Framing Writing as Code Generation: An Agentic LaTeX IDE for Scientific Manuscript Composition

Zhisheng Tang[1], Mayank Kejriwal[1]

[1]GRAIL, USA
https://grailai.io

Abstract

While manuscript preparation has traditionally been viewed as distinct from code development, both scholarly writing and software engineering involve translating ideas with relatively prespecified structures and norms into *formal* specifications. In this paper, we argue that narrative scholarly writing—particularly scientific manuscript generation—should be reframed as a form of code generation. By arguing that LaTeX is fundamentally a programming language rather than merely a document formatting tool, we demonstrate how AI-assisted generation can produce high-quality manuscripts, figures, references, and methodological descriptions from specifications. We present the design and capabilities of the GRAIL platform, a specialized agentic LaTeX editor that enables this code generation paradigm. Preliminary evaluation on a deployed system demonstrates significant adoption, processing 13,910 requests from 332 unique users, with 119 users (35.8% of the user base) making more than 10 requests and accumulated engagement of 921.5 hours across 2,603 sessions. This work positions manuscript preparation as the final critical stage in the autonomous science pipeline, where research outputs are automatically documented and disseminated. We show how this perspective opens new possibilities for integration with code analysis, figure generation, and reference management systems.

Keywords

code generation, manuscript generation, LaTeX, autonomous science, AI-assisted writing

1. Introduction

The preparation of scientific manuscripts has long occupied a liminal space between creative writing and technical specification. Researchers conduct experiments, analyze data, and develop software, but the translation of these rigorous activities into prose has remained largely separate from the computational frameworks that generated the research itself. This separation reflects a historical artifact: manuscript preparation predates modern computational science and has retained its cultural identity as a primarily linguistic task. Recent advances in large language models (LLMs) and autonomous research systems have begun to challenge this separation [1, 2]. However, we posit that manuscript generation continues to be one of the final "human" stages in the research pipeline.

We propose a fundamental reframing of this landscape. Scientific manuscripts, when written in LaTeX, represent a specialized form of code generation. LaTeX is not merely a document formatting system but a Turing-complete programming language with control structures, macros, and precise syntax requirements. When we generate LaTeX source code programmatically, be it for figure descriptions, reference lists, or methodology sections, we are engaging in code generation in the fullest sense. Recent advances in code generation systems demonstrate that generating formal syntactic artifacts is substantially more tractable than generating unstructured natural language [3, 4].

The emergence of LLMs has enabled impressive feats of natural language generation. Tools like ChatGPT can draft prose in response to user prompts, generating fluent text that appears authoritative. However, this capability, while valuable for certain writing tasks, has inherent limitations when applied to scientific documentation [5, 6]. Natural language is inherently ambiguous, difficult to verify, and prone to hallucination. A language model asked to "write a paragraph about our experimental results" produces free-form text that may contain invented citations, contradictions with other sections, or

GeCoIn 2026: Generative Code Intelligence Workshop, co-located with the 35th International Joint Conference on Artificial Intelligence (IJCAI-ECAI 2026), August 15–17, 2026 — Bremen, Germany
zhisheng@grailai.io (Z. Tang); mayank@grailai.io (M. Kejriwal)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-08-14

CEUR-WS.org/Vol-4238/paper4.pdf

subtle misrepresentations of statistical significance. Recent studies have documented systematic failures of LLMs in scientific writing, including citation fabrication and claim inconsistency [7, 8, 9]. The user must then manually edit, verify, and integrate this text—a process that shifts the cognitive burden rather than eliminating it.

In contrast, targeting code as the generation output offers decisive advantages. When we generate LaTeX source code, we are producing a formal specification: a document that, when compiled, deterministically produces a specific visual artifact. The generated code can be parsed, validated, and composed with other generated or hand-written segments. Bibliography entries can be checked against database queries. Cross-references are automatically resolved. Mathematical notation is unambiguous. Figure placement and sizing are explicit. This shift from prose generation to code generation transforms manuscript preparation from an exercise in prose polishing into a problem of formal artifact synthesis, one that benefits from the tools and practices of software engineering: version control, testing, composition, and reproducibility [10, 4]. Furthermore, recent work on program-aided language models and symbolic reasoning demonstrates that code-based representations enable stronger semantic guarantees and integration with external verification systems [3].

Furthermore, code generation enables *integration with other computational systems.* A language model generating free text is a terminal point: once prose is generated, integrating it with data processing pipelines, figure generation systems, or reference management becomes manual work. In contrast, when manuscripts are generated as LaTeX code, they become composable artifacts. A data analysis script can directly produce embedded table code. A plotting library can generate publication-ready figure definitions with correct citation keys. A bibliography system can produce BibTeX entries that integrate seamlessly with the manuscript. This compositional approach (combining generated and hand-written code, data-driven sections, and verified references) is far more powerful than any chatbot's ability to generate fluent prose.

This reframing unlocks significant practical and philosophical advantages. From a practical perspective, it enables end-to-end automation of manuscript generation [11] from raw research outputs: experimental data, source code, figures, and datasets can flow directly through a code generation pipeline to produce manuscripts that pass a first round of quality tests [12]. From a philosophical perspective, it legitimizes manuscript preparation as a core component of the autonomous science agenda [13, 14], wherein the entire research lifecycle, ranging from hypothesis formation to final dissemination, operates under unified computational principles.

In this paper, we present the GRAIL agentic LaTeX editor, which was designed explicitly to support the **manuscript-as-code-generation paradigm**. GRAIL integrates LLMs, figure generation systems, reference management, and code analysis tools into a cohesive environment where users can request that entire manuscript components be generated programmatically. Recent work on automated survey generation [15, 16] and multi-agent scientific pipelines [17] has demonstrated the viability of AI-driven document generation; we extend this work by recognizing the manuscript itself as a code generation target. We discuss the technical foundations that make this possible, showcase concrete applications, and situate this work within the broader context of autonomous science.

2. Manuscript Components as Code Generation Targets

Different aspects of manuscript preparation present distinct opportunities and challenges for code generation. Understanding these variations is essential for designing a practical system.

2.1. LaTeX as a Programming Language

To establish manuscript generation as code generation, we must first clarify why LaTeX deserves to be understood as a programming language rather than a mere formatting markup. While this may seem obvious to TeX practitioners, it is often overlooked in contemporary discourse about scientific writing.

LaTeX possesses the essential characteristics of a programming language: it features variables (registers and macros), control flow structures (conditionals and loops via `\if` and iterative constructs), function

definitions (via `\newcommand` and `\newenvironment`), and libraries (document classes and packages). The grammar is formal and unambiguous, and compilation is deterministic. Perhaps most importantly, LaTeX is Turing-complete, meaning it can express any computable algorithm, though practical limitations make extreme computation impractical.

The implications are deeper than they seem at first glance. When a system generates LaTeX source code programmatically, it is not simply producing formatted text but executable programs that, when compiled, yield *precise programmatic and visual artifacts*. This is fundamentally different from generating plain text. The generation process can draw upon the full computational power of LaTeX to express complex document structures, conditional content, automatic numbering and cross-referencing, and sophisticated typographic rules.

Consider a concrete example: generating a bibliography. Rather than producing a plain-text list, a code generation system can produce BibTeX entries with correct field formatting, generate `\cite` commands with appropriate keys, and use BibTeX and natbib packages to automatically enforce citation style, handle special characters, and manage duplicate detection. This is code generation because the output is a program (in BibTeX and LaTeX syntax) that, when executed, produces the desired bibliography.

2.2. Automating Manuscript Components

Each manuscript component presents distinct technical challenges and opportunities for automation. These range from well-defined, structural tasks such as reference formatting to more complex linguistic challenges such as narrative prose generation. Understanding these distinct categories is essential for assessing what can be feasibly automated and what requires human expertise. Table 1 summarizes the primary components and their key automation challenges.

Component	Key Automation Challenges
Narrative Text	Ensuring factual accuracy, domain terminology correctness, appropriate uncertainty representation, and logical coherence across sections
Figures & Illustrations	Specifying visual intent programmatically; integrating data with visualization generation; ensuring reproducibility
References & Citations	Achieving comprehensive coverage, verifying correctness against authoritative databases, enforcing consistent formatting
Methodology Descriptions	Translating implementation details from code into accurate prose; maintaining fidelity between documented and actual methods

Table 1
Manuscript components and their primary automation challenges. Each component requires specific technical approaches to realize its automation potential.

Narrative text generation presents perhaps the most severe challenge. Scientific writing imposes strict constraints that general prose generation cannot satisfy: statements must be factually accurate, terminology must be used correctly within the discipline, uncertainty must be properly qualified, and logical connections across sections must be maintained. Recent research has identified systematic failures of language models in scientific writing, including citation fabrication, claim inconsistency, and subtle misrepresentation of statistical significance [8, 9]. The fundamental problem is that natural language is inherently ambiguous and difficult to verify, making generated prose vulnerable to these errors without substantial human oversight.

Figure generation is more tractable. The core challenge is specification: how can visual intent be expressed in a form amenable to code generation? Once intent is specified, well-established programmatic methods can generate figures from data (matplotlib, ggplot2, gnuplot) or produce illustrations from natural language descriptions. However, seamless integration between data, visualization parameters, and LaTeX document generation remains non-trivial. The goal is to ensure that figures are reproducible from their underlying data and specifications, not manually edited artifacts.

Bibliography and reference management face challenges of coverage, correctness, and consistency. Given a description of desired source material, a system must identify comprehensive, relevant papers;

retrieve correctly formatted bibliographic entries; and ensure consistency with citation databases. The risk is incomplete coverage, missing key literature, or citations that do not precisely match the retrieved entries. Verification against authoritative sources is essential.

Methodology descriptions present a distinct challenge: bridging the gap between implementation and documentation. When experimental code or analysis pipelines exist, the manuscript must describe these methods accurately and completely. The challenge is automatically extracting this information from code and translating it into narrative form. Systems that analyze code structure through abstract syntax trees or static analysis can identify key computational steps, but converting these into clear prose descriptions of methodology requires sophisticated code-to-language translation while maintaining fidelity to the actual implementation. Inconsistencies between documented and actual methods represent a significant source of irreproducibility in computational research.

3. The GRAIL Platform

GRAIL[1] is a specialized LaTeX editor built on the principle that manuscripts are programs that should be generated, refined, and compiled through an integrated development environment. It was designed to address many of the automation challenges discussed earlier. Rather than a traditional document editor (like Overleaf or Google Docs), GRAIL functions as an Integrated Development Environment (IDE) for manuscript development. As shown in Figure 1, the platform integrates a source editor, live PDF preview, document outline navigation, and direct access to AI-assisted tools in a unified workspace.

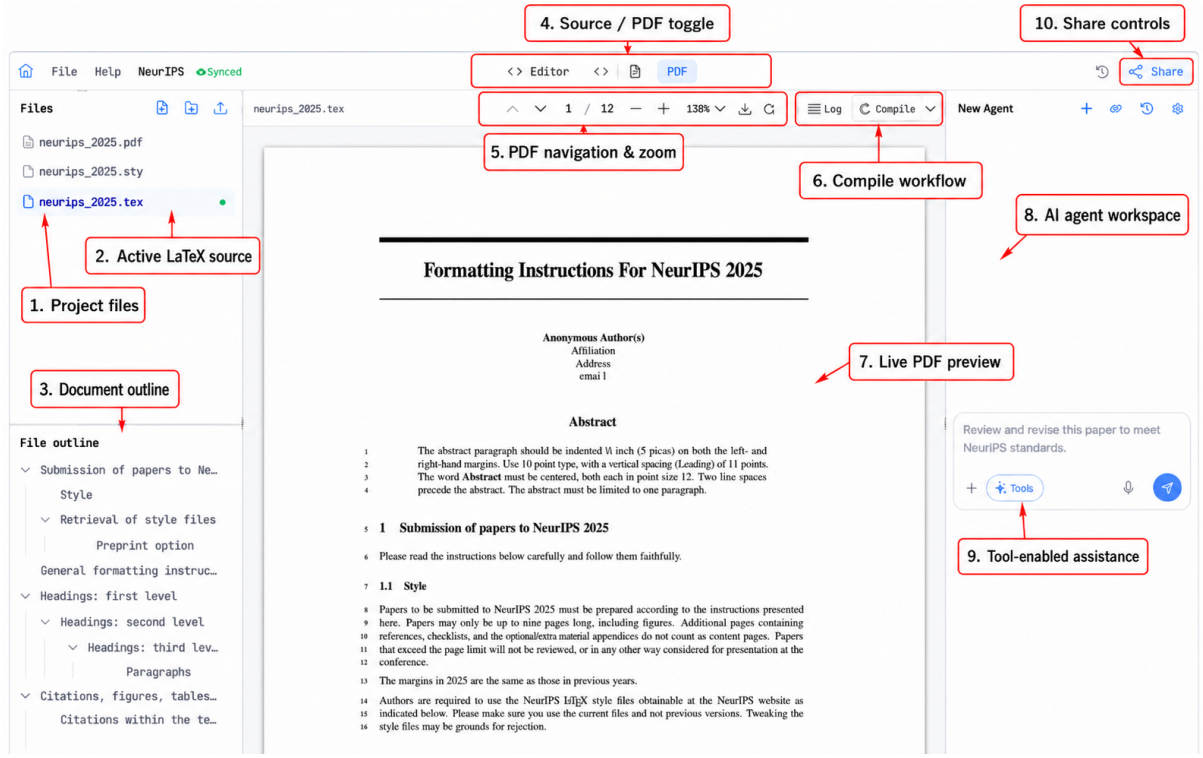

Figure 1: GRAIL platform interface showing the integrated editing environment. The left panel displays the project file structure and document outline, while the center panel shows the active LaTeX source. The right panel provides live PDF preview with compilation feedback and access to AI-assisted tools. Numbered features highlight key interface elements: (1) project files, (2) active LaTeX source indicator, (3) document outline navigation, (4) source and PDF toggle, (5) PDF navigation and zoom controls, (6) compile workflow button, (7) live preview pane, (8) AI agent workspace access, (9) tool-enabled assistance panel, and (10) share and collaboration controls.

[1] https://grailai.io

3.1. Core Architecture

GRAIL's architecture is built on a fundamental insight: the CRDT (Conflict-free Replicated Data Type) document model—specifically a shared Y.js document—serves simultaneously as the workspace for human collaboration, the state representation for the agent, and the canonical source for compilation and versioning. This unified model ensures there is no second source of truth; all subsystems (editor, agent, compiler, version history) operate on the same underlying document.

Humans collaborate in realtime with other users via the same CRDT, maintaining awareness of concurrent edits, presence, and granular undo/redo. The agent reads a snapshot of this CRDT document, materializes it into a working directory, runs an LLM-driven tool loop (invoking external services for search, analysis, or generation), and publishes its changes back as a unified diff. The collaboration server applies this diff inside a transaction, so the agent's edits appear to humans as normal collaborative updates with full undo capability. Critically, because both humans and the agent operate on the same CRDT model, concurrent edits are automatically reconciled—there is no risk of overwriting each other's work or managing version conflicts manually.

The key innovation is a request-response model embedded within LaTeX comments, inspired by agentic reasoning frameworks [18, 19, 20]. A user writes a specially formatted comment such as `% @generate: "Explain the experiments in a methodology section"` and invokes a keyboard shortcut. The system parses this request and invokes the agent, which analyzes the manuscript context and generates LaTeX code at the appropriate location within the document structure. The agent's edits are inserted as tracked changes, and the user can then review, refine, or request regeneration. This human-in-the-loop approach ensures that users maintain agency and oversight while benefiting from AI assistance.

Integration Points. GRAIL orchestrates multiple specialized agents and external services. A dedicated related-work agent handles literature synthesis: it performs multi-phase search across academic databases, retrieves papers, mines citations from PDFs, validates coverage of the requested topic, and synthesizes BibTeX entries and `\cite` commands that integrate directly into the manuscript. Other integrations include large language models for narrative generation, image generation models for illustrations, and code analysis tools for extracting methodology descriptions from source repositories. The agent can chain these tools together—for instance, searching for papers, retrieving their full texts, mining relevant citations, and generating a complete Related Work section with proper bibliography integration—all orchestrated through a single user request. All integrations are transparent to the user; they simply make requests in natural language and receive properly formatted LaTeX code that compiles correctly.

4. Experiments and Preliminary Data

To evaluate GRAIL's real-world performance and user adoption, we deployed the platform and collected preliminary metrics. The system processed 13,910 user requests from 332 unique users, with significant engagement concentrated among active users: 119 users (35.8% of the user base) made more than 10 requests, and 20 power users generated over 100 requests each. This distribution indicates strong adoption among a dedicated core group. Daily active users over the deployment period ranged from 1 to 33, with sustained peaks in late January and early March, as shown in Figure 2.

Session analysis reveals that engaged users spend significant time with the system. We define a session as consecutive agent requests without a break exceeding 30 minutes. The median session duration is 7.7 minutes, with average sessions lasting 21.2 minutes. Across 2,603 total sessions, users accumulated 921.5 hours of engaged time, demonstrating sustained and intensive interaction with the platform. The system generated 4.4 billion tokens per request, indicating that the system performs substantial reasoning and code generation.

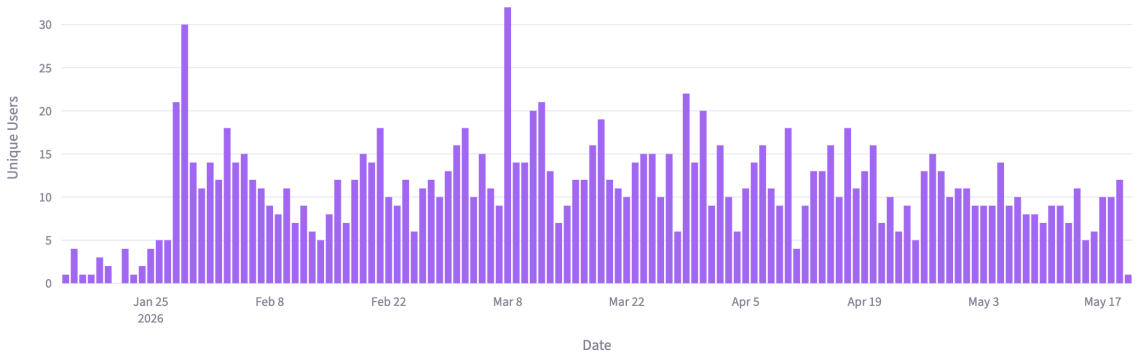

Figure 2: Daily active users over the measurement period (January 25 – May 17, 2026), showing sustained engagement with peaks in late January and early March.

4.1. Applications and Case Studies

To illustrate the practical value of manuscript-as-code-generation, we describe several concrete applications.

Grant Proposal Generation. Funding agencies increasingly request detailed methodology and significance statements. A researcher with existing preliminary data and publications can use GRAIL to rapidly generate a grant proposal scaffold. She provides abstract summaries of each research aim, indicates data files to be described, and requests that GRAIL generate methodology and preliminary results sections. GRAIL queries her data files, generates summary statistics, creates figures showing preliminary results, and writes narrative descriptions of the methodology. The researcher then refines the prose, adds additional context, and produces a complete proposal in a fraction of the time required for manual writing.

Autonomous Science Documentation. In autonomous science workflows, experimental systems perform research with minimal human intervention [13, 21]. These systems make decisions about experimental design, collect data, and analyze results. GRAIL enables automatic manuscript generation by consuming outputs from the autonomous system (experiment logs, results databases, figure specifications) and generating the corresponding LaTeX manuscript sections. This closes the loop: research is conducted automatically, and results are automatically documented.

Multilingual Document Generation. LaTeX source can be generated in multiple languages by varying the prompt or selecting different language models. A researcher can maintain a single LaTeX source template with generation requests, then invoke language-specific generation to produce manuscripts in English, Spanish, and Mandarin, with minimal additional work.

5. Positioning Within Autonomous Science

The autonomous science vision imagines research systems that operate with high degrees of autonomy: autonomous hypothesis generation, experimental design, execution, data analysis, and conclusions [13, 14, 17]. A recognized gap in this pipeline has been manuscript preparation. How does an autonomous system document its findings? Traditional approaches rely on post-hoc human authorship, which is inefficient and defeats some of the purpose of automation.

By treating manuscript generation as code generation, we integrate documentation into the autonomous pipeline. An autonomous science system can generate its own LaTeX source code describing its methodology and results. This source code can be compiled to produce publication-ready PDFs, submitted to preprint servers, or further refined by human authors. The key insight is that there is no fundamental distinction between autonomous code generation for a scientific analysis tool and autonomous code generation for a manuscript describing that tool. Frameworks like CycleResearcher [22] and aiXiv [23] show how AI-generated research outputs can be systematically evaluated and disseminated.

6. Related Work

The evolution of artificial intelligence in scientific research has transitioned from retrieval aids to autonomous agents capable of hypothesis generation and document synthesis [1, 2]. Early foundational efforts focused on domain-specific scientific models such as SciBERT [5], Galactica [6], and SciGPT [24], which established the groundwork for citation reasoning and knowledge grounding. Recent advancements have pushed toward fully automated manuscript generation, as demonstrated by the AI Scientist-v1 and v2 frameworks [13, 14], as well as specialized systems like DeepScientist [21]. To ensure academic integrity, tools like ScholarCoPilot [8] introduce citation-aware triggers, while CiteGuard [9] provides retrieval-augmented validation to ensure model-generated claims match their supporting literature.

Automated writing increasingly relies on agentic reasoning and multi-agent coordination. General frameworks such as Chain-of-Thought (CoT) [18], ReAct [19], and Tree-of-Thoughts [25] have been synthesized as fundamental architectural patterns for scientific agents [20]. In the context of complex document generation, multi-agent systems like SurveyForge [15] and SciSage [16] employ specialized roles for outline planning and memory management. Furthermore, comprehensive pipelines such as AI-Coscientist [17] integrate literature synthesis with experimental design to support end-to-end discovery.

Beyond natural language, scientific discovery often requires executable code and rigorous evaluation. Program-Aided Language models (PAL) [3] and CodeScientist [26] leverage Python execution for protocol scripting and experimental automation, while symbolic reasoning tools like LLM-SR [10] and ChemCrow [4] enable precise equation discovery and chemical synthesis planning. To complete the research lifecycle, automated review systems such as DeepReview [27] and the CycleResearcher framework [22] provide human-like deep thinking for peer review and manuscript refinement. Large-scale evaluation ecosystems like aiXiv [23] further facilitate impact prediction and structured judgment of AI-generated scientific outputs.

Our work on GRAIL positions manuscript generation as a specialized form of code generation, recognizing that LaTeX is a programming language rather than mere markup. This framing extends existing work on automated document generation by explicitly treating the output (LaTeX source code) as a target for code generation techniques, enabling integration with code analysis tools and making the manuscript preparation process reproducible and auditable.

7. Discussion and Conclusion

The manuscript-as-code-generation paradigm reframes scientific writing as a formal artifact synthesis problem rather than a primarily linguistic task. This framing elevates manuscript preparation from a post-hoc documentation activity into a core component of the research pipeline—one amenable to the rigorous practices of software engineering: version control, testing, composition, and reproducibility. By treating LaTeX as a programming language and recognizing manuscripts as executable programs, we enable integration with code analysis tools, data pipelines, and automated verification systems. A key implication is that automating routine aspects of manuscript generation (figure synthesis, reference formatting, methodology extraction from code) frees researchers to focus on the aspects that truly require expertise: conceptual clarity, logical coherence, and narrative structure.

However, significant challenges remain. Generated prose is inherently imperfect and requires human review; large language models continue to make errors, hallucinate citations, and occasionally misrepresent technical content [8, 9]. The most sophisticated automation targets—narrative text generation and synthesis across sections—involve inherent ambiguity and verification difficulties that resist full automation. Recent work on deep review systems [27] demonstrates both promise and limitations in AI-assisted evaluation of scientific content. Additionally, treating manuscripts as code-generated artifacts introduces new challenges: ensuring that generated output retains fidelity to intended meaning, maintaining consistency across regenerations, and preserving human accountability

in research dissemination. These constraints argue for a human-in-the-loop model where systems generate initial content that authors actively review and refine, rather than producing autonomous final copy.

Looking forward, several directions merit investigation. Deeper integration of code generation with experimental workflows and autonomous science systems could enable end-to-end manuscript generation from experiment logs and analysis code. Stronger formal guarantees about accuracy in generated text—perhaps through symbolic reasoning or structured validation against external knowledge bases—could mitigate hallucination risks. Better tools for collaborative manuscript development, where humans and agents iteratively refine content with full undo and presence awareness, remain an open frontier. The code generation perspective on manuscript preparation is not a solution to authorship, but rather a foundational shift in how we understand scientific documentation: as a computational problem with the same rigor and tooling that scientific software development demands.

Acknowledgments

This work was supported in part by the GRAIL platform development initiative. We thank our colleagues who provided early feedback on the conceptual framing of writing as code generation.

Declaration on Generative AI

Appropriately, a paper on manuscript generation as code generation has itself been composed using generative AI assistance. The authors used large language models to draft initial versions of several sections and to refine prose for clarity and academic style. All generated content was reviewed and significantly edited by human authors, who made structural decisions about paper organization, developed the core conceptual framing, and take full responsibility for the paper's scientific content and claims. All citations and references have been verified by human authors.

References

[1] Q. Chen, M. Yang, L. Qin, J. Liu, Z. Yan, J. Guan, D. Peng, Y. Ji, H. Li, M. Hu, Y. Zhang, Y. Liang, Y. Zhou, J. Wang, Z. Chen, W. Che, AI4Research: A survey of artificial intelligence for scientific research, arXiv preprint arXiv:2507.01903, 2025.

[2] H. Zhang, R. Li, Y. Zhang, T. Xiao, J. Chen, J. Ding, H. Chen, The evolving role of large language models in scientific innovation: Evaluator, collaborator, and scientist, arXiv preprint arXiv:2507.11810, 2025.

[3] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, G. Neubig, PAL: Program-aided language models, in: International Conference on Machine Learning, 2023.

[4] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, P. Schwaller, Augmenting large language models with chemistry tools, Nature Machine Intelligence 6 (2024) 525–535. URL: https://doi.org/10.1038/s42256-024-00832-8. doi:10.1038/s42256-024-00832-8.

[5] I. Beltagy, K. Lo, A. Cohan, SciBERT: A pretrained language model for scientific text, arXiv preprint arXiv:1903.10676, 2019.

[6] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, R. Stojnic, Galactica: A large language model for science, arXiv preprint arXiv:2211.09085, 2022.

[7] X. Wu, Z. Tang, M. Kejriwal, Reliable scientific grounding as a foundation for autonomous science, 2026. URL: https://doi.org/10.13140/RG.2.2.10744.81920, preprint.

[8] Y. Wang, X. Ma, P. Nie, H. Zeng, Z. Lyu, Y. Zhang, B. Schneider, Y. Lu, X. Yue, W. Chen, ScholarCopilot: Training large language models for academic writing with accurate citations, arXiv preprint arXiv:2504.00824, 2025.

[9] Y. M. Choi, X. Guo, Y. R. Fung, Q. Wang, CiteGuard: Faithful citation attribution for LLMs via retrieval-augmented validation, arXiv preprint arXiv:2510.17853, 2025.

[10] P. Shojaee, K. Meidani, S. Gupta, A. B. Farimani, C. K. Reddy, LLM-SR: Scientific equation discovery via programming with large language models, arXiv preprint arXiv:2404.18400, 2024.
[11] V. L. Ravideshik, M. Kejriwal, Can AI evaluate AI scientists? A benchmarking study of autonomous research generation systems using automated multi-model review, 2026. Preprint.
[12] M. Kejriwal, Z. Tang, N. Nananukul, A. Aravindan, Y. Sun, A hybrid-ai architecture for rapid autonomous scientific discovery, 2026. doi:10.13140/RG.2.2.22908.30084.
[13] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, D. Ha, The AI Scientist: Towards fully automated open-ended scientific discovery, arXiv preprint arXiv:2408.06292, 2024.
[14] Y. Yamada, R. T. Lange, C. Lu, S. Hu, C. Lu, J. Foerster, J. Clune, D. Ha, The AI Scientist-v2: Workshop-level automated scientific discovery via agentic tree search, arXiv preprint arXiv:2504.08066, 2025.
[15] X. Yan, S. Feng, J. Yuan, R. Xia, B. Wang, L. Bai, B. Zhang, SurveyForge: On the outline heuristics, memory-driven generation, and multi-dimensional evaluation for automated survey writing, in: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, 2025.
[16] X. Shi, Q. Kou, Y. Li, N. Tang, J. Xie, L. Yu, S. Wang, H. Zhou, SciSage: A multi-agent framework for high quality scientific survey generation, arXiv preprint arXiv:2506.12689, 2025.
[17] J. Gottweis, W.-H. Weng, A. Daryin, T. Tu, P. Sirkovic, A. Myaskovsky, G. Glowaty, F. Weissenberger, A. Orlandi, D. Popovici, A. Palepu, K. Rong, R. Tanno, K. Saab, F. Zhang, J. Blum, A. Carroll, K. Kulkarni, N. Tomasev, D. Zverinski, I. Rendulic, E. Vedadi, F. Hasler, L. Rimanic, M. Boia, I. Budiselic, B. Feinstein, M. Bellaiche, T. Sheffer, J. Freyberg, J. Ratcliff, O. Bertolli, K. Chou, A. Hassidim, B. Gokturk, A. Vahdat, Y. Guan, V. Dhillon, E. D. Vaishnav, B. Lee, T. R. D. Costa, J. R. Penadés, G. Peltz, Y. Matias, J. Manyika, D. Hassabis, Y. Xu, P. Kohli, A. Pawlosky, A. Karthikesalingam, V. Natarajan, Accelerating scientific discovery with Co-Scientist, Nature 655 (2026) 487–496. URL: https://doi.org/10.1038/s41586-026-10644-y. doi:10.1038/s41586-026-10644-y.
[18] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Advances in Neural Information Processing Systems, 2022.
[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing reasoning and acting in language models, in: The Eleventh International Conference on Learning Representations, 2023.
[20] B. Zhao, L. G. Foo, P. Hu, C. Theobalt, H. Rahmani, J. Liu, LLM-based agentic reasoning frameworks: A survey from methods to scenarios, arXiv preprint arXiv:2508.17692, 2025.
[21] Y. Weng, M. Zhu, Q. Xie, Q. Sun, Z. Lin, S. Liu, Y. Zhang, DeepScientist: Advancing frontier-pushing scientific findings progressively, arXiv preprint arXiv:2509.26603, 2025.
[22] Y. Weng, M. Zhu, G. Bao, H. Zhang, J. Wang, Y. Zhang, L. Yang, CycleResearcher: Improving automated research via automated review, in: International Conference on Learning Representations (ICLR), 2025.
[23] P. Zhang, X. Hu, G. Huang, Y. Qi, H. Zhang, X. Li, J. Song, J. Luo, Y. Li, S. Yin, C. Dai, E. H. Jiang, X. Zhou, Z. Yin, B. Yuan, J. Dong, G. Su, G. Qiao, H. Tang, A. Du, L. Pan, Z. Lan, X. Liu, aiXiv: A next-generation open access ecosystem for scientific discovery generated by AI scientists, arXiv preprint arXiv:2508.15126, 2025.
[24] F. She, N. Wang, H. Wu, Z. Wan, J. Wang, C. Wang, SciGPT: A large language model for scientific literature understanding and knowledge discovery, arXiv preprint arXiv:2509.08032, 2025.
[25] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, K. Narasimhan, Tree of Thoughts: Deliberate problem solving with large language models, in: Advances in Neural Information Processing Systems, 2023.
[26] P. Jansen, O. Tafjord, M. Radensky, P. Siangliulue, T. Hope, B. Dalvi Mishra, B. P. Majumder, D. S. Weld, P. Clark, CodeScientist: End-to-end semi-automated scientific discovery with code-based experimentation, in: Findings of the Association for Computational Linguistics: ACL 2025, 2025, pp. 13370–13467. URL: https://aclanthology.org/2025.findings-acl.692/. doi:10.18653/v1/2025.findings-acl.692.
[27] M. Zhu, Y. Weng, L. Yang, Y. Zhang, DeepReview: Improving LLM-based paper review with human-like deep thinking process, in: Proceedings of the 63rd Annual Meeting of the Association

for Computational Linguistics (Volume 1: Long Papers), 2025, pp. 29330–29355. URL: https://aclanthology.org/2025.acl-long.1420/.