al. The AFLOW standard for high-throughput materials science calculations. *Comput. Mater. Sci.* **2015**, *108*, 233-8. DOI
11. Huber, S. P.; Bosoni, E.; Bercx, M.; et al. Common workflows for computing material properties using different quantum engines. *npj. Comput. Mater.* **2021**, *7*, 136. DOI
12. Mathew, K.; Montoya, J. H.; Faghaninia, A.; et al. Atomate: a high-level interface to generate, execute, and analyze computational materials science workflows. *Comput. Mater. Sci.* **2017**, *139*, 140-52. DOI
13. Yao, T.; Yang, Y.; Yan, Y.; et al. Knowledge-extractor: a self-evolving scientific framework for hydrogen energy research driven by AI agents. *AI. Agent.* **2025**, *1*, 7. DOI
14. Jia, S.; Zhang, C.; Fung, V. LLMatDesign: autonomous materials discovery with large language models. *arXiv* **2024**, arXiv:2406.13163. Available online: https://doi.org/10.48550/arXiv.2406.13163 (accessed 1 September 2026).
15. Ghafarollahi, A.; Buehler, M. J. Automating alloy design and discovery with physics-aware multimodal multiagent AI. *Proc. Natl. Acad. Sci. U. S. A.* **2025**, *122*, e2414074122. DOI PubMed PMC
16. Bran, A. M.; Cox, S.; Schilter, O.; Baldassari, C.; White, A. D.; Schwaller, P. Augmenting large language models with chemistry tools. *Nat. Mach. Intell.* **2024**, *6*, 525-35. DOI PubMed PMC
17. Zhang, D.; Jia, X.; Liu, H.; et al. Cloud synthesis: a global closed-loop feedback powered by autonomous AI-driven catalyst design agent. *AI. Agent.* **2025**, *1*, 2. DOI
18. Zhou, L.; Ling, H.; Yan, K.; et al. Toward greater autonomy in materials discovery agents: unifying planning, physics, and scientists. In *Transactions on Machine Learning Research*, 2026. https://openreview.net/forum?id=Cwq1U8tbWW. (accessed 2026-09-01).
19. Boiko, D. A.; Macknight, R.; Kline, B.; Gomes, G. Autonomous chemical research with large language models. *Nature* **2023**, *624*, 570-8. DOI PubMed PMC
20. Hu, Z.; Talit, K.; Wang, Z.; et al. TritonDFT: automating DFT with a multi-agent framework. *arXiv* **2026**, arXiv:2603.03372. Available online: https://doi.org/10.48550/arXiv.2603.03372 (accessed 1 September 2026).
21. Shi, Z.; A, H.; Shao, Y.; et al. MDAgent2: large language model for code generation and knowledge Q&A in molecular dynamics. *arXiv* **2026**, arXiv:2601.02075. Available online: https://doi.org/10.48550/arXiv.2601.02075 (accessed 1 September 2026).
22. Yang, F.; Evans, J. D. QUASAR: a universal autonomous system for atomistic simulation and a benchmark of its capabilities. *J. Chem. Inf. Model.* **2026**, *66*, 5911-8. DOI PubMed
23. Soleymanibrojeni, M.; Aydin, R.; Guedes-Sobrinho, D.; et al. GENIUS: an agentic AI framework for autonomous design and execution of simulation protocols. *Commun. Mater.* **2026**, *7*, 115. DOI
24. Ding, M.; Huang, C.; Hu, Y.; et al. Automating computational chemistry workflows via OpenClaw and domain-specific skills. *J. Chem. Theory. Comput.* **2026**, *22*, 5919-29. DOI PubMed
25. Zhang, C.; Yakobson, B. I. MatClaw: an autonomous code-first LLM agent for End-to-End materials exploration. *arXiv* **2026**, arXiv:2604.02688. Available online: https://doi.org/10.48550/arXiv.2604.02688 (accessed 1 September 2026).
26. Gupta, T.; Zaki, M.; Krishnan, N. M. A. Mausam. MatSciBERT: a materials domain language model for text mining and information extraction. *npj. Comput. Mater.* **2022**, *8*, 102. DOI
27. Ahlawat, D.; Mishra, V.; Singh, S.; et al. A family of large language models for materials research with insights into model adaptability in continued pretraining. *Nat. Mach. Intell.* **2026**, *8*, 435-48. DOI
28. Tang, Y.; Xu, W.; Cao, J.; et al. A multimodal large language model for materials science. *Nat. Mach. Intell.* **2026**, *8*, 588-601. DOI
29. Wu, Q.; Bansal, G.; Zhang, J.; et al. AutoGen: enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. https://openreview.net/forum?id=BAakY1hNKS. (accessed 2026-09-01).

30. Vriza, A.; Kornu, U.; Koneru, A.; Chan, H.; Sankaranarayanan, S. K. R. S. Multi-agentic AI framework for end-to-end atomistic simulations. *Digit. Discov.* **2026**, *5*, 440-52. DOI
31. Ferrag, M. A.; Tihanyi, N.; Debbah, M. From LLM reasoning to autonomous AI agents: a comprehensive review. *arXiv* **2026**, arXiv:2504.19678. Available online: https://doi.org/10.48550/arXiv.2504.19678 (accessed 1 September 2026).
32. Ramos, M. C.; Collison, C. J.; White, A. D. A review of large language models and autonomous agents in chemistry. *Chem. Sci.* **2025**, *16*, 2514-72. DOI PubMed PMC
33. Wang, Z.; Huang, H.; Zhao, H.; et al. DREAMS: density functional theory based research engine for agentic materials simulation. *arXiv* **2025**, arXiv:2507.14267. Available online: https://doi.org/10.48550/arXiv.2507.14267 (accessed 1 September 2026).
34. Yang, P.; Zhang, Z.; Li, Y.; et al. AutoDFT: A closed-loop multi-agent framework for autonomous DFT calculations. *arXiv* **2026**, arXiv:2605.26179. Available online: https://doi.org/10.48550/arXiv.2605.26179 (accessed 1 September 2026).
35. Liu, G.; Yang, S.; Zhong, Y. Masgent: an AI-assisted materials simulation agent. *Digit. Discov.* **2026**, *5*, 2151-71. DOI
36. Yu, B.; Baker, F. N.; Chen, Z.; et al. Tooling or not tooling? The impact of tools on language agents for chemistry problem solving. In *Findings of the Association for Computational Linguistics: NAACL 2025*, Albuquerque, New Mexico, March 2025; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp 7635-55. DOI PubMed PMC
37. Zaki, M.; Jayadeva; Mausam; Krishnan, N. M. A. MaScQA: investigating materials science knowledge of large language models. *Digit. Discov.* **2024**, *3*, 313-27. DOI
38. Cheung, J. J.; Shen, S.; Zhuang, Y.; Li, Y.; Ramprasad, R.; Zhang, C. MSQA: benchmarking LLMs on graduate-level materials science reasoning and knowledge. *arXiv* **2025**, arXiv:2505.23982. Available online: https://doi.org/10.48550/arXiv.2505.23982 (accessed 1 September 2026).
39. Bajan, C.; Lambard, G. Exploring the expertise of large language models in materials science and metallurgical engineering. *Digit. Discov.* **2025**, *4*, 500-12. DOI
40. Liu, S.; Hu, B.; Ye, B.; et al. MatTools: benchmarking large language models for materials science tools. *arXiv* **2025**, arXiv:2505.10852. Available online: https://doi.org/10.48550/arXiv.2505.10852 (accessed 1 September 2026).
41. Shi, Z.; Xin, C.; Huo, T.; et al. A fine-tuned large language model based molecular dynamics agent for code generation to obtain material thermodynamic parameters. *Sci. Rep.* **2025**, *15*, 10295. DOI PubMed PMC
42. Kumar, S. G. H.; Zou, Y.; Wang, A.; et al. El Agente Sólido: a new age(nt) for solid state simulations. *arXiv* **2026**, arXiv:2602.17886. Available online: https://doi.org/10.48550/arXiv.2602.17886 (accessed 1 September 2026).
43. Huang, Z.; Cao, Y.; Shargh, A. K.; et al. Can coding agents reproduce findings in computational materials science? *arXiv* **2026**, arXiv:2605.00803. Available online: https://doi.org/10.48550/arXiv.2605.00803 (accessed 1 September 2026).
44. Holbrook, E.; Verduzco, J. C.; Strachan, A. Evaluating LLM-generated code for domain-specific languages: molecular dynamics with LAMMPS. *Comput. Mater. Sci.* **2026**, *272*, 114839. DOI
45. Zhang, Z.; Yin, A.; Baweja, A.; et al. El agente forjador: task-driven agent generation for quantum simulation. In *AI4X - Accelerate Conference 2026*, 2026. https://openreview.net/forum?id=7aXeu0hHo5. (accessed 2026-09-01).
46. Liang, P.; Bommasani, R.; Lee, T.; et al. Holistic evaluation of language models. In *Transactions on Machine Learning Research*, 2023. https://openreview.net/forum?id=iO4LZibEqW. (accessed 2026-09-01).
47. Pham, T. D.; Tanikanti, A.; Keçeli, M. ChemGraph as an agentic framework for computational chemistry workflows. *Commun. Chem.* **2026**, *9*, 33. DOI PubMed PMC