JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim Viewpoint

Chat-Driven Computational (Bio)chemistry: Using LLM Agents to Accelerate Bio- and Chemoinformatics

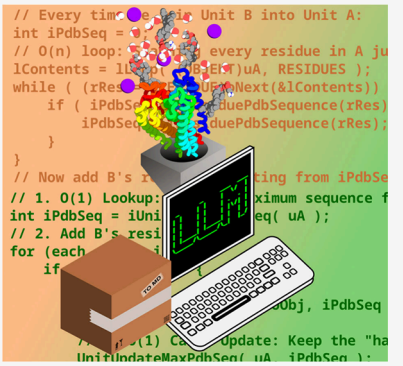
Published as part of Journal of Chemical Information and Modeling special issue "Harnessing the Power of Large Language Model-Based Chatbots for Scientific Discovery".

Stephan Schott-Verdugo and Holger Gohlke*

Cite This: J. Chem. Inf. Model. 2026, 66, 3397−3401 Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: Large-language models (LLMs) have rapidly become essential in software engineering, evolving from simple code suggestion tools to autonomous agents that directly read, modify, compile, and test local code bases. Recent LLMs perform well in software engineering benchmarks, showing good performance on complex multifile projects, generating new options for improving and developing bio- and chemoinformatic tools. We showcase this capability with the AMBER molecular dynamics suite, where the setup program *LEaP* suffered an $O(N^2)$ merge routine and a 32-bit integer overflow, limiting simulation systems to ∼6 million atoms. By using an LLM, we implemented an optimized unit merge algorithm and 64-bit indexing, cutting the parametrization time by more than 10-fold for mid-sized systems and allowing one to parametrize multimillion-molecule systems. This case illustrates how natural scientists can make use of LLM agents to modernize, optimize, and develop computational (bio)chemistry tools while also raising new challenges for software provenance and developer roles.



During the last year, LLMs have been rapidly incorporated into routines and workflows at the consumer and professional level, with applications as general assistants, for content generation, customer support, or educational purposes.[1] Starting from the promise shown by OpenAI's GPT-3 in 2020, followed in 2022 by more practical applications of ChatGPT/GPT-4, Anthropic's Claude or, more recently, by Google's Gemini (a comprehensive timeline can be seen in ref 2), these different models have been able to help in everyday problem fixing, but also in different realms of science. Probably one of the areas where they are causing the biggest impact is in software engineering, with uses in code generation and debugging.[3] While still met with certain levels of skepticism,[4] using AI agents for coding has caught attention,[5] with, e.g., Spotify acknowledging by the end of 2025 that their main developers are vibe-coding, i.e., not coding themselves anymore and only proof-reading the code written by LLM agents.[6]

While models before 2025 allowed this to be done for simpler problems, their use for bigger code structures and complex tasks was rather limited until recently, as evidenced by the software engineering benchmark SWE-Verified[7,8] (Figure 1; see p 72 of ref 9 for an extrapolation). While many early models and platforms supported uploading folder structures with a given program code, they could not capture the global context or come up with a valid solution. Instead, non-existing modules were hallucinated, or whole code sections were mistyped. From personal experience, we got the impression that every other instruction failed and that the main goal established in the initial prompt got lost in the context of trying to correct the mistakes made by the LLM itself.

This changed dramatically with newer models and the implementation of Agentic AI tools, such as Anthropic's Claude Code,[10] OpenAI Codex,[11] Google's Gemini CLI,[12] and OpenCode.[13] These programs run as a shell instance that establishes a direct connection between the cloud LLM and a given local folder structure that holds, e.g., the source code for an already existing program, or might as well be an empty folder where one would like to start a new project. From there, and similar to agents implemented in integrated development environments (IDEs), the programs can read the source code as its context directly, without requiring the user to upload and download files from the web. With this, the agent is able to make modifications to the files directly, call executables present in the environment that could be

Received: March 1, 2026
Published: March 18, 2026

https://doi.org/10.1021/acs.jcim.6c00633
J. Chem. Inf. Model. 2026, 66, 3397−3401

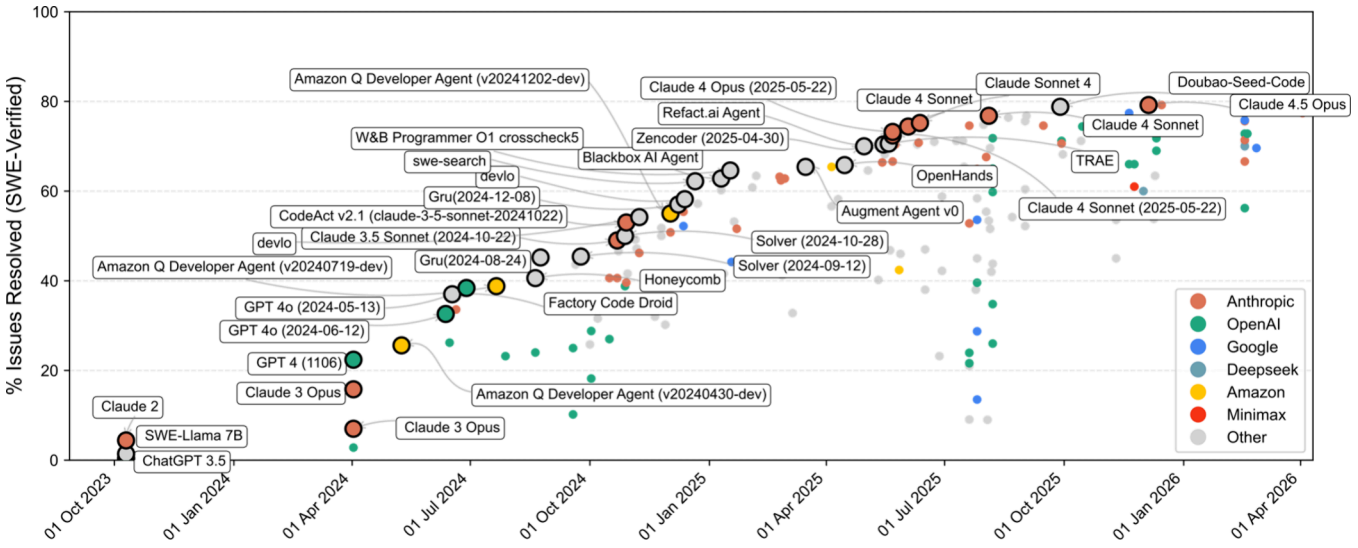


Figure 1. Percentage of solved tasks of the software engineering SWE-Verified benchmark versus submission date as of February 2026.[8] Colors reflect the model family/developer, and labels show the names of the entries, omitting the running agent for clarity. Only models at the frontier of the state of the art are labeled, and they are highlighted with bigger scatter dots. Multiple entries are present for the same model, as they can be run from different agents.

required to debug, and test the implementation of the problem at hand. Its noteworthy that, while agents usually ask the user for permission before running relevant commands, there have been reports of miserable failures of running agents without safety nets.[14] Furthermore, these programs are able to orchestrate multiple other agents, in practice, resulting in a "team of coders" with multiple individual subtasks running at the same time, and potentially reducing development time massively. In the particular case of Claude Code, its easy-to-use implementation together with their latest model releases (Opus and Sonnet 4.5 and recently 4.6) have drawn significant attention by reaching the top of coding benchmarks (Figure 1), resulting in public outlets acknowledging the potential impact that generative AI will have on society and not only on specialized users.[5,6]

In the case of bio- and chemoinformatic tools, many algorithms are well established but may have been implemented using old code bases or programming languages, and many have not been updated to current standards or adapted to current scientific questions. Additionally, they may have been conceived with a different scope of problems in mind, as at the time, the community was not considering the computing and storage capacities we have nowadays, not to mention the usage of GPUs or newer computing paradigms such as exascale or quantum computing.[15] In addition, while many tools are scientifically correct, they have been written by coders by trade rather than by training, which suggests there are marked optimization options in many of the currently used approaches.

AMBER is a molecular dynamics (MD) software suite in development since the early 1980s,[16] which has more recently been accompanied by the open-source AmberTools package.[17] The suite includes one of the fastest MD engines running in a single GPU, *pmemd.cuda*,[18] and very powerful structure and MD trajectory analysis with *cpptraj*.[19] While there are constant improvements to the source code of base programs made by the Amber community,[20,17] many of them were developed in the 1990s/2000s and are not proactively improved.

A prime example of this is *LEaP*,[21,22] the main parametrization tool for the setup of simulation systems in AMBER. While it performs very well for small and mid-sized systems (up to about 100,000 atoms), the running time when considering millions of atoms becomes prohibitive, particularly when iterations are needed to accommodate modifications in the molecular system to be simulated. This contributed to the development of a port of the Amber force field to CHARMM formats, which, in this way, allows to parametrize and simulate systems with multiple million atoms in NAMD.[23,24] The authors claimed that the main cause for the lack of performance on *LEaP*'s side was due to the single-core implementation, but no attempt was made to optimize the code. Furthermore, it was not attempted to parametrize systems with more than 7 million atoms in *LEaP*, as the current code breaks when trying such big systems after running for more than a day.

As a putative reason for that, *LEaP* was written in the 1990s in C in a comprehensive but convoluted way with over a million lines of code, making optimization campaigns difficult. That way, *LEaP* has gained fame for how daunting it is to improve its code beyond its current functionalities, and because it has a prominent place as the only parametrization tool in the current AMBER workflow, the potential introduction of bugs may lead to a high-cost endeavor. This has resulted in an ongoing attempt at reimplementing adapted functions of *LEaP* into *cpptraj*[25] in order to replace it in a forthcoming release.

For exactly such problems, the current LLMs could provide a solution. Considering that *LEaP*'s code is very well structured and commented, it is the perfect input for such models. Initial attempts to identify the source of the crash when parametrizing a water box with more than 2,421,262 water molecules (7,263,786 atoms) using Claude's Web user interface and Sonnet 3.5 were unsuccessful. Following the release of Sonnet 4.5 in September 2025, a request was made to improve the speed performance of the program, aiming at faster debugging/testing cycles for the system size-related crash. By compiling the code with profiling options and

https://doi.org/10.1021/acs.jcim.6c00633
J. Chem. Inf. Model. 2026, 66, 3397−3401

Scheme 1. Pseudocode Part of (top) Original *LEaP* and (bottom) Modified *LEaP* for "Unit" Merging—The LLM-derived Hashing Approach (Bottom) Changed the Complexity from Originally $O(N^2)$ to $O(N)$

Original *LEaP*

```
// Every time we join Unit B into Unit A:
int iPdbSeq = 0;
// O(n) loop: Scanning every residue in A just to find the starting point for B
lContents = lLoop( (OBJEKT)uA, RESIDUES );
while ( (rRes = (RESIDUE)oNext(&lContents)) ) {
   if ( iPdbSeq < iResiduePdbSequence(rRes) ) {
      iPdbSeq = iResiduePdbSequence(rRes);
   }
}
// Now add B's residues starting from iPdbSeq + 1
```

Modified *LEaP*

```
// 1. O(1) Lookup: Get the maximum sequence directly from Unit struct
int iPdbSeq = uA->iMaxPdbSeq;
// 2. Add B's residues
for (each object in B) {
   if ( is_residue ) {
      iPdbSeq++;
      ResidueSetPdbSequence( oObj, iPdbSeq );

      // 3. O(1) Direct Update: Update the max sequence member directly.
      uA->iMaxPdbSeq = iPdbSeq;
   }
   ContainerAdd( uA, oObj );
}
```

providing gprof's[26] output to the LLM, it was identified that the "unit" merging function was spending the majority of the time in an unoptimized routine. In the context of *LEaP*, a "unit" is an object that holds force field parameters for protein residues, nucleotides, lipid head groups or tails, or water molecules. When parametrizing a system, all units are identified and merged iteratively into a single object or global unit, which is then output to the AMBER topology file. It turned out that in every iteration of the original merging routine, the highest residue number was identified by going over all residues inside of the "first" unit (which holds all residues resulting from the previous merge), resulting in an $O(N^2)$ scaling process (Original *LEaP*; Scheme 1). The LLM's suggestions to change the current behavior for a hashing approach, where the previous highest number is stored and summed to the current unit highest number, resulted in a massive reduction of execution time ($\gg$100 times faster for systems with over a million atoms) (Figure 2). Upon further discussion with other developers, it was noted that the highest number can be assigned directly, without using an additional hashing function structure (Modified LEaP; Scheme 1). As a follow-up, this reduction in execution time allowed to identify that the crash for big systems occured due to the element array used within *LEaP* being casted as a 32-bit integer instead of a 64-bit array. Both modifications will be included in the release of AMBER 26 and allow to parametrize systems of any size in a reasonable amount of time, where the only current limitation is the parm7 topology and the "10I8" format used to write dihedral indices, resulting in a current maximum of 11 million atoms.[27]

The experience described above goes beyond the implications for *LEaP* and AMBER in particular, and begs the question of what the future of coding for bio- and chemoinformatic tools is. Our practical example demonstrates that LLMs are in a position where a person with certain, but not necessarily application-specific, knowledge of coding can have a major impact on the development or optimization of software tools. It has already been disclosed that, in some cases, the main coders of commercial programs are not coding directly anymore, but rather guiding multiple agents toward a specific design goal.[6] The current state suggests that having notions of computer science and proper software design questions, and a vision on how to implement such solutions, will be more relevant than the coding itself. At the same time, the input from experienced coders and thorough quality control becomes relevant, particularly while current models are still being improved.

This poses several advantages but also drawbacks. On the one hand, this has the potential to boost tremendously the development of new tools, as well as to improve existing ones that have not been actively developed in a while. On the other hand, there are risks of over-reliance on coding assistants[29] or AI hallucinations, and experienced developers will have to adapt themselves to new coding approaches to stay competitive.[30] Furthermore, the increase in AI-based merge requests from low or no-experience coders in GitHub repositories of open-source tools has caused many repositories to change to a "merge request per invitation" approach,[31]

https://doi.org/10.1021/acs.jcim.6c00633
J. Chem. Inf. Model. 2026, 66, 3397−3401

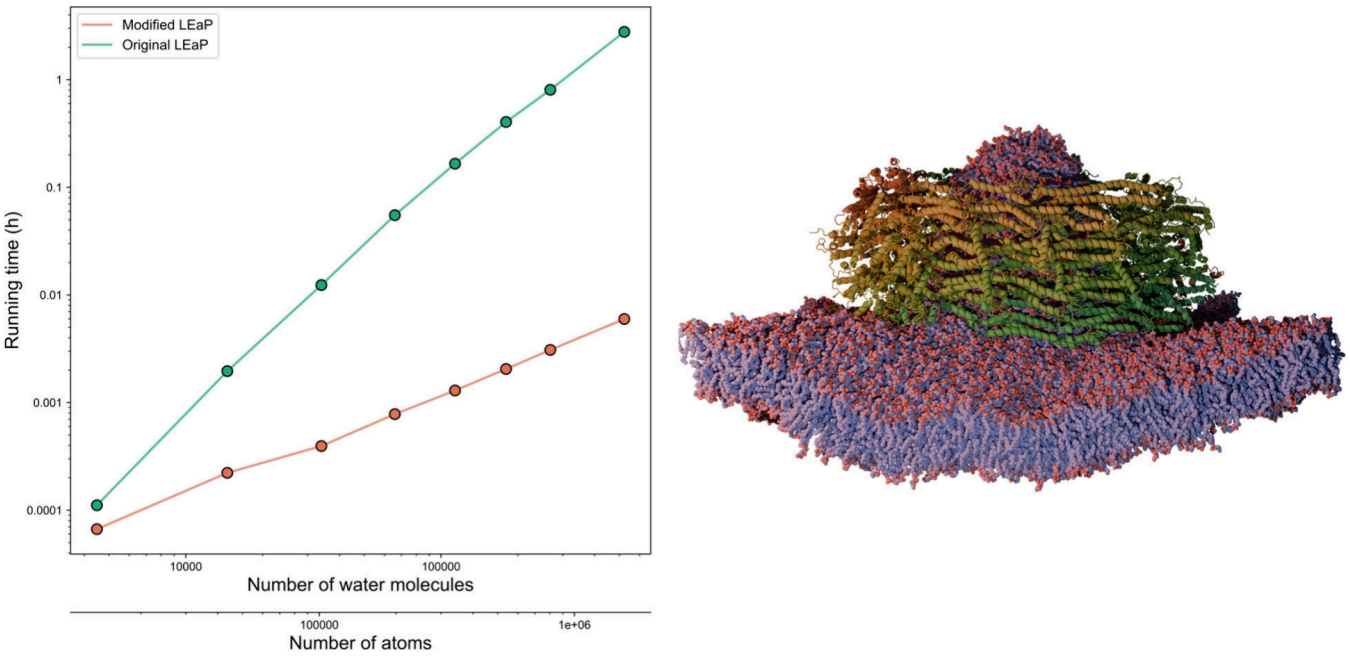


Figure 2. (left) Running time in hours for parametrizing a TIP3P water box using the Original *LEaP* or modified *LEaP* code following the suggestions made by Claude Sonnet 4.5 versus the number of water molecules (×3 to get the number of atoms) using a single Intel Core i7-10700 thread. Note the logarithmic scale on both axes. (right) As an example, the 60-mer protein complex of *PspA* interacting with a membrane in explicit solvent (omitted here for clarity) with 5.4 million atoms used in our previous work[28] took approximately a day to parametrize with *LEaP* in AmberTools25, while with the modified AmberTools26 implementation it takes less than 3 min.

with GitHub setting up tools to be able to manage the flood.[32] In addition, LLMs themselves have already changed how users try to solve their problems, with platforms such as Stack Overflow seeing a dramatic decline in their use and with vibe-coding resulting in a weaker user engagement, which is deemed a paradigm shift in open-source software development.[33] Last, but certainly not least, we are moving toward a state where, to code competitively, we need to pay a subscription to one of the big AI platforms mentioned above, imposing dependency and privacy risks.[1] This might change, likely even in the next months, with the development of openly available models that can potentially run on local hardware, but the current situation is that only models developed by large AI corporations are able to vibe-code in large projects with a lower probability of causing so-called AI-slop, or LLM-derived mistakes. All in all, it appears we are heading toward major changes during the next year.

■ ASSOCIATED CONTENT

Data Availability Statement

The Amber suite of biomolecular simulation programs, including AmberTools, is available at https://ambermd.org. The modified *LEaP* program will be made available in AmberTools26.

■ AUTHOR INFORMATION

Corresponding Author

Holger Gohlke − *Institute of Bio- and Geosciences (IBG-4: Bioinformatics), Forschungszentrum Jülich GmbH, 52425 Jülich, Germany; Institute for Pharmaceutical and Medicinal Chemistry, Heinrich Heine University Düsseldorf, 40225 Düsseldorf, Germany;* orcid.org/0000-0001-8613-1447; Email: gohlke@uni-duesseldorf.de

Author

Stephan Schott-Verdugo − *Institute of Bio- and Geosciences (IBG-4: Bioinformatics), Forschungszentrum Jülich GmbH, 52425 Jülich, Germany*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jcim.6c00633

Author Contributions

S.S.-V. designed the study, performed the research, and wrote the initial manuscript. H.G. analyzed the data, revised the manuscript, and secured funding.

Funding

H.G. is supported by the Helmholtz Foundation Model Initiative within the project "PROFOUND".

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

We acknowledge the developers of AMBER for fruitful discussions, in particular Daniel R. Roe, Juno Krahn, and David Case.

■ REFERENCES

(1) Barberá, I. *AI Privacy Risks & Mitigations: Large Language Models (LLMs)*; European Data Protection Board, 2025.

(2) *AI Timeline—Complete History of 194+ Large Language Models.* LLM Timeline. https://llm-timeline.com/ (accessed 2026-02-23).

(3) *Unlocking the value of AI in software development.* McKinsey, November 3, 2025. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/unlocking-the-value-of-ai-in-software-development (accessed 2026-02-23).

(4) Espada, I. *Why expert developers refuse to "vibe" with AI coding tools.* PPC Land, January 10, 2026. https://ppc.land/why-expert-

developers-refuse-to-vibe-with-ai-coding-tools/ (accessed 2026-02-23).
(5) Olson, B. Claude Is Taking the AI World by Storm, and Even Non-Nerds Are Blown Away. *The Wall Street Journal*, January 17, 2026. https://www.wsj.com/tech/ai/anthropic-claude-code-ai-7a46460e.
(6) Perez, S. *Spotify says its best developers haven't written a line of code since December, thanks to AI*. TechCrunch, February 12, 2026. https://techcrunch.com/2026/02/12/spotify-says-its-best-developers-havent-written-a-line-of-code-since-december-thanks-to-ai/ (accessed 2026-02-23).
(7) Jimenez, C. E.; et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv (Computer Science.Computation and Language)*, November 11, 2024, 2310.06770, ver. 3. https://arxiv.org/abs/2310.06770.
(8) *Official Leaderboards*. SWE-bench. https://www.swebench.com/ (accessed 2026-02-26).
(9) *AI in 2030: Extrapolating Current Trends*. https://epoch.ai/files/AI_2030.pdf (accessed 2026-02-26).
(10) *anthropics/claude-code*. https://github.com/anthropics/claude-code (accessed 2026-02-23).
(11) *openai/codex*. https://github.com/openai/codex (accessed 2026-02-23).
(12) *google-gemini/gemini-cli*. https://github.com/google-gemini/gemini-cli (accessed 2026-02-23).
(13) *anomalyco/opencode*. https://github.com/anomalyco/opencode (accessed 2026-02-23).
(14) Nolan, B. An AI-powered coding tool wiped out a software company's database, then apologized for a 'catastrophic failure on my part'. *Fortune*, July 23, 2025. https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/ (accessed 2026-02-23).
(15) Di Felice, R.; et al. A Perspective on Sustainable Computational Chemistry Software Development and Integration. *J. Chem. Theory Comput.* **2023**, *19* (20), 7056−7076.
(16) Weiner, P. K.; Kollman, P. A. AMBER : Assisted model building with energy refinement. A general program for modeling molecules and their interactions. *J. Comput. Chem.* **1981**, *2* (3), 287−303.
(17) Case, D. A.; et al. AmberTools. *J. Chem. Inf. Model.* **2023**, *63* (20), 6183−6191.
(18) Walker, R. C.; Le Grand, S., AMBER (PMEMD) GPU Support. https://ambermd.org/GPUSupport.php.
(19) Roe, D. R.; Cheatham, T. E. PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. *J. Chem. Theory Comput.* **2013**, *9* (7), 3084−3095.
(20) Case, D. A.; et al. Recent Developments in Amber Biomolecular Simulations. *J. Chem. Inf. Model.* **2025**, *65* (15), 7835−7843.
(21) Schafmeister, C.; Ross, W.; Romanovski, V. *LEaP*; University of California: San Francisco.
(22) Pearlman, D. A.; et al. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Comput. Phys. Commun.* **1995**, *91* (1−3), 1−41.
(23) Phillips, J. C.; et al. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *J. Chem. Phys.* **2020**, *153* (4), No. 044130.
(24) Antolínez, S.; Jones, P. E.; Phillips, J. C.; Hadden-Perilla, J. A. AMBERff at Scale: Multimillion-Atom Simulations with AMBER Force Fields in NAMD. *J. Chem. Inf. Model.* **2024**, *64* (2), 543−554.
(25) Roe, D. R., *drroe/cpptraj*. https://github.com/drroe/cpptraj (accessed 2026-02-26).
(26) Graham, S. L.; Kessler, P. B.; Mckusick, M. K. Gprof: A call graph execution profiler. *SIGPLAN Not.* **1982**, *17* (6), 120−126.
(27) *Amber File Formats: "PARM'" parameter/topology file specification*. https://ambermd.org/FileFormats.php#topology (accessed 2026-02-26).
(28) Hudina, E.; Schott-Verdugo, S.; Junglas, B.; Kutzner, M.; Ritter, I.; Hellmann, N.; Schneider, D.; Gohlke, H.; Sachse, C. The bacterial ESCRT-III PspA rods thin lipid tubules and increase membrane curvature through helix α0 interactions. *Proc. Natl. Acad. Sci.* **2025**, *122*, e2506286122.
(29) O'Brien, G. Threats to scientific software from over-reliance on AI code assistants. *Nat. Comput. Sci.* **2025**, *5* (9), 701−703.
(30) Hoover, A. The year coding changed forever. *Business Insider*, December 15, 2025. https://www.businessinsider.com/year-coding-changed-forever-silicon-valley-2025-12 (accessed 2026-02-26).
(31) *ghostty/CONTRIBUTING.md*. https://github.com/ghostty-org/ghostty/blob/main/CONTRIBUTING.md (accessed 2026-02-26).
(32) Wolf, A. Welcome to the Eternal September of open source. Here's what we plan to do for maintainers. *GitHub Blog*, February 12, 2026. https://github.blog/open-source/maintainers/welcome-to-the-eternal-september-of-open-source-heres-what-we-plan-to-do-for-maintainers/ (accessed 2026-02-26).
(33) Koren, M.; Békés, G.; Hinz, J.; Lohmann, A. Vibe Coding Kills Open Source. *arXiv (Economics.General Economics)*, January 21, 2026, 2601.15494, ver. 1. https://arxiv.org/abs/2601.15494.

https://doi.org/10.1021/acs.jcim.6c00633
J. Chem. Inf. Model. 2026, 66, 3397−3401