Deep-Reproducer: From Paper Understanding to Code Generation

Pengcheng Chen[2*] **Ning Yan**[1] **Zihan Zhao**[3] **Yixiao Lin**[4] **Huaibo Chen**[5]
Yue Hu[6] **Qinbo Bai**[7] **Xiang Li**[7] **Masood S. Mortazavi**[1†]

[1]Futurewei Inc [2]University of Washington [3]University of California, San Diego
[4]Cornell University [5]Massachusetts Institute of Technology [6]University of Southern California
[7]Purdue University

Abstract

Recent Large Language Models (LLMs) demonstrate strong code generation capabilities, however, they often fall short in translating complex, multi-component research methodologies into a coherent, functional codebase and automatic repository-level code synthesis from research papers remains a formidable challenge. Despite promising results from current paper reproduction agents, particularly their efficiency in generating code repositories from scratch, their reliance on staged prompt engineering falls short for complex implementation tasks. In this paper, we propose a multi-agent framework for automated paper reproduction, leveraging a combination of deep research mechanisms, a long-short term memory architecture, and modular generation strategies driven by Large Language Models (LLMs). Our system employs a structured workflow where specialized agents autonomously decompose complex implementation tasks into manageable subtasks, thereby facilitating efficient and scalable code synthesis. The experimental evaluation on PaperBench demonstrates the state-of-the-art performance in the implementation of automated research papers, achieving a Replication Score of 63.2%.

1 Introduction

With rapid LLM advancements, coding capabilities have improved significantly, outperforming humans on SWE benchmarks [6], LeetCode [7], and IOI tasks [5]. However, in domains requiring deep domain expertise and systematic engineering—such as paper reproduction—LLMs still lag due to the non-linear complexity of real-world workflows [3, 4, 15]. Solving highly abstract tasks usually requires more than simple prompts, as these often lack the necessary context and structure. Coding agents [19, 8, 13] address this by decomposing projects into manageable subtasks, allowing LLMs to tackle low-complexity components sequentially, which leads to better performance than standalone models. PaperBench[14], introduced by OpenAI, provides a benchmark for evaluating LLMs' abilities to replicate research papers, where agents must reproduce 20 ICML 2024 papers from scratch–understanding contributions, developing codebases, and running experiments–using only paper text. PaperBench measures LLMs' agentic research skills across 8,316 gradable tasks with rubrics co-developed with authors. It uses an LLM-based SimpleJudge [14] that achieves an F1 score of 0.83 against human grading. Recently, multi-agent approaches have been developed to automate the generation of executable code from research papers. AutoP2C[8] processes papers through blueprint extraction, content parsing, task decomposition, and iterative debugging. AutoReproduce[19] extracts

*Email: pengcc@uw.edu
†Corresponding author: mmortaza@futurewei.com

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Deep Learning for Code in the Agentic Era (DL4C) @ NeurIPS 2025.

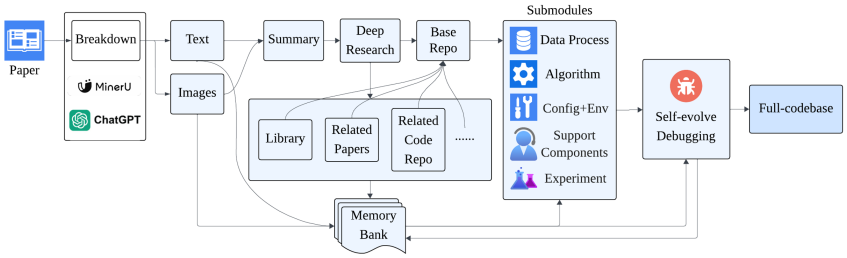

Figure 1: **The Deep-Reproducer pipeline.** From paper breakdown to deep research, the system populates a Memory Bank that guides modular code generation with self-correction, producing a complete executable codebase.

implicit knowledge from citations using literature review, paper lineage, and code development. PaperCoder[13] operates through planning, analysis, and generation stages, showing superior performance on PaperBench. However, these approaches have some critical limitations. First, they fail to conduct comprehensive research beyond paper content and citations, missing opportunities to leverage suitable existing repositories and implementations from the Web domain. Second, they lack memory management systems, relying solely on LLM context windows without maintaining persistent research findings and debugging history. Third, their debugging pipelines follow simple linear loops without effectively utilizing historical debugging experiences for iterative improvement.

To address these limitations, we propose **Deep-Reproducer**, a novel multi-agent framework that introduces three key innovations: (1) We integrate deep research that searches across the entire Web to identify the most suitable, extensible, or widely-used repositories as base implementations, going beyond the original paper citations. (2) We develop multi-agent memory bank management that records and maintains deep research findings as a temporary external database, enabling more efficient and accurate data retrieval than direct API calls. (3) We design a self-evolving debugging pipeline that provides more sophisticated code correction mechanisms. Our approach achieves the state-of-the-art performance on PaperBench-Dev, reaching **55.7%** replication score with o3-mini and **63.2%** with GPT-5, substantially outperforming all existing methods.

2 Methods

Deep-Reproducer, as shown in Figure 1, begins with a comprehensive paper analysis phase that extracts and organizes multimodal content from research papers. We employ MinerU [16] combined with `GPT-4o` [10] for PDF content extraction and JSON standardization, generating indexed text JSON files alongside corresponding images. This structured content is stored in a memory bank, creating a searchable repository of paper components. Following content extraction, we utilize `GPT-5` [11] to perform multi-aspect summarization of the PDF content and images. The summarization process focuses on six critical dimensions: (1) Data requirements – identifying datasets, data formats, and preprocessing needs; (2) Methodological approaches – extracting algorithmic details and implementation strategies; (3) Task objectives – clarifying the research goals and evaluation metrics; (4) Research methodology – understanding the experimental design and validation approach; (5) Related work – identifying similar approaches and comparative studies; and (6) Foundational methods – determining the base techniques and prior work that the paper builds upon. Through multiple rounds of summarization, we generate comprehensive search objectives that capture the paper's essential requirements for reproduction.

2.1 Deep Research Integration

The output from the summarization stage is consumed by a deep research module, which executes Web-scale retrieval to identify relevant components for reproduction. This module could employ an existing LLM-powered deep research engine (e.g., `GPT-5 deep research`) or specialized agents (e.g., Miroflow [9]) to conduct targeted searches across the Internet, while explicitly excluding original paper repositories and unofficial implementations to ensure independent reproduction. Specifically, our deep research process identifies the following essential components: (1) Datasets and download links – locating official datasets, preprocessing scripts, and data access procedures. (2) High-relevance

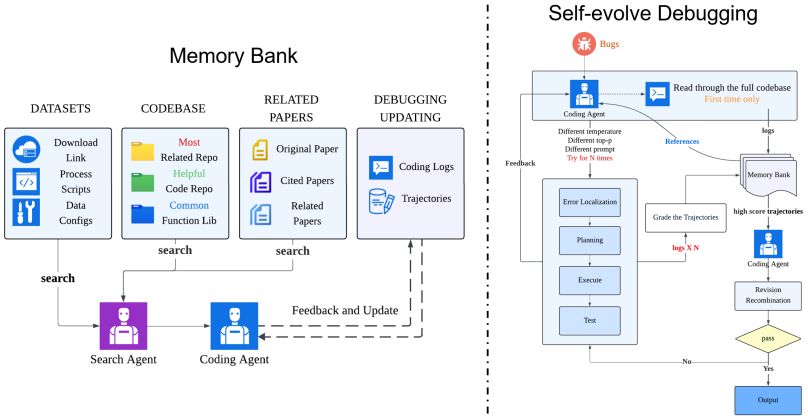

Figure 2: **Memory Bank architecture and Self-Evolve Debugging mechanism.** The Memory Bank organizes data into four categories accessed by a Search Agent. The debugging pipeline iteratively refines code through error localization, planning, and testing, with trajectory optimization via revision and recombination operators.

code repositories – finding well-maintained, widely-used implementations of similar methodologies. (3) Algorithm-specific implementations – discovering existing code libraries for specific algorithmic steps described in the paper. (4) Contemporary work and codebases – identifying concurrent research with available implementations. (5) Research paradigm implementations – locating repositories that follow similar research methodologies and experimental designs. These components provide the foundational building blocks required for faithful and reproducible implementation

2.2 Memory Bank Organization

The **Memory Bank**, as shown in Figure 2 (left), is a lightweight, task-specific external store that aggregates *Deep-Research hits* and *agent-side debugging artifacts*. It keeps pointers and minimal summaries rather than mirroring full code, so that retrieval remains precise and low-overhead. We organize input data entries into four categories: (1) Datasets: download links, preprocessing notes/scripts, and configuration snippets (sizes, splits, environment dependencies). (2) Code pointers: URLs to potentially relevant repositories/pages, suspected file/function paths (if available), and short method notes for features referenced by the target paper; common *open-source libraries* are recorded as package names and pinned versions. (3) Related papers: the target paper and cited works used as methodological references or for specification checks. (4) Debugging memory: prompts, failure traces, applied patches, and concise commit-style logs that support the self-evolving debugging loop. Each entry carries minimal metadata: title, 1–2 line summary, license, version/tag/commit, and timestamp.

Unlike the traditional RAG agents [18, 17, 2], which rely on an unfiltered database, our lightweight Memory Bank stores only pointers and short notes filtered from the results of Deep Research, not full code or full paper, for higher efficiency and to save the context memory of LLM. The Coding Agent interacts with the Memory Bank under the assistance of the Searching Agent to guide code generation and re-implementation. The Search Agent, built on `GPT-5`, first queries the Memory Bank (keyword/fuzzy matching over metadata) and, through the Model Context Protocol (MCP) [1], can escalate to web search for verification and gap-filling, returning compressed hits—URLs, file paths, commit hashes, and concise usage notes—for context-aware reuse. We discuss more details of Memory Bank design issues in the **Supplementary Materials** due to space limitations.

2.3 Self-Evolving Debugging

Top-K related repos are ranked by the deep research model with deterministic tie-breaking based on compliance and licenses, and the Top-1 related repo were selected as the base code for reproduction. Inspired by the SE-Agent [12], Deep-Reproducer employs a self-evolving debugging process, as shown in Figure 2 (right). Rather than simply retry after failures, each generation runs dual-track trajectories under fixed decoding (temperature 0.1, top-$p = 1$, fixed seeds) that execute `locate` $\rightarrow$ `patch` $\rightarrow$ `execute` $\rightarrow$ `test`. For every step, build/test outcomes, traces, diffs, and one-line LLM

Table 1: The evaluation results of **Deep-Reproducer (Ours)** on PaperBench Code-dev benchmarks. The AutoP2C, AutoReproduce and PaperCoder are the original implementations from submission.

Baselines	Coding LLM	Judge	Replication Score (%)
AutoP2C	o3-mini	o3-mini-high	49.2
AutoReproduce	o3-mini	o3-mini-high	49.6
PaperCoder	o3-mini	o3-mini-high	45.1
Ours	o3-mini	o3-mini-high	55.7
Ours (w/o self-evolve debugging)	o3-mini	o3-mini-high	51.9
Ours	GPT-5	o3-mini-high	**63.2**

rationales are written as compact trajectory cards and log snippets into the Memory Bank (Debugging Memory) with metadata (timestamp, commit hash, error signature, judge score). This yields a replay buffer that supports reuse and cross-trajectory recombination. We assign a conservative fitness prioritizing task success with light regularization: $\mathrm{Fit}(\tau) = 1.00 * \mathrm{PassAll} + 0.60 * \mathrm{PassFrac} + 0.25 * \mathrm{Build} - 0.25 * \widehat{\mathrm{Err}} - 0.10 * \widehat{\mathrm{Act}} - 0.05 * \widehat{\mathrm{Tok}}$, with costs normalized to $[0, 1]$. We use an evolutionary search strategy: the higher-scoring trajectory is always kept, with ties favoring shorter ones. New candidates are generated by revising failed steps using Memory-Bank feedback, or recombining high-quality fragments from different trajectories. Offspring are re-executed, and improvements replace their parents. We stop on success (all checks pass), no improvement for T generations ($\delta = 0.02$), or budget limits (steps/tokens/time).

3 Experiments

Evaluation Protocol & Compliance. We evaluate Deep-Reproducer on PaperBench Code-Dev under the official protocol. Unless otherwise noted, the agent receives only the paper PDF (plus clarifications) and must *build a runnable repository from scratch*. The target paper's official or unofficial implementations are never viewed or used. To enforce the "from-scratch" constraint, we apply the official PaperBench blacklist verbatim (commit `<hash>` / release `<tag>`): all tool calls and resolved URLs are checked via an offline log sweep, and any hit is discarded. As a final safeguard, we rebuild using only `reproduce.sh` and verify that no external assets are fetched at grading time. For transparency, we release the exact blacklist file we used and the minimal checker script.
Judge Alignment & Reproducibility. All results are graded with SimpleJudge [14] using the official PaperBench scaffold and default prompts/hyperparameters. Because LLM-based judging is stochastic, we run the grade three independent times per submission with fixed seeds and report the mean across runs. Unless otherwise specified, all leaderboard claims refer to this SimpleJudge configuration.

We summarize *Replication Score* results in Table 1. Our method attains $\mathbf{55.7}\%$ with an `o3-mini` coding backbone and a fixed `o3-mini-high` judge, surpassing the strongest reproduced baseline AutoReproduce (49.6%) by $\Delta = 6.1\%$ points, corresponding to a relative gain of 12.3%. Compared to PaperCoder (45.1%), the absolute improvement is 10.6% points with a relative gain of 23.5%. To verify the impact of the self-evolving debugging loop, we removed this component and used the traditional "Error localization $\rightarrow$ Planning $\rightarrow$ Generation $\rightarrow$ Implementation" loop, which is similar to AutoP2C. The results show that performance drops from 55.7% to 51.9%, making it only slightly outperform previous work—a relative decrease of 6.8%. These results demonstrate the effectiveness of the self-evolving debugging loop. Using a stronger coding backbone (`GPT-5`) while keeping the judge fixed to `o3-mini-high` yields $\mathbf{63.2}\%$, an absolute gain of 7.5 over `o3-mini` and a relative improvement of 13.5%. All results are generated using a consistent judging setup (`o3-mini-high`) and the official execution/verification protocol, with no custom scoring heuristics.

4 Conclusion

We presented *Deep-Reproducer*, a multi-agent framework integrating Web-scale deep research, a task-specific Memory Bank, and a self-evolving debugging pipeline. Our work achieves strong performance on PaperBench-Dev while revealing a clear path to further gains via lower-friction knowledge acquisition (API-ready sources, standardized metadata), cheaper tool-use policies, and stronger trajectory optimization. These components are broadly applicable beyond AI-paper reproduction,

pointing toward reliable, auditable, and cost-conscious paper-to-code systems. The deep-research stage currently incurs a notable human-in-the-loop burden: many high-value sources (dataset portals, institutional Websites, older GitHub releases) lack stable APIs. One future improvement is to automate targeted Web search and manual curation before ingesting artifacts into the Memory Bank.

5 Acknowledgments

We thank Linda Zhang and Dr. Peng for their invaluable assistance with server infrastructure, repository management, and general project support. We also thank Zongfang Lin for his insightful comments during our group discussions. This work was supported by Futurewei Technologies, with Dr. Liang Peng serving as the funding manager.

References

[1] Anthropic. Introducing the model context protocol (mcp). https://www.anthropic.com/news/model-context-protocol, 2024. Accessed: 2025-08-25.

[2] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair. *arXiv preprint arXiv:2403.17134*, 2024. doi: 10.48550/arXiv.2403.17134. URL https://arxiv.org/abs/2403.17134.

[3] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1978. ISBN 0201006502.

[4] Leila Hedayatifar, Alfredo J. Morales, Dominic E. Saadi, Rachel A. Rigg, Olha Buchel, Amir Akhavan, Egemen Sert, Aabir Abubaker Kar, Mehrzad Sasanpour, Irving R. Epstein, and Yaneer Bar-Yam. Predicting system dynamics of universal growth patterns in complex systems, 2025. URL https://arxiv.org/abs/2501.07349.

[5] IOI Foundation. International olympiad in informatics (ioi). https://ioinformatics.org/, 2025. Accessed: 2025-08-25.

[6] Carlos E. Jimenez, Tim Rocktäschel, Sean Welleck, et al. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.06770. OpenReview: VTF8yNQM66.

[7] LeetCode. Leetcode weekly contests. https://leetcode.com/contest/, 2025. Accessed: 2025-08-25.

[8] Zijie Lin, Yiqing Shen, Qilin Cai, He Sun, Jinrui Zhou, and Mingjun Xiao. Autop2c: An LLM-based agent framework for code repository generation from multimodal content in academic papers. *arXiv preprint arXiv:2504.20115*, 2025. doi: 10.48550/arXiv.2504.20115. URL https://arxiv.org/abs/2504.20115.

[9] MiroMind AI. Miroflow. https://github.com/MiroMindAI/MiroFlow, 2025. GitHub repository. Accessed: 2025-08-25.

[10] OpenAI. Hello GPT–4o. https://openai.com/index/hello-gpt-4o/, 2024. Accessed: 2025-08-25.

[11] OpenAI. Introducing GPT–5. https://openai.com/index/introducing-gpt-5/, 2025. Accessed: 2025-08-25.

[12] Boyuan Peng, Zhenyu Tang, Jixuan Zhang, Zhu Li, Jun Xie, Tiezheng Chi, Yilun Du, Shuyan Zhou, Yanan Zheng, Yinpeng Chen, Junxian He, Asli Celikyilmaz, Kyunghyun Cho, and Yue Zhang. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. *arXiv preprint arXiv:2503.21466*, 2025. URL https://arxiv.org/abs/2503.21466.

[13] Minju Seo, Seungone Lee, Hyunwoo Jo, Jinwoo Kim, Min Jun Seo, Doheon Lee, Hanseok Park, Hyunggi Park, Sungrae Park, Sihyung Kim, Dong-Wan Yoo, Minsu Cho, and Gunhee Kim. Paper2code: Paper-centric code generation for research reproduction. *arXiv preprint arXiv:2504.17192*, 2025. URL https://arxiv.org/abs/2504.17192.

[14] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Mia Glaese, Tejal Patwardhan, and OpenAI. Paperbench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025. URL https://arxiv.org/abs/2504.01848.

[15] Stefan Thurner, Peter Klimek, and Rudolf Hanel. *Introduction to the Theory of Complex Systems*. Oxford University Press, 09 2018. ISBN 9780198821939. doi: 10.1093/oso/9780198821939.001.0001. URL https://doi.org/10.1093/oso/9780198821939.001.0001.

[16] Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, Bo Zhang, Liqun Wei, Zhihao Sui, Wei Li, Botian Shi, Yu Qiao, Dahua Lin, and Conghui He. Mineru: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839*, 2024. doi: 10.48550/arXiv.2409.18839. URL https://arxiv.org/abs/2409.18839.

[17] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13643–13658, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.737. URL https://aclanthology.org/2024.acl-long.737/.

[18] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24)*, pages 1592–1604, Vienna, Austria, 2024. ACM. doi: 10.1145/3650212.3680384.

[19] Xuanle Zhao, Zilin Sang, Yuxuan Li, Qi Shi, Weilun Zhao, Shuo Wang, Duzhen Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. Autoreproduce: Automatic AI experiment reproduction with paper lineage. *arXiv preprint arXiv:2505.20662*, 2025. doi: 10.48550/arXiv.2505.20662. URL https://arxiv.org/abs/2505.20662.