Self-Guided Planning and Repair Framework for Code Generation (Student Abstract)

Chun-Wei Kang,[1] **Chung-Chi Chen,**[2] **An-Zi Yen**[1]

[1]National Yang Ming Chiao Tung University, Taiwan
[2]Artificial Intelligence Research Center, AIST, Japan
nick020789.cs13@nycu.edu.tw, c.c.chen@acm.org, azyen@nycu.edu.tw

Abstract

Large Language Models (LLMs) demonstrate strong capabilities in code generation but often lack adaptability in planning and refinement. We propose **Self-PR**, a framework that integrates *adaptive plan selection* and *iterative repair* to improve correctness and generalization. Self-PR constructs a reusable plan database via task clustering and trains a selector to choose task-specific strategies. Incorrect outputs are refined through multi-round feedback until correctness. Trained only on HumanEval, Self-PR generalizes well to out-of-distribution tasks (MBPP), improving pass@1 by **+4.9% on HumanEval** and **+5.5% on MBPP** compared to Modularization-of-Thought prompting. Experiments across Llama-3 (8B, 70B) and GPT-4o-mini confirm robustness and scalability. These findings suggest that adaptive planning and feedback-driven repair are essential for reliable LLM-based code generation.

Code — https://github.com/wei020789/Self-PR

Introduction

Large Language Models (LLMs) have achieved remarkable success in code generation tasks, enabling applications in automated programming, bug fixing, and software engineering workflows. Despite these advances, generating functionally correct and logically consistent code remains challenging, especially in scenarios requiring multi-step reasoning or task-specific strategies. Conventional prompting methods, such as zero-shot and few-shot prompting (Chen et al. 2021), rely on static templates, which often fail to capture the diversity of programming tasks. As a result, LLMs frequently produce incomplete solutions or incorrect implementations that require manual intervention.

Recent research has introduced structured reasoning approaches such as Chain-of-Thought (CoT) prompting (Wei et al. 2023), which encourages step-by-step reasoning before code generation, and Modularization-of-Thought (MoT) prompting (Pan and Zhang 2025), which decomposes tasks into hierarchical subtasks. While these methods improve reasoning, they assume a single fixed planning style for all problems, limiting adaptability. Different categories of tasks, such as mathematical computation, data manipulation, or string processing, require distinct reasoning patterns and error-handling mechanisms. Without the ability to dynamically select an appropriate strategy, LLMs risk generating brittle solutions that fail under distributional shifts or edge cases. In addition to planning, robustness against initial errors is critical. Self-repair techniques (Olausson et al. 2024) aim to address this by iteratively refining incorrect outputs. However, most existing repair methods operate on a single initial solution, which may already be misaligned with the problem's requirements. This limitation significantly reduces their effectiveness, particularly for complex or ambiguous tasks.

To overcome these challenges, we propose **Self-PR**, a self-guided planning and repair framework designed to integrate strategy diversity with adaptive error recovery. Unlike prior approaches, Self-PR constructs a reusable Plan Database, learns a selector to identify the most suitable strategy for each task, and employs an iterative repair mechanism to correct errors. By combining these components, Self-PR not only improves first-attempt accuracy but also enhances resilience under challenging conditions, achieving strong generalization to out-of-distribution tasks without additional fine-tuning.

Proposed Framework

Self-PR consists of three components, illustrated in Figure 1:

1. Strategic Plan Constructor. This module builds a reusable *Plan Database* to capture diverse problem-solving strategies. We embed task descriptions into a semantic vector space and apply K-means clustering to group similar problems together. For each cluster, an LLM generates step-by-step solution plans. These plans act as structured guides for future tasks that share semantic similarity. By storing multiple strategies, the system supports adaptive reuse instead of a fixed, one-size-fits-all prompt.

2. Selector Module. The selector automatically identifies the most suitable plan for a new task, minimizing reliance on manual prompt engineering. We fine-tune the selector using a preference-based loss on labeled pairs of "good" and "bad" plans, determined by execution success. At inference, the selector ranks all candidate plans and retrieves the highest-scoring one. This chosen plan is prepended to the

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

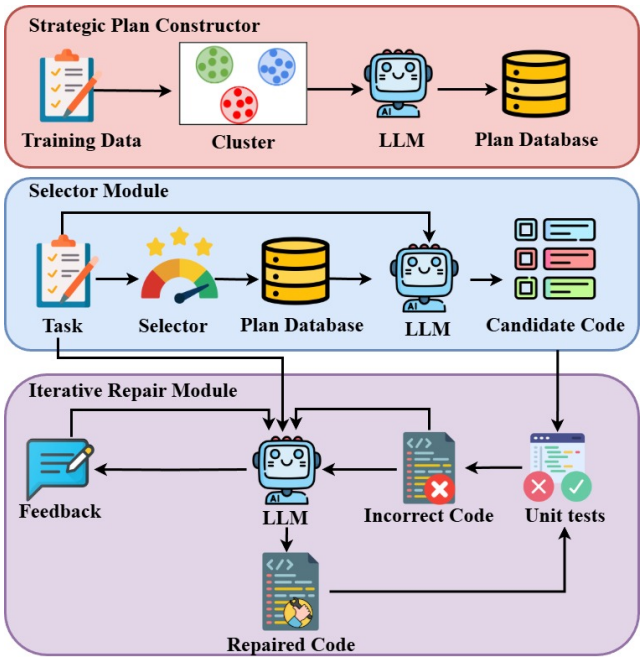

Figure 1: Overview of the Self-PR framework. It includes a Strategic Plan Constructor, a Selector Module, and an Iterative Repair Module.

task description to form an enriched prompt. Unlike heuristic similarity-based retrieval, the learned selector accounts for both semantic closeness and empirical plan effectiveness, ensuring robust performance across diverse scenarios.

3. Iterative Repair Module. Even with a strong initial plan, generated code can fail due to logic errors or overlooked edge cases. The repair module introduces an adaptive feedback loop that analyzes failures and regenerates improved solutions. Upon encountering a failing unit test, the LLM diagnoses the issue, explains the root cause, and proposes a fix in natural language. This feedback is combined with the original task and previous code to generate a revised implementation. The process repeats for up to 6 iterations or until all tests pass. This iterative mechanism significantly enhances reliability and reduces trial-and-error cycles.

Results

We compare Self-PR against established prompting baselines using GPT-4o-mini. Table 1 summarizes pass@1 results on HumanEval (in-domain) and MBPP (OOD). Self-PR achieves the highest accuracy across both benchmarks. Compared to the strongest baseline (MoT), Self-PR delivers an absolute gain of +4.9% on HumanEval and +5.5% on MBPP, highlighting its effectiveness and robust generalization across both in-domain and out-of-distribution settings. The ablation results further show that removing the selector leads to a moderate drop in accuracy, while removing the repair module causes a larger decline, especially on MBPP. This indicates that both adaptive planning and iterative repair are essential and complementary for maximizing performance.

Method	HumanEval	MBPP
Zero-shot (Chen et al. 2021)	88.4	59.9
Few-shot (Chen et al. 2021)	82.3	49.1
CoT (Wei et al. 2023)	87.8	61.2
Self-planning (Jiang et al. 2024)	87.2	52.1
SCoT (Li et al. 2023)	86.6	63.9
CodeCoT (Huang et al. 2024)	83.5	55.6
MoT (Pan and Zhang 2025)	92.1	73.9
Self-PR (Ours)	**97.0**	**79.4**
Self-PR (w/o Selector)	**95.1 (-1.9%)**	**76.7 (-2.7%)**
Self-PR (w/o Repair)	**93.3 (-3.7%)**	**72.0 (-7.4%)**

Table 1: Pass@1 (%) on HumanEval and MBPP for various approaches, using GPT-4o-mini as the base model. Scores in parentheses indicate the relative gap from Self-PR.

Future Work

In future work, we plan to extend Self-PR in several directions. One avenue is dynamic plan synthesis, where the system generates new plans on-the-fly for tasks that deviate significantly from existing clusters. Another is cross-domain and multilingual adaptation, enabling the framework to support additional programming languages and specialized domains such as data science or embedded systems. We also aim to reduce the computational overhead introduced by iterative repair through more efficient feedback generation or selective repair triggers. Ultimately, we envision Self-PR as a foundation for autonomous programming agents that can plan, adapt, and self-correct in real-world software development environments.

Acknowledgements

This research was partially supported by National Science and Technology Council, Taiwan, under grant NSTC 114-2221-E-A49-057-MY3.

References

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Huang, D.; Bu, Q.; Qing, Y.; and Cui, H. 2024. CodeCoT: Tackling Code Syntax Errors in CoT Reasoning for Code Generation. arXiv:2308.08784.

Jiang, X.; Dong, Y.; Wang, L.; Fang, Z.; Shang, Q.; Li, G.; Jin, Z.; and Jiao, W. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7): 1–30.

Li, J.; Li, G.; Li, Y.; and Jin, Z. 2023. Structured Chain-of-Thought Prompting for Code Generation. arXiv:2305.06599.

Olausson, T. X.; Inala, J. P.; Wang, C.; Gao, J.; and Solar-Lezama, A. 2024. Is Self-Repair a Silver Bullet for Code Generation? In *ICLR*.

Pan, R.; and Zhang, H. 2025. Modularization is Better: Effective Code Generation with Modular Prompting. arXiv:2503.12483.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.