DeepRepro: State-Aware Subplanning for Paper-to-Code Reproduction in Evolving Repositories

Hongru Song
Ruqing Zhang*
State Key Laboratory of AI Safety,
Institute of Computing Technology,
Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
songhongru24s,zhangruqing@ict.ac.cn

Jiafeng Guo
Xueqi Cheng
State Key Laboratory of AI Safety,
Institute of Computing Technology,
Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
guojiafeng,cxq@ict.ac.cn

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
m.derijke@uva.nl

Abstract

Recent advances in agentic large language models (LLMs) have enabled increasingly autonomous software engineering workflows, yet automatic machine learning (ML) paper-to-code reproduction remains a challenging long-horizon problem. Unlike conventional code generation, this task requires constructing and maintaining a fully functional repository whose state continuously evolves during execution. Existing systems typically rely on static upfront planning followed by sequential file-level generation, which often leads to inconsistencies as dependencies, interfaces, and execution feedback change over time. We propose DeepRepro, a state-aware framework for paper-to-code reproduction based on execution-state-aware subplanning. DeepRepro dynamically transforms evolving repository states and runtime feedback into fine-grained implementation subplans, keeping planning aligned with execution throughout repository construction. The framework further incorporates repository-aware orchestration and a lightweight process-aware interface for transparent monitoring of long-horizon reproduction. Experiments on PaperBench Code-Dev show that DeepRepro consistently outperforms strong scientific and commercial code-agent baselines. Code and demo are available at https://github.com/ruyisy/DeepRepro and https://youtu.be/ZvmtVX7Rleo.

CCS Concepts

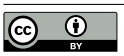
• **Computing methodologies** → **Planning and scheduling**.

Keywords

Multi-agent systems, Research reproduction, Code agents

ACM Reference Format:
Hongru Song, Ruqing Zhang, Jiafeng Guo, Xueqi Cheng, and Maarten de Rijke. 2026. DeepRepro: State-Aware Subplanning for Paper-to-Code Reproduction in Evolving Repositories. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM '26), November 07–11, 2026, Rome, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3799682.3840257

*Ruqing Zhang is the corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
CIKM '26, Rome, Italy
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2539-5/2026/11
https://doi.org/10.1145/3799682.3840257

1 Introduction

Recent advances in agentic large language models (LLMs) are rapidly transforming code assistants from passive completion tools into autonomous software engineering agents capable of building full repositories from high-level instructions [6, 7, 17, 23]. A particularly challenging instantiation of this trend is *scientific paper-to-code reproduction*, where an agent must reconstruct a complete, executable codebase from a research paper description [13, 18]. Unlike traditional code generation, this task requires not only understanding algorithms, but also assembling interdependent modules, resolving implicit implementation details, and producing reproducible experimental pipelines.

Recent benchmarks such as PaperBench provide a structured evaluation setting with fine-grained rubrics for measuring reproduction quality [19]. Despite progress in agentic coding systems, even strong iterative agents still fall far behind human experts. Recent specialized frameworks improve over general-purpose coding agents, yet their performance remains limited in long-horizon reproduction settings [18, 25]. A key limitation is that most existing systems rely heavily on an initial global plan, and then proceed with incremental repository construction.

However, in paper-to-code reproduction, the repository is a dynamic and evolving artifact. As code is generated, new interfaces emerge, dependencies become concrete, and earlier assumptions may become invalid. As a result, a static plan quickly becomes misaligned with the actual implementation state, leading to cascading inconsistencies and repeated repair cycles. In addition, since reproduction is long-running and resource-intensive, users also lack a clear view into intermediate progress, partial failures, and system-level diagnostics. This reveals a fundamental challenge: *effective paper-to-code reproduction requires continual alignment between planning and an evolving codebase state, rather than a one-shot decomposition followed by static execution.*

To this end, we propose DeepRepro, a state-aware framework for paper-to-code reproduction based on iterative subplanning over evolving repositories. The key idea is to treat repository construction as an iterative process where planning is continuously conditioned on the current implementation state. As shown in Figure 1,

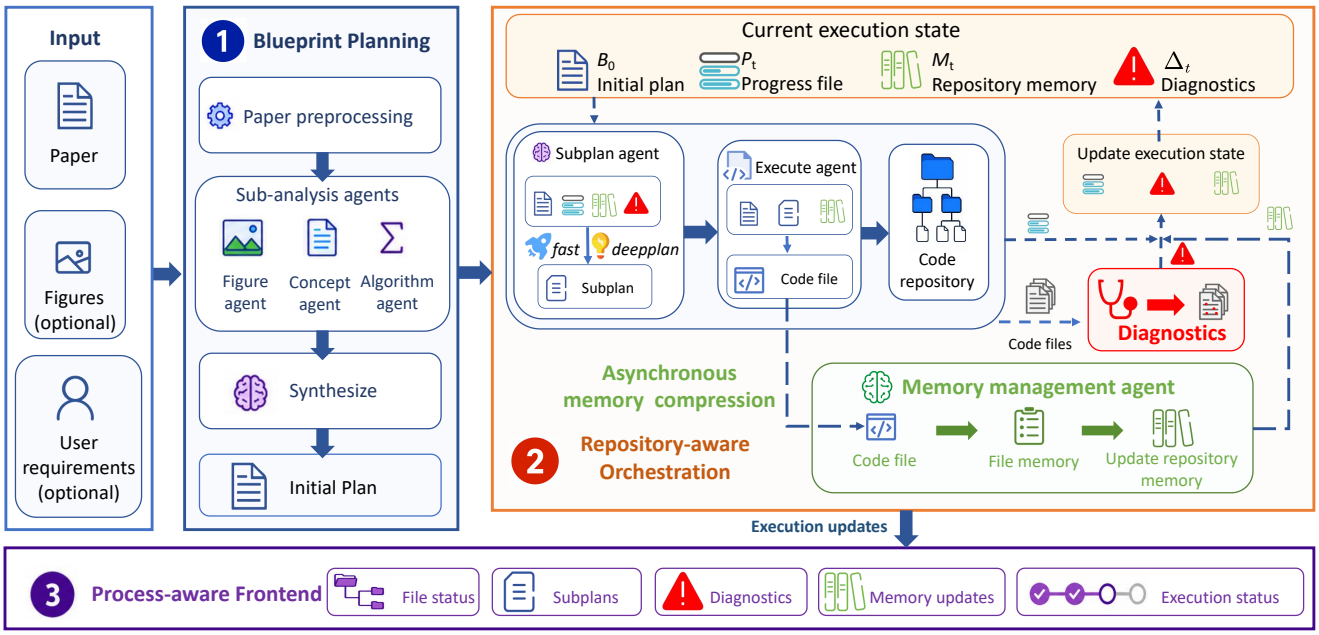


Figure 1: Overview of the DeepRepro framework for paper-to-code reproduction.

DeepRepro consists of three tightly coupled components: ❶ *Blueprint planning* first distills a research paper into a repository-level skeleton, including core modules, execution flow, dependencies, and evaluation protocols. ❷ *Repository-aware orchestration* then constructs the repository through a state-conditioned iterative loop. At each round, a subplanning module dynamically selects a set of tightly coupled files or repair targets based on the current repository state; an execution module implements them using tool-augmented generation; a memory module compresses completed artifacts into structured repository memory; and a bounded repair mechanism feeds execution diagnostics back into subsequent planning steps. This design supports both lightweight file-level execution (*fast mode*) and deeper dependency-aware grouping and ordering (*deepplan mode*). And ❸ a *process-aware frontend* exposes the intermediate reproduction process, including file status, evolving subplans, and runtime diagnostics, enabling transparent monitoring of long-horizon execution.

We evaluate DeepRepro on PaperBench [19]. On the five-paper subset, DeepRepro achieves an average score of 84.2, improving over DeepCode, Cursor, and Codex by 1.3, 17.7, and 23.2 points, respectively; on the shared human-expert subset, it exceeds the reported human expert results by 5.4 points. Further analysis shows that deep subplanning brings a 2.15-point improvement.

2 The DeepRepro Framework

Given $\mathcal{D} = (p, \mathcal{F}, u)$, where p is the input paper, $\mathcal{F}$ denotes optional figures, and u denotes optional user requirements, DeepRepro aims to synthesize a repository $\mathcal{R}^* = \arg\max_{\mathcal{R}} \text{Score}(\mathcal{R} \mid \mathcal{D})$. As shown in Figure 1, DeepRepro first constructs a blueprint-level plan B_0, and then performs repository construction over the evolving state.

2.1 Blueprint planning

Paper input and preprocessing. Blueprint planning starts by normalizing heterogeneous inputs. Given a local paper file or URL, optional user requirements, and auxiliary figures, DeepRepro creates a task-specific workspace, copies or downloads the paper, and converts it into a Markdown document for later agents. It then standardizes paths for planning and implementation artifacts, and chooses either full-document reading or section-based segmented reading according to the document size and preprocessing status. The output is a clean workspace and a preprocessed document representation $\tilde{\mathcal{D}}$ for initial planning.

Paper analysis and initial planning. DeepRepro uses a decompose-and-summarize pipeline to build the initial reproduction blueprint. Three analysis agents produce complementary evidence views:

$$Z_{\text{fig}} = A_{\text{fig}}(\mathcal{F}), \quad Z_{\text{con}} = A_{\text{con}}(\tilde{\mathcal{D}}), \quad Z_{\text{alg}} = A_{\text{alg}}(\tilde{\mathcal{D}}), \tag{1}$$

where A_{fig} extracts visual evidence from figures, A_{con} summarizes the problem, method, datasets, metrics, and results, and A_{alg} extracts implementation-critical details such as algorithms, equations, model components, losses, training procedures, hyperparameters, and assumptions. The planning agent then synthesizes these views with user requirements into the initial plan:

$$B_0 = A_{\text{plan}}(Z_{\text{fig}}, Z_{\text{con}}, Z_{\text{alg}}, u). \tag{2}$$

The resulting B_0 specifies the target repository structure, implementation order, core modules, interfaces, data flow, reproduction settings, environment requirements, and validation procedures. Following DeepCode [13], DeepRepro can also retrieve repositories associated with important cited papers as optional auxiliary references. Once planning finishes, B_0 serves as the global specification for the generation loop.

2.2 Repository-aware orchestration

Given the initial plan B_0, DeepRepro initializes the project file tree and progress file, and then enters an agentic execution loop. Each execution round corresponds to one local implementation step: the system selects a small set of related files or repair targets, implements them, updates repository memory, and feeds diagnostics into

the next round. At round t, the execution state is

$$S_t = (B_0, \mathcal{P}_t, \mathcal{M}_t, \Delta_t), \tag{3}$$

where $\mathcal{P}_t$ records completed and remaining files, $\mathcal{M}_t$ is the repository memory, and Δ_t stores diagnostics and repair signals.

Subplanning. During execution, the repository state evolves as files are completed, memory is accumulated, and diagnostics are collected. The subplan agent A_{sub} uses these signals to compute the next subplan:

$$B_t = A_{\text{sub}}(B_0, \mathcal{P}_t, \mathcal{M}_t, \Delta_t). \tag{4}$$

The subplan B_t selects a target set G_t, which may contain new files or files to repair. If $G_t = \emptyset$, the agentic loop stops and DeepRepro finalizes the repository and reproduction report. Otherwise, the selected files are implemented in the current round. The same interface supports two modes: in *fast* mode, B_t only specifies the target file set G_t for efficient generation; in *deepplan* mode, $B_t = (G_t, \rho_t, \pi_t, \mathcal{I}_t, \mathcal{N}_t, \mathcal{Q}_t)$ additionally includes the grouping rationale ρ_t, execution order π_t, per-file instructions $\mathcal{I}_t$, integration notes $\mathcal{N}_t$, and acceptance checks $\mathcal{Q}_t$.

Execution and memory update. Given B_t, the execution agent A_{exec} implements each target file $c_{t,i} \in G_t$ with tool calls. For each file, it uses the file-level context $X_{t,i} = (B_0, B_t, \mathcal{M}_t, c_{t,i})$, where B_0 provides the global plan, B_t provides the current subplan, $\mathcal{M}_t$ provides repository memory, and $c_{t,i}$ identifies the target file. The execute agent generates the file content as $\hat{c}_{t,i} = A_{\text{exec}}(X_{t,i})$ and writes it to the target file $c_{t,i}$. After a successful write, DeepRepro records the file in $\mathcal{P}_{t+1}$ and launches the memory agent A_{mem} for asynchronous compression:

$$m_{t,i} = A_{\text{mem}}(\hat{c}_{t,i}), \qquad \mathcal{M}_{t+1} = \mathcal{M}_t \cup \{m_{t,i}\}_{c_{t,i} \in G_t}. \tag{5}$$

Each memory entry summarizes the file purpose, interfaces, dependencies, and implementation notes. Before the next round, pending memory updates are synchronized so that the next subplan observes the latest repository memory.

Diagnostics and repair. After each round, DeepRepro collects diagnostics for the generated or updated files in G_t as $\Delta_{t+1} = \Delta_t \cup A_{\text{diag}}(G_t)$, including tool errors, invalid writes, syntax errors, dependency issues, and layout problems. These diagnostics serve as repair signals for later subplans. In *deepplan* mode, a bounded quality gate further identifies blocking issues and turns them into localized repair targets for later subplanning and execution.

2.3 Process-aware frontend

DeepRepro provides a process-aware frontend for monitoring the long-horizon generation process. As shown in Figure 1, it receives execution updates from the repository-aware orchestration loop and exposes file status, subplans, diagnostics, memory updates, and execution status, allowing users to inspect progress and identify potential issues during reproduction.

3 Experimental Setup

Dataset and evaluation. We evaluate DeepRepro on PaperBench Code-Dev [19], a code-development subset of PaperBench built from 20 ICML 2024 Spotlight and Oral papers. Following the official Code-Dev evaluation, we skip reproduction execution and grade only the Code Development requirements in the hierarchical rubrics, **with final scores reported as percentages**. We report full-set scores and fixed subset averages when needed; the five-paper subset includes FRE, RICE, PINN, BaM, and Mech-U. For cost control, our main LLM-judge experiments use GPT-4.1-mini; for the human-expert comparison, we additionally use o3-mini for a fairer comparison with the PaperBench judging setup.

Table 1: Overall performance of DeepRepro under different execution modes on the five-paper subset. Highest scores in each column are boldfaced.

Mode	FRE	RICE	PINN	BaM	Mech-U	Avg.
deepplan+ref	74.14	**73.72**	**94.77**	**90.40**	88.36	**84.28**
deepplan	**74.29**	73.49	94.50	90.23	**88.61**	84.22
fast+ref	70.15	71.86	94.05	87.18	87.32	82.11
fast	69.46	72.51	93.95	87.20	87.23	82.07

Table 2: Impact of subplanning and execution models. Highest scores in each column are boldfaced.

Variant	FRE	RICE	PINN	BaM	Mech-U	Avg.
deepplan-5.4	**74.29**	**73.49**	**94.50**	**90.23**	**88.61**	**84.22**
deepplan-5.4+DS	67.18	72.90	92.92	86.10	86.87	81.15
fast-5.4	69.46	72.51	93.95	87.20	87.23	82.07
fast-DS	64.24	68.93	89.31	82.74	82.60	77.56

Baselines. We compare DeepRepro with three groups of baselines: (i) *Scientific code agents*. DeepCode is our main scientific baseline [13], and we also compare with PaperCoder [18] and AutoReproduce [25]; (ii) *Commercial code assistants*. We compare with Cursor and Codex, two general-purpose coding agents widely used in practice [2, 15]; (iii) *Human experts*. We use the PaperBench human baseline [19], where eight ML PhD students or graduates worked part-time on three papers (FRE, All-in-One, and Stay-Topic). We report the best-of-three attempts as the expert-level reference.

Implementation details. Unless otherwise specified, DeepRepro uses GPT-5.4 with high reasoning effort as the backbone model [16]. In model-ablation experiments, the blueprint-planning stage is fixed, and only the subplan and execute models vary. All methods run without extra paper figures or user requirements, and agents do not consult the target paper's official source code. Following prior scientific code agent evaluations, we report both subset and full-set comparisons; some full-set numbers are reused from prior work because of evaluation cost, and we treat them as reference points rather than strict controlled claims.

Model variants. The suffix *+ref* means that external code references are enabled; *deepplan-5.4* uses GPT-5.4 for both subplanning and execution; *deepplan-5.4+DS* uses GPT-5.4 for subplanning and DeepSeek-V4-Flash [5] for execution; and *fast-5.4*/*fast-DS* use *fast* mode with GPT-5.4/DeepSeek-V4-Flash execution.

4 Experimental Results

Overall performance. Table 1 reports DeepRepro under four execution settings. We find: (i) external code references have only marginal impact on this subset; (ii) the *deepplan* mode consistently outperforms *fast*, improving the average score from 82.07 to 84.22.

Effect of subplanning and model choices. Table 2 further analyzes how subplanning interacts with different execution models. We find that *deepplan* consistently improves performance across both strong and weaker executors, and the gain becomes more

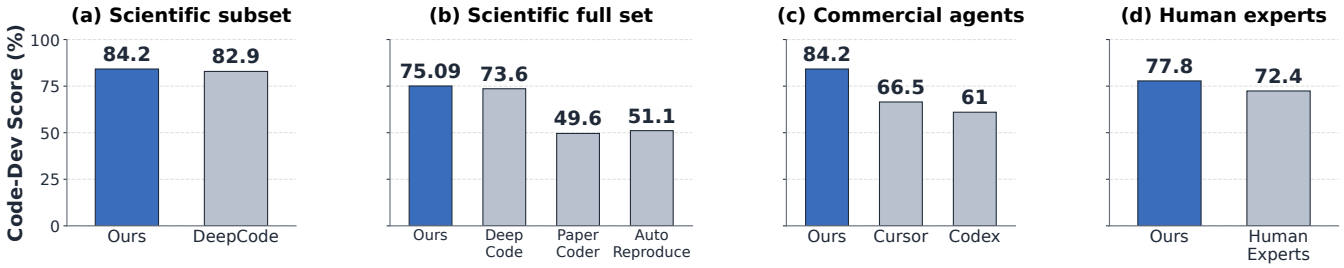


Figure 2: Comparative Code-Dev results: (a) scientific-agent subset comparison, (b) full-set scientific-agent comparison, (c) commercial-agent comparison, and (d) human-expert comparison. DeepRepro is shown in blue and all baselines are shown in gray. Panel (b) includes full-set baselines from prior work and should not be interpreted as a strict comparison.

Table 3: Full-set DeepRepro scores on 20 PaperBench papers.

Paper	Score	Paper	Score	Paper	Score
FRE	74.29	RICE	73.49	PINN	94.50
Will-For.	74.09	BaM	90.23	All-in-One	76.62
Adapt. Prune	56.60	LBCS	83.39	Mech-U	88.61
TT Adapt.	65.13	Sample Spec.	86.70	Bridge Gap	61.55
Stay-Topic	77.80	Stochastic	81.25	LCA-Line	78.37
Seq. Neural	87.44	SAPG	60.22	FTRL	41.91
Robust-CLIP	82.96	BBox	66.59	Avg.	**75.09**

pronounced when the execution model is weaker. This indicates that subplanning plays a complementary role by compensating for execution-level limitations and improving global coordination over long-horizon tasks.

Full-set evaluation. Table 3 reports results on all 20 Code-Dev papers. DeepRepro achieves consistent performance across diverse tasks, with an overall average score of 75.09, further demonstrating robustness beyond the controlled five-paper setting.

Comparative experiments. Figure 2 summarizes the comparison results: (i) *Scientific code agents.* DeepRepro outperforms DeepCode on the controlled five-paper subset (Fig. 2(a)) and shows the same positive trend in the 20-paper setting (Fig. 2 (b)), indicating robust gains across evaluation scales; (ii) *Commercial code assistants.* As shown in Fig. 2 (c), DeepRepro surpasses Cursor and Codex under identical evaluation protocols on the five-paper subset; (iii) *Human experts.* Fig. 2 (d) shows that DeepRepro exceeds the PaperBench Best@3 human baseline on the shared subset.

Runtime and cost analysis. We further report approximate runtime and cost for practical reference. These numbers are not strictly controlled but reflect real deployment behavior. In *fast* mode DeepRepro incurs similar costs to DeepCode (about US$10±1 per paper), while the *deepplan* mode increases costs moderately to around US$12.5±1. In terms of runtime, DeepRepro significantly improves efficiency: *fast* typically completes within one hour, while *deepplan* finishes in approximately 1.5 hours, compared to nearly two hours for DeepCode. Overall, *fast* is suitable for quick reproduction, while *deepplan* is preferred for higher-fidelity results.

5 Related Work

General coding agents. LLM-based coding agents have evolved from code completion to autonomous software engineering [6, 7, 10]. Prior work explores role-based collaboration and workflow decomposition [11, 17], repository-level editing and testing [23], and issue resolution through search, localization, task graphs, and iterative validation [3, 24]. These capabilities have also been adopted by IDE- and terminal-based coding agents [1, 2, 4, 8, 9, 15, 21, 22]. Scientific reproduction is more demanding than general coding because agents must translate research contributions and experimental methodology into executable implementations [18].

Scientific coding agents. Scientific coding agents generate executable research artifacts from papers or experimental specifications. Paper-to-code systems have been evaluated through model-based assessment, human evaluation, and comparison with author repositories, while PaperBench standardizes evaluation using 20 ICML 2024 papers and hierarchical reproduction rubrics [18, 19]. Recent approaches include analysis–planning–generation pipelines [18], paper-lineage-based reproduction [25], autonomous research loops [12, 14, 20], and planning with memory and verification [13].

6 Conclusion

We have presented DeepRepro, an automatic framework for ML paper-to-code reproduction that treats repository construction as a state-aware, iterative process. DeepRepro combines blueprint planning with execution-state-aware round-level subplanning over evolving repositories, and integrates asynchronous memory compression, bounded repair, and a process-aware interface for long-horizon execution and monitoring. Experiments on PaperBench Code-Dev demonstrate that DeepRepro consistently outperforms strong scientific and commercial code-agent baselines.

Acknowledgements

This work was funded by the Strategic Priority Research Program of the CAS under Grants No. XDB0680102, the National Natural Science Foundation of China (NSFC) under Grants No. 62472408, U25B2076 and 62441229, the National Key Research and Development Program of China under Grants No. 2023YFA1011602. This research was also (partially) supported by the Dutch Research Council (NWO), under project numbers 024.004.022, NWA.1389.20.183, and KICH3.LTP.20.006, and the European Union under grant agreement No. 101201510 (UNITE).

All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

GenAI Usage Disclosure

We used generative AI tools to assist with grammar checking and language polishing during manuscript preparation.

References

[1] Anthropic. 2025. Claude Code: AI Coding Agent, Terminal, IDE. https://claude.com/product/claude-code. Accessed: 2026-05-19.

[2] Anysphere. 2025. Cursor: The Best Way to Code with AI. https://cursor.com. Accessed: 2026-05-19.

[3] Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. 2024. CodeR: Issue Resolving with Multi-Agent and Task Graphs. arXiv:2406.01304 [cs.CL] https://arxiv.org/abs/2406.01304

[4] Cline. 2025. Cline: AI Coding, Open Source and Uncompromised. https://cline.bot. Accessed: 2026-05-19.

[5] DeepSeek-AI. 2026. DeepSeek V4 Preview Release. https://api-docs.deepseek.com/news/news260424. Accessed: 2026-05-27.

[6] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A Survey on Code Generation with LLM-based Agents. arXiv:2508.00083 [cs.SE] https://arxiv.org/abs/2508.00083

[7] Yuyao Ge, Lingrui Mei, Zenghao Duan, Tianhao Li, Yujia Zheng, Yiwei Wang, Lexin Wang, Jiayu Yao, Tianyu Liu, Yujun Cai, Baolong Bi, Fangda Guo, Jiafeng Guo, Shenghua Liu, and Xueqi Cheng. 2025. A Survey of Vibe Coding with Large Language Models. arXiv:2510.12399 [cs.AI] https://arxiv.org/abs/2510.12399

[8] GitHub. 2025. GitHub Copilot: The AI Assistant That Builds with You. https://github.com/copilot. Accessed: 2026-05-19.

[9] Google. 2025. Gemini CLI: An Open-Source AI Agent. https://github.com/google-gemini/gemini-cli. Accessed: 2026-05-19.

[10] Jiale Guo, Suizhi Huang, Mei Li, Dong Huang, Xingsheng Chen, Regina Zhang, Zhijiang Guo, Han Yu, Siu-Ming Yiu, Pietro Lio, and Kwok-Yan Lam. 2025. A Comprehensive Survey on Benchmarks and Solutions in Software Engineering of LLM-Empowered Agentic System. arXiv:2510.09721 [cs.SE] https://arxiv.org/abs/2510.09721

[11] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] https://arxiv.org/abs/2308.00352

[12] Andrej Karpathy. 2026. AutoResearch: AI Agents Running Research on Nanochat. https://github.com/karpathy/autoresearch. Accessed: 2026-05-19.

[13] Zongwei Li, Zhonghang Li, Zirui Guo, Xubin Ren, and Chao Huang. 2025. DeepCode: Open Agentic Coding. arXiv:2512.07921 [cs] doi:10.48550/arXiv.2512.07921

[14] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery. arXiv:2408.06292 [cs.AI] https://arxiv.org/abs/2408.06292

[15] OpenAI. 2025. Codex CLI: A Coding Agent from OpenAI. https://github.com/openai/codex. Accessed: 2026-05-19.

[16] OpenAI. 2026. OpenAI API Models. https://developers.openai.com/api/docs/models. Accessed: 2026-05-27.

[17] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. arXiv:2307.07924 [cs.SE] https://arxiv.org/abs/2307.07924

[18] Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2026. Paper2Code: Automating Code Generation from Scientific Papers in Machine Learning. arXiv:2504.17192 [cs.CL] https://arxiv.org/abs/2504.17192

[19] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. 2025. PaperBench: Evaluating AI's Ability to Replicate AI Research. arXiv:2504.01848 [cs.AI] https://arxiv.org/abs/2504.01848

[20] Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. 2025. AI-researcher: Autonomous Scientific Innovation. In *Advances in Neural Information Processing Systems* (2025), D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (Eds.), Vol. 38. Curran Associates, Inc., 9481–9520. https://proceedings.neurips.cc/paper_files/paper/2025/file/0d904d300a105809a2114d727851e759-Paper-Conference.pdf

[21] TRAE. 2025. TRAE: Collaborate with Intelligence. https://www.trae.ai. Accessed: 2026-05-19.

[22] Windsurf. 2025. Windsurf: The Best AI for Coding. https://windsurf.com. Accessed: 2026-05-19.

[23] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems* (2024), A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 50528–50652. doi:10.52202/079017-1601

[24] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1592–1604. doi:10.1145/3650212.3680384

[25] Xuanle Zhao, Zilin Sang, Yuxuan Li, Qi Shi, Weilun Zhao, Shuo Wang, Duzhen Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. 2026. AutoReproduce: Automatic AI Experiment Reproduction with Paper Lineage. arXiv:2505.20662 [cs.AI] https://arxiv.org/abs/2505.20662