SkillPuzzler: A Self-Evolving Agentic Framework for Materials and Chemistry Research with Minimal Reliance on Predefined Tools

Xu Huang
Department of Materials Science and Engineering, University of California, Berkeley
Materials Sciences Division, Lawrence Berkeley National Laboratory, California, United States
xu_huang@berkeley.edu

Junwu Chen*
Laboratory of Artificial Chemical Intelligence (LIAC)
École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
junwu.chen@epfl.ch

Philippe Schwaller†
Laboratory of Artificial Chemical Intelligence (LIAC)
École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
philippe.schwaller@epfl.ch

Gerbrand Ceder‡
Department of Materials Science and Engineering, University of California, Berkeley
Materials Sciences Division, Lawrence Berkeley National Laboratory, California, United States
gceder@berkeley.edu

Abstract

The prevailing "large language model (LLM) + tool-use" paradigm relies on handcrafted tool interfaces, which constrain an agent's ability to solve complex problems and hinder the adoption of agents as scientific copilots across the broader research community. We advocate a dynamic, scalable "LLM + skill-acquisition" paradigm and present SKILLPUZZLER as one concrete instantiation of it. SKILLPUZZLER combines only 4 specialized agents with 15 general-purpose tools, yet exhibits self-evolution while tackling diverse research tasks in materials science and chemistry. Its behavior is driven by prompt-encoded mindsets tailored to our customized Model Context Protocol (MCP) servers. SKILLPUZZLER autonomously acquires new skills by learning and adapting knowledge into self-defined tools for problem-solving. It achieves 96.7% accuracy with OpenAI O3 model on our 74-task benchmark and outperforms two baselines by a wide margin, demonstrating the robustness and effectiveness of its self-evolution mechanism.

*Corresponding author.
†Corresponding author.
‡Corresponding author.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Science workshop (NeurIPS 2025).

1 Introduction

Large language models (LLMs) stand apart from mainstream machine learning models because their mastery of natural language confers a rudimentary capacity for reasoning and enables the "LLM + tool-use" paradigm [1, 2]. The capability to use tools further augments LLM agents' performance on scientific tasks [3, 4, 5, 6]. However, most reported LLM agents rely heavily on customized workflows and predefined tool interfaces, which are crafted by human experts for specific applications [7, 8, 9, 10]. This dependency constrains the agents' capability range and impedes extension to complex tasks requiring new tools or software. Moreover, this paradigm is hard to scale for the broader community, since every specialized domain—especially in science—requires experts to curate new tool catalogs, write detailed usage notes and craft bespoke prompts [7, 11, 12, 13, 14, 15].

In practical research, human scientists continuously learn new tools based on their needs, and iteratively refine their procedures to master related skills. To fully unleash an agentic system's potential, we must progress from human-defined behavior toward human-like intelligence rooted in autonomous skill acquisition. We therefore advocate the dynamic and scalable "LLM + skill-acquisition" paradigm, instantiated in our SKILLPUZZLER, a novel self-evolving agentic framework for materials and chemistry research with minimal reliance on predefined tools. SKILLPUZZLER employs reusable agentic routines encoded as mindsets to explore, leverage and update tools on its own (crafting custom puzzle pieces), and fuse them into complex problem-solving procedures (completing the puzzle). These mindsets not only help to solve various tasks but also forge richer skills, yielding a sophisticated, customizable toolset that can be shared with other agents or human researchers.

We demonstrate the framework on a 74-problem benchmark covering diverse real-world simulation and data-related workflows in materials science and chemistry. When coupled with the OpenAI O3 model, SKILLPUZZLER solves 96.7% of all tasks, including 94.3% of the more demanding Level 1 questions. The superior performance of SKILLPUZZLER against two baselines further validates its real-time self-evolving capability through self-learning and iterative refinement.

2 Methods

2.1 Multi-agent system architecture

Using the OpenAI Agents SDK, we designed SKILLPUZZLER to orchestrate four specialized agents that collaboratively solve complex materials- and chemistry-related tasks while autonomously acquiring new tools and skills. Figure 1(a) illustrates "LLM + skill-acquisition" paradigm of SKILLPUZZLER versus the current "LLM + tool-use" paradigm. Figure 1(b) describes SKILLPUZZLER's 4-step sequential workflow with conditional parallel debugging, where each agent has distinct responsibilities. Step 1 involves solution research where **Solution Researcher** receives the user query, conducts comprehensive analysis using web searches, identifies required software, extracts code examples from URLs, and generates initial code solutions. **Code Agent** verifies that the solution meets basic requirements before execution, installs software dependencies and executes the code, determining if debugging is needed based on execution success and result accuracy. Step 3 is conditional parallel debugging, where three **Debug Agent** instances run simultaneously if the initial execution fails, each employing different debugging trajectories to fix the code. The parallel debugging approach increases the likelihood of successful problem resolution by exploring multiple solution strategies simultaneously. Step 4 involves output processing where **Output Processor Agent** evaluates all available results, selects the best solution, and extracts the final answer in the exact format required by the user for further automatic accuracy evaluation. We defined the output format for each agent to ensure more reliable data flow between agents and enable easier automated evaluation of the agentic system performance.

2.2 MCP server infrastructure and tools

Our system leverages three specialized Model Context Protocol (MCP) servers to provide comprehensive capabilities for research, code execution, and workspace management, including one third-party API-based server and two custom-developed servers.

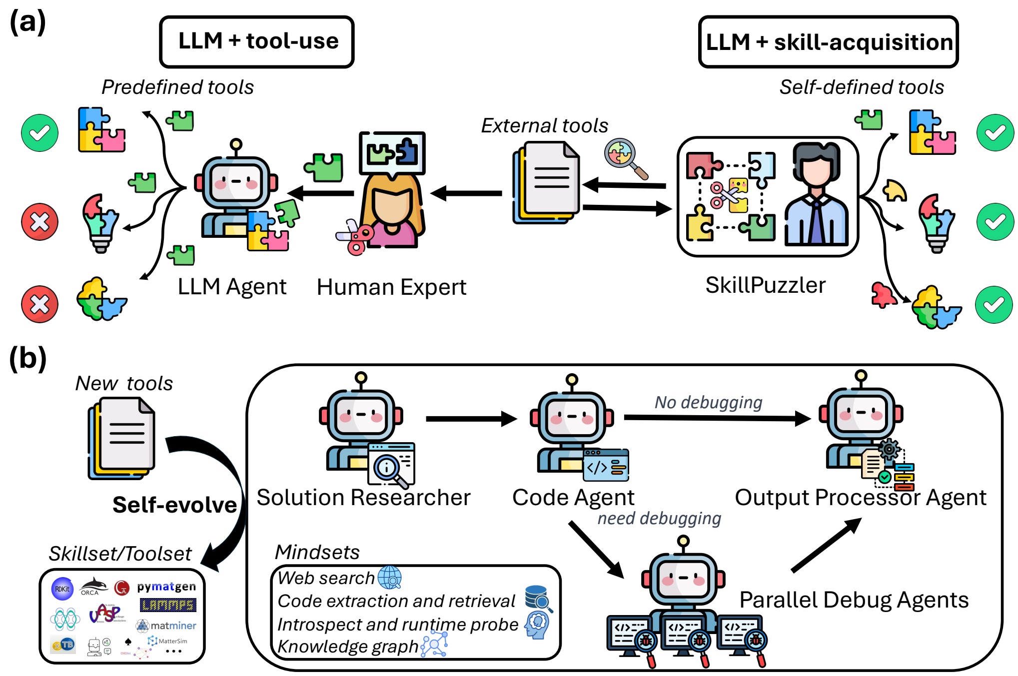

Figure 1: **"LLM + skill-acquisition" paradigm and self-evolving agentic framework**. **(a)** Schematic illustration of dynamic and scalable "LLM + skill-acquisition" paradigm versus the current "LLM + tool-use" paradigm for solving scientific problems. **(b)** SKILLPUZZLER's self-evolving architecture: guided by reusable mindsets, the system acquires new tool sets and hones new skill sets in materials science and chemistry.

2.2.1 Tavily server

We use the Tavily Search Engine `tavily-search` through the Tavily MCP server, which provides real-time web search capabilities and enables agents to discover relevant documentation, code examples, and implementation resources from the web.

2.2.2 Research server

Research Server implements a sophisticated code intelligence and knowledge discovery system that provides comprehensive capabilities for web code extraction, Agentic RAG (Retrieval-Augmented Generation), code introspection, runtime probing, and knowledge graph construction and exploration.

Code extraction and retrieval `extract_code_from_url` Implements intelligent web crawling with multi-strategy extraction capabilities and caching mechanisms using Supabase database storage, which is a Postgres development platform. The tool supports both single-page extraction and smart crawling strategies with intelligent fallback mechanisms. It automatically detects content types and applies specialized extractors for different documentation systems, including Jupyter notebooks, ReadTheDocs/Sphinx and MkDocs documentation, raw code files from repositories, and markdown content with intelligent parsing of fenced code blocks and command examples. It also extracts relevant text before and after code blocks using intelligent paragraph boundary detection, providing semantic context for code understanding. Optional LLM summary for the extracted code is also available. `retrieve_extracted_code` Implements vector-based similarity search through extracted code blocks using embeddings with configurable match count.

Code analysis `quick_introspect` Implements static-first analysis using Jedi [16] for import resolution and error diagnosis without runtime execution. The tool provides comprehensive package, class, method, and function discovery with fuzzy matching capabilities. `runtime_probe_snippet` Provides code snippets to enable runtime key and attribute probing. When KeyError or AttributeError occurs, the tool shows available keys/attributes, object type information, and similar name suggestions to help resolve access issues. `parse_local_package` Implements direct Neo4j knowledge graph construction from local Python packages using AST (Abstract Syntax Tree). The tool extracts classes, methods, functions, attributes, and import relationships with detailed parameter information including type hints and default values. `query_knowledge_graph` Provides advanced querying capabilities for exploring repository structures, class hierarchies, method signatures, and code relationships in the knowledge graph using Cypher query language.

2.2.3 Workspace server

Workspace Server provides a multi-environment code execution and management system. It supports conda, virtualenv, and UV environments with cross-platform compatibility for Windows and Unix-like systems. The server prevents access to the benchmark directory to avoid solution leakage and enforces security boundaries by confining all operations to the designated project root scope.

Package management `check_installed_packages` Lists all installed packages in the current Python environment with version information and package count. `install_dependencies` Installs Python packages based on environment configuration. `check_package_version` Performs detailed package analysis including version detection, installation path resolution, and module location identification. The tool takes package names as input and handles name variations (hyphens, underscores, dots).

Code execution `execute_code` Executes Python code in the configured environment. The tool saves code to temporary files in the code storage directory and executes with detailed output capture including stdout and stderr. `execute_shell_command` Executes shell commands in the code storage directory. `create_and_execute_script` Creates and executes shell scripts in the code storage directory.

File operations `read_file` Reads content from any text file. `save_file` Saves content for later reuse in the designated directory.

2.3 Agent-tool integration and workflow details

Each agent in SkillPuzzler leverages specific MCP servers and tools to accomplish their specialized tasks within the collaborative workflow.

Solution Researcher initiates the workflow by conducting comprehensive research to generate initial code solutions for the user query. It connects to two MCP servers: Tavily server and Research server. The agent is designed to perform a systematic research process that typically involves: understanding the request, searching for relevant information using `tavily-search` with advanced search parameters, extracting code examples from identified URLs using `extract_code_from_url`, reviewing and understanding additional requirements, and synthesizing the final solution. The agent uses `retrieve_extracted_code` for vector-based similarity search when the extracted content is overwhelming, and optionally employs `quick_introspect` to confirm exact import paths and class/method/function names. For complex problems without explicit step-by-step instructions, it is inspired to plan and decompose tasks, select appropriate tools to achieve objectives, and learn how to use them effectively. The output includes the original user query, required packages list, and complete code solution.

Code Agent receives the research results and executes the code solution. It connects to three MCP servers: Tavily server, Research server, and Workspace server. It is designed to perform a 5-step execution process: analyzing input, verifying solution requirements using research tools if needed, managing packages through `check_installed_packages` and `install_dependencies`, executing code using `execute_code`, and evaluating results to determine if debugging is needed.

Debug Agent instances run in parallel when the initial solution fails. Each agent connects to three MCP servers: Tavily server, Research server, and Workspace server. The agents use multiple debugging approaches after analyzing the error. They employ `execute_code` to test fixes: **Direct Fix** for obvious errors, **Introspection/Probe Fix** using `quick_introspect` and `runtime_probe_snippet` for Python package symbol resolution and runtime key/attribute access errors, **Knowledge Graph Fix** using `parse_local_package` and `query_knowledge_graph` which serves as the global exploration layer when the fast, error-site Introspection/Probe Fix doesn't resolve the issue, **Local Package**

Fix using `execute_shell_command`, `create_and_execute_script`, and `read_file` for examining specific files such as package-related code and output files containing results, and **Research Fix** using `tavily-search`, `extract_code_from_url`, and `retrieve_extracted_code` for finding documentation and solutions, especially for external program issues. The agents can also employ **Diagnostic Fix**, and **Result Processing Fix** to modify code for producing processable results. Each agent is asked to conduct up to 30 debugging attempts, combining different approaches to maximize the likelihood of successful problem resolution.

Output Processor Agent evaluates all available results and processes the output to required format. When debugging is needed, the agent receives three debug results and evaluates each based on successful execution, presence of required data, and quality of results. It then selects the best result and extracts the final answer in the exact format required by the user. When no debugging is needed, it processes the successful execution result directly. The output includes original user query, success status, final code, execution results, and processed output, where processed output serves as the key field for automated correctness evaluation.

3 Experiment

3.1 Experimental setting

3.1.1 Benchmarks

To comprehensively evaluate this agentic system's performance on solving real-world computational tasks in materials science and chemistry, we constructed a diverse benchmark comprising 74 total problems (37 Level 0 and 37 Level 1 tasks). Level 0 tasks tend to simulate computational scientists who understand core functions to use and provide some guidance but prefer not to handle implementation details and potential errors, while Level 1 tasks represent experimental scientists who know their research objectives but lack computational expertise and require more autonomous problem-solving from the system. The benchmark spans two principal categories: data-oriented tasks and computation-oriented tasks. Data tasks comprise (i) data-retrieval problems from resources such as the Materials Project [17], the Inorganic Crystal Structure Database (ICSD) [18], Matminer [19], MP-Contribs [20], the Materials Data Facility [21], OQMD[22], OBELiX [23] and GMAE [24, 25, 26]; (ii) data-analysis problems that use packages and databases including pymatgen [27, 28, 29, 30, 31], Matminer, SMACT [32], the Materials Project and OBELiX; (iii) data-management problems with pymatgen-db [33] and MongoDB [34]; and (iv) data-processing problems that rely on RDKit [35], Matminer, Magpie [36], Robocrystallographer [37], pymatgen, enumlib [38, 39, 40, 41], spglib [42], the Materials Project and OBELiX. Computation tasks include (v) simulation problems with xTB [43], ORCA [44], ASE [45], LAMMPS [46] and CHGNet [47, 48, 49], together with (vi) problems involving specialized models and toolkits such as CHGNet, MACE [50] and mlip [51]. These tasks range from straightforward code migration from documentation with simple parameter changes to non-documented questions that require code exploration, newly released packages not covered in most models' training data, and packages with out-of-sync documentation still available online that may cause confusion.

Here we show a representative benchmark example: the Level 1 problem is: Write code for querying the formation energy and the bulk modulus (shown in the order of voigt, reuss, and vrh values in an array) of Li_6FeN_4 (Materials Project ID: mp-1029739). The Level 0 variant is identical but appends the explicit instruction "using materials.summary.search()".

3.1.2 Baselines

To further explore SKILLPUZZLER's effectiveness, we benchmarked it against two baselines. In the **Native baseline** which is designed to reveal model's native ability without any self-evolution, Solution Researcher produces code without access to Tavily Server or Research Server, Code Agent (connected only to Workspace Server) executes it once, and Output Processor Agent retains its processing ability and returns the processed result. In the **Search&Debug baseline**, Solution Researcher is encouraged to call Tavily search and Code Agent (connected only to Workspace Server) is required to debug if needed, yet neither component receives our specialized prompts or Research Server.

3.2 Experimental results

We conducted 3 independent repetitions for each benchmark question (222 questions in total) using separate Python processes to ensure complete isolation. Prior to each experiment, we cleared the Supabase database and removed temporary files to prevent cross-contamination. In accuracy calculation, we excluded workflow failures (such as MCP tool timeouts).

Figure 2 compares the performance of SKILLPUZZLER with the Native baseline and the Search&Debug baseline across 7 OpenAI models. Among the models, O3 attains the highest performance (96.7% overall accuracy with SKILLPUZZLER), whereas GPT-4.1 Mini exhibits the lowest performance (18.1% overall accuracy under the Native baseline). Across all settings, Level 0 accuracy consistently exceeds Level 1 accuracy, reflecting the expected increase in task difficulty. Moreover, model performance follows a clear upward trend from the Native baseline, which is designed to measure the inherent answering capability of LLMs, to the Search&Debug baseline, which leverages Tavily Search Engine to access external web information and Workspace Server for debugging, and further to SKILLPUZZLER, which integrates Tavily Server, Research Server, and Workspace Server under carefully designed prompting. These progressive improvements highlight the agent's capacity for self-evolution and, in particular, demonstrate SKILLPUZZLER's effective and robust skill acquisition in real time.

4 Conclusion

In this work, we have introduced the "LLM + skill-acquisition" paradigm and demonstrated its feasibility through SKILLPUZZLER, a multi-agent framework for materials science and chemistry problems. With only four specialized agents and minimal reliance on predefined tools, SKILLPUZZLER is able to self-evolve through real-time learning and iterative refinement.

Acknowledgments and Disclosure of Funding

This work was primarily funded and intellectually led by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Materials Sciences and Engineering Division under Contract No. DE-AC0205CH11231 (Materials Project program KC23MP). The work was also supported by the computational resources provided by the Extreme Science and Engineering Discovery Environment (XSEDE), supported by National Science Foundation grant number ACI1053575; the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory; and the Swift Cluster resource provided by the National Renewable Energy Laboratory (NREL). J.C. and P.S. acknowledge support from the NCCR Catalysis (grant number 225147), a National Centre of Competence in Research funded by the Swiss National Science Foundation. We sincerely thank all the researchers and developers who created and maintained the datasets, packages, and software tools utilized in this work.

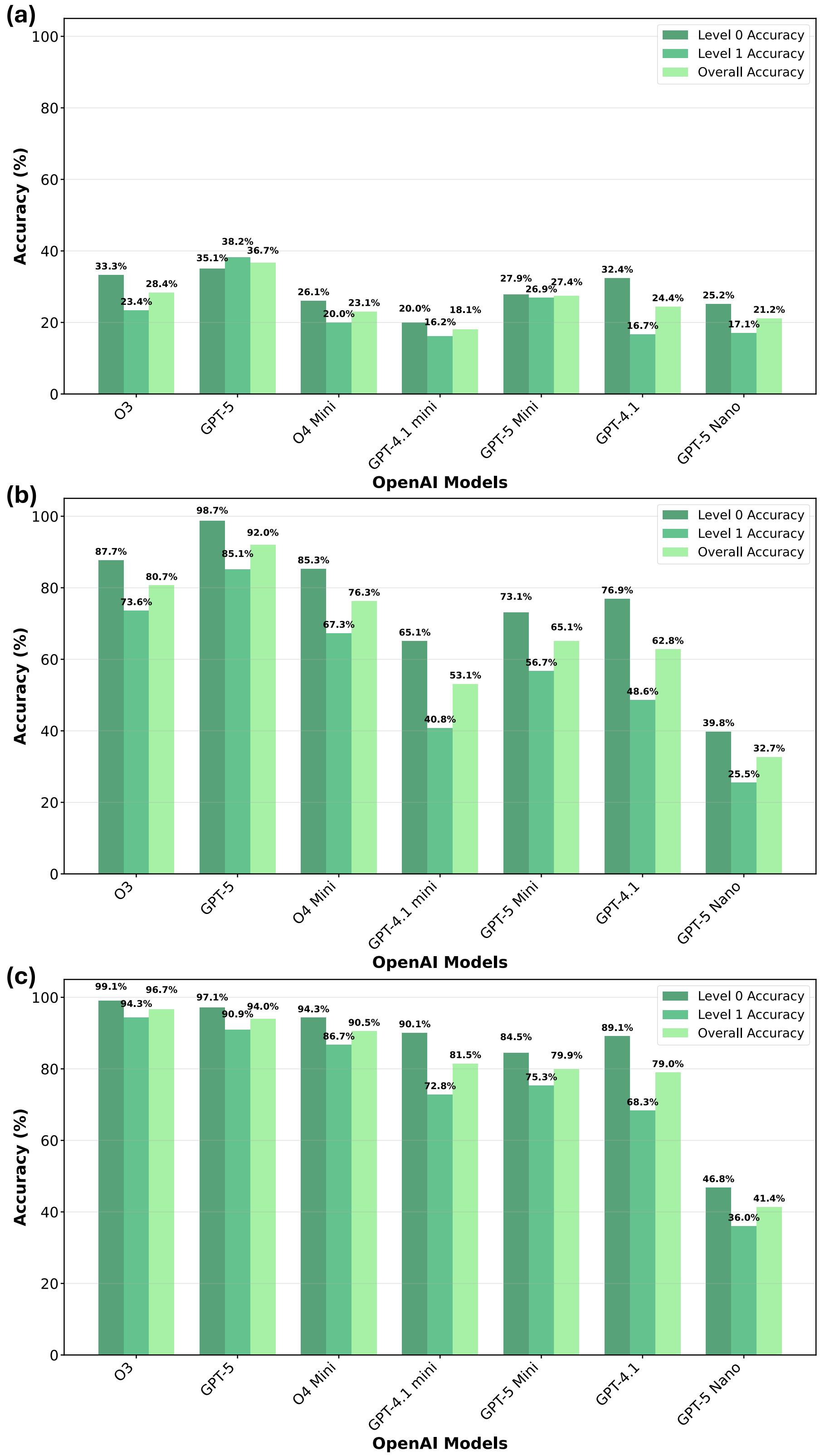

Figure 2: **Comparison of SkillPuzzler with baseline methods across OpenAI models**. **(a)** Native baseline. **(b)** Search&Debug baseline. **(c)** SkillPuzzler. The x-axis lists 7 OpenAI models, while the y-axis denotes accuracy (%). Each model is represented by three bars corresponding to Level 0 accuracy, Level 1 accuracy, and overall accuracy.

References

[1] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[2] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

[3] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.

[4] Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

[5] Adib Bazgir, Yuwen Zhang, et al. Matagent: A human-in-the-loop multi-agent llm framework for accelerating the material science discovery cycle. In *AI for Accelerated Materials Design-ICLR 2025*, 2025.

[6] Keyan Ding, Jing Yu, Junjie Huang, Yuchen Yang, Qiang Zhang, and Huajun Chen. Scitoolagent: a knowledge-graph-driven scientific agent for multitool integration. *Nature Computational Science*, pages 1–11, 2025.

[7] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, et al. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv preprint arXiv:2505.20286*, 2025.

[8] Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025.

[9] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[10] Xiang Fei, Xiawu Zheng, and Hao Feng. Mcp-zero: Active tool discovery for autonomous llm agents. *arXiv preprint arXiv:2506.01056*, 2025.

[11] Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelović, and Jakob Nikolas Kather. Llm agents making agent tools. *arXiv preprint arXiv:2502.11705*, 2025.

[12] Jiabin Tang, Tianyu Fan, and Chao Huang. Autoagent: A fully-automated and zero-code framework for llm agents. *arXiv preprint arXiv:2502.05957*, 2025.

[13] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[14] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

[15] Ruofan Jin, Zaixi Zhang, Mengdi Wang, and Le Cong. Stella: Self-evolving llm agent for biomedical research. *arXiv preprint arXiv:2507.02004*, 2025.

[16] Dave Halter et al. Jedi: an awesome auto-completion, static analysis and refactoring library for python. *Online document https://jedi. readthedocs. io*, 2022.

[17] Matthew K Horton, Patrick Huck, Ruo Xi Yang, Jason M Munro, Shyam Dwaraknath, Alex M Ganose, Ryan S Kingsbury, Mingjian Wen, Jimmy X Shen, Tyler S Mathis, et al. Accelerated data-driven materials science with the materials project. *Nature Materials*, pages 1–11, 2025.

[18] G Bergerhoff, R Hundt, R Sievers, and I D Brown. The inorganic crystal structure data base. *Journal of Chemical Information and Computer Sciences*, 23(2):66–69, 1983.

[19] Logan Ward, Alexander Dunn, Alireza Faghaninia, Nils ER Zimmermann, Saurabh Bajaj, Qi Wang, Joseph Montoya, Jiming Chen, Kyle Bystrom, Maxwell Dylla, et al. Matminer: An open source toolkit for materials data mining. *Computational Materials Science*, 152:60–69, 2018.

[20] Patrick Huck, Dan Gunter, Shreyas Cholia, Donald Winston, AT N'Diaye, and Kristin Persson. User applications driven by the community contribution framework mpcontribs in the materials project. *Concurrency and Computation: Practice and Experience*, 28(7):1982–1993, 2016.

[21] Ben Blaiszik, Kyle Chard, Jim Pruyne, Rachana Ananthakrishnan, Steven Tuecke, and Ian Foster. The materials data facility: data services to advance materials science research. *Jom*, 68(8):2045–2052, 2016.

[22] James E Saal, Scott Kirklin, Muratahan Aykol, Bryce Meredig, and Christopher Wolverton. Materials design and discovery with high-throughput density functional theory: the open quantum materials database (oqmd). *Jom*, 65:1501–1509, 2013.

[23] Félix Therrien, Jamal Abou Haibeh, Divya Sharma, Rhiannon Hendley, Leah Wairimu Mungai, Sun Sun, Alain Tchagang, Jiang Su, Samuel Huberman, Yoshua Bengio, et al. Obelix: A curated dataset of crystal structures and experimentally measured ionic conductivities for lithium solid-state electrolytes. *arXiv preprint arXiv:2502.14234*, 2025.

[24] Junwu Chen, Xu Huang, Cheng Hua, Yulian He, and Philippe Schwaller. A multi-modal transformer for predicting global minimum adsorption energy. *Nature Communications*, 16(1):3232, 2025.

[25] Junwu Chen, Xu Huang, Cheng Hua, Yulian He, and Philippe Schwaller. Adsgt: Graph transformer for predicting global minimum adsorption energy. In *NeurIPS 2023 AI for Science Workshop*, 2023.

[26] Junwu Chen, Xu Huang, Cheng Hua, Yulian He, and Philippe Schwaller. AdsMT: Multi-modal transformer for predicting global minimum adsorption energy. `https://doi.org/10.6084/m9.figshare.25966573.v1`, 2024. Figshare.

[27] Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L Chevrier, Kristin A Persson, and Gerbrand Ceder. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319, 2013.

[28] William D Richards, Lincoln J Miara, Yan Wang, Jae Chul Kim, and Gerbrand Ceder. Interface stability in solid-state batteries. *Chemistry of Materials*, 28(1):266–273, 2016.

[29] Yihan Xiao, Yan Wang, Shou-Hang Bo, Jae Chul Kim, Lincoln J Miara, and Gerbrand Ceder. Understanding interface stability in solid-state batteries. *Nature Reviews Materials*, 5(2):105–126, 2020.

[30] Michael W Gaultois, Taylor D Sparks, Christopher KH Borg, Ram Seshadri, William D Bonificio, and David R Clarke. Data-driven review of thermoelectric materials: performance and resource considerations. *Chemistry of Materials*, 25(15):2911–2920, 2013.

[31] Christopher J Bartel, Amalie Trewartha, Qi Wang, Alexander Dunn, Anubhav Jain, and Gerbrand Ceder. A critical examination of compound stability predictions from machine-learned formation energies. *npj computational materials*, 6(1):97, 2020.

[32] Daniel W Davies, Keith T Butler, Adam J Jackson, Jonathan M Skelton, Kazuki Morita, and Aron Walsh. Smact: Semiconducting materials by analogy and chemical theory. *Journal of Open Source Software*, 4(38):1361, 2019.

[33] Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, et al. Commentary: The materials project: A materials genome approach to accelerating materials innovation. *APL materials*, 1(1), 2013.

[34] Shannon Bradshaw, Eoin Brazil, and Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. " O'Reilly Media, Inc.", 2019.

[35] Greg Landrum. Rdkit documentation. *Release*, 1(1-79):4, 2013.

[36] Logan Ward, Ankit Agrawal, Alok Choudhary, and Christopher Wolverton. A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Computational Materials*, 2(1):1–7, 2016.

[37] Alex M Ganose and Anubhav Jain. Robocrystallographer: automated crystal structure text descriptions and analysis. *MRS Communications*, 9(3):874–881, 2019.

[38] Gus LW Hart and Rodney W Forcade. Algorithm for generating derivative structures. *Physical Review B—Condensed Matter and Materials Physics*, 77(22):224115, 2008.

[39] Gus LW Hart and Rodney W Forcade. Generating derivative structures from multilattices: Algorithm and application to hcp alloys. *Physical Review B—Condensed Matter and Materials Physics*, 80(1):014120, 2009.

[40] Gus LW Hart, Lance J Nelson, and Rodney W Forcade. Generating derivative structures at a fixed concentration. *Computational Materials Science*, 59:101–107, 2012.

[41] Wiley S Morgan, Gus LW Hart, and Rodney W Forcade. Generating derivative superstructures for systems with high configurational freedom. *Computational Materials Science*, 136:144–149, 2017.

[42] Atsushi Togo, Kohei Shinohara, and Isao Tanaka. Spglib: a software library for crystal symmetry search. *Science and Technology of Advanced Materials: Methods*, 4(1):2384822, 2024.

[43] Christoph Bannwarth, Sebastian Ehlert, and Stefan Grimme. Gfn2-xtb—an accurate and broadly parametrized self-consistent tight-binding quantum chemical method with multipole electrostatics and density-dependent dispersion contributions. *Journal of chemical theory and computation*, 15(3):1652–1671, 2019.

[44] Frank Neese. The orca program system. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(1):73–78, 2012.

[45] Ask Hjorth Larsen, Jens Jørgen Mortensen, Jakob Blomqvist, Ivano E Castelli, Rune Christensen, Marcin Dułak, Jesper Friis, Michael N Groves, Bjørk Hammer, Cory Hargus, et al. The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017.

[46] Aidan P Thompson, H Metin Aktulga, Richard Berger, Dan S Bolintineanu, W Michael Brown, Paul S Crozier, Pieter J In't Veld, Axel Kohlmeyer, Stan G Moore, Trung Dac Nguyen, et al. Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer physics communications*, 271:108171, 2022.

[47] Bowen Deng, Peichen Zhong, KyuJung Jun, Janosh Riebesell, Kevin Han, Christopher J Bartel, and Gerbrand Ceder. Chgnet as a pretrained universal neural network potential for charge-informed atomistic modelling. *Nature Machine Intelligence*, 5(9):1031–1041, 2023.

[48] Xu Huang, Bowen Deng, Peichen Zhong, Aaron D Kaplan, Kristin A Persson, and Gerbrand Ceder. Cross-functional transferability in foundation machine learning interatomic potentials. *npj Computational Materials*, 11(1):313, 2025.

[49] Xu Huang, Bowen Deng, Peichen Zhong, Aaron Kaplan, Kristin Persson, and Gerbrand Ceder. Materials Project Trajectory Dataset of r2SCAN (MP-r2SCAN). `https://doi.org/10.6084/m9.figshare.28245650.v2`, 4 2025. Figshare.

[50] Ilyes Batatia, David P Kovacs, Gregor Simm, Christoph Ortner, and Gábor Csányi. Mace: Higher order equivariant message passing neural networks for fast and accurate force fields. *Advances in neural information processing systems*, 35:11423–11436, 2022.

[51] Christoph Brunken, Olivier Peltre, Heloise Chomet, Lucien Walewski, Manus McAuliffe, Valentin Heyraud, Solal Attias, Martin Maarand, Yessine Khanfir, Edan Toledo, et al. Machine learning interatomic potentials: library for efficient training, model development and simulation of molecular systems. *arXiv preprint arXiv:2505.22397*, 2025.