PairCoder: A Multi-Agent Framework for Code Generation

Taylan Bapur, Jan Härtrich

Summer 2025 – Seminar on Machine Learning in Software Engineering

Overview

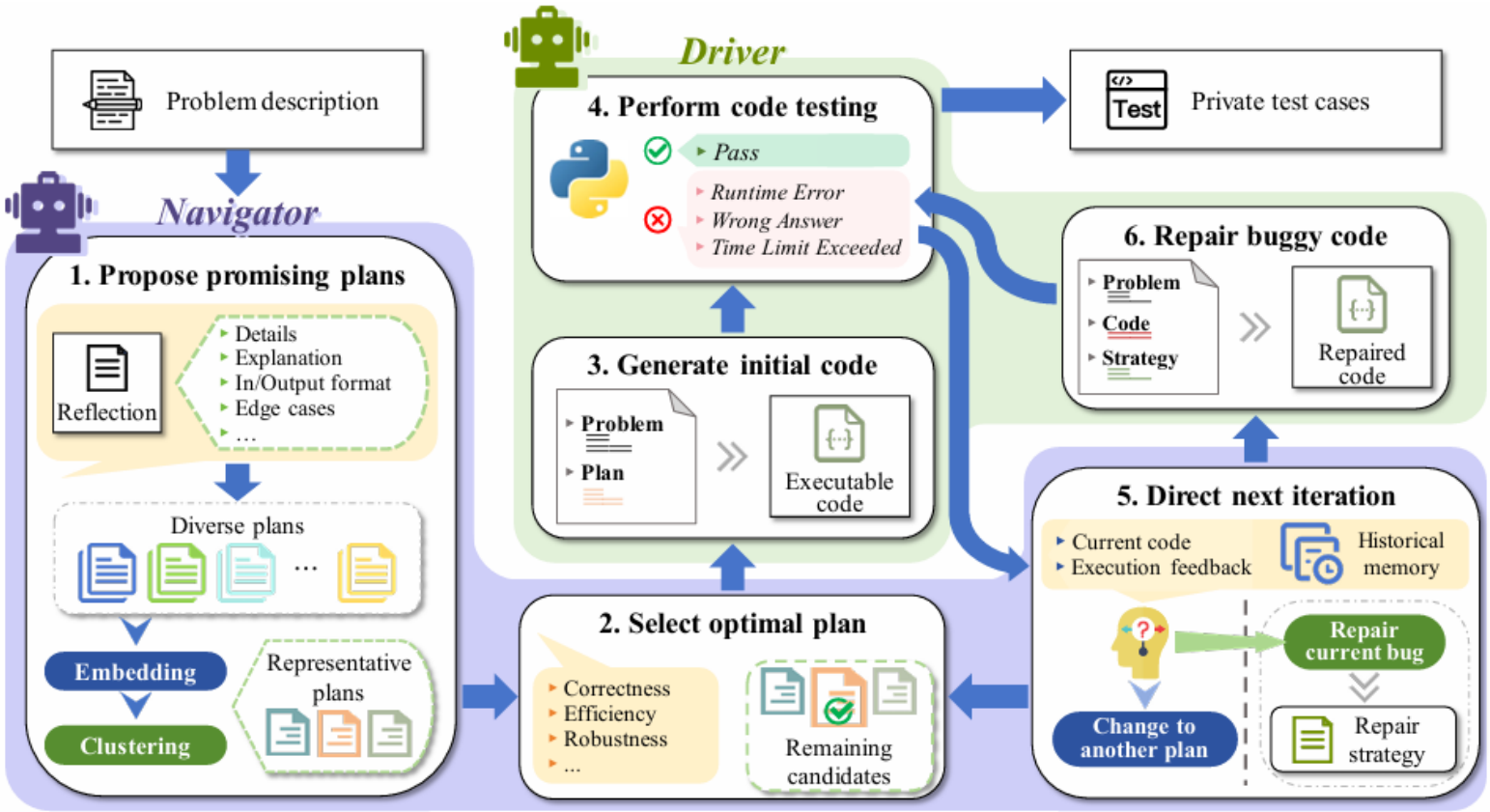
- Code generation with Large Language Models (LLMs) is improving, but challenges remain for complex tasks.
- **PairCoder** is a novel multi-agent system inspired by human pair programming.
- Compares favorably to Guided Code Generation and MapCoder.

Motivation

- Single-agent LLMs struggle with:
  - Complex reasoning
  - Iterative debugging
  - Backtracking and adaptation
- Multi-agent systems aim to mitigate these issues by delegating roles.

PairCoder Framework Overview

- Two agents: **Navigator** (planning) and **Driver** (execution).
- Iterative feedback loop until code passes test cases or max iterations.



Navigator Agent

- Reflects on problem, input/output format, edge cases.
- Proposes multiple diverse solution plans.
- Uses semantic embeddings and k-means++ clustering.
- Tracks history to avoid redundant strategies.

Driver Agent

- Stateless and reactive to Navigator's instructions.
- Implements selected plan into executable code.
- Tests code on public test cases.
- Applies repair strategies if needed.

Core Techniques

Multi-Plan Exploration

- Diverse plans: brute force, greedy, DP, etc.
- Clustering avoids redundancy.
- Evaluation by correctness, efficiency, robustness.

Feedback-Driven Refinement

- Results: Pass, Wrong Answer, Runtime Error, Time Limit Exceeded.
- Navigator chooses to repair or switch plans.
- Leverages historical memory.

Evaluation Benchmarks

- **Benchmarks:** HumanEval, MBPP, CodeContest
- **Models:** GPT-3.5-Turbo, DeepSeek-Coder
- Measured using `pass@1` (pass all private tests on first try).

Performance Results

Approaches	GPT-3.5-Turbo						DeepSeek-Coder					
	HumanEval		MBPP		CodeContest		HumanEval		MBPP		CodeContest	
	Orig	Plus	Orig	Plus	Valid	Test	Orig	Plus	Orig	Plus	Valid	Test
Direct prompting	67.68	60.98	66.80	66.42	6.84	6.06	76.22	67.78	66.40	64.41	5.98	6.67
CoT prompting	68.90	62.80	69.00	67.17	5.13	5.45	77.27	68.90	67.60	67.67	5.45	6.67
SCoT prompting	68.29	61.59	62.60	61.40	5.98	4.24	73.17	65.85	61.80	60.15	4.27	6.06
Self-planning	72.56	64.63	69.60	67.67	7.69	6.06	74.39	68.90	65.80	68.17	6.84	9.09
Self-collaboration†	74.40	-	68.20	-	-	-	-	-	-	-	-	-
Self-repair	73.17	65.24	70.60	69.42	7.69	6.67	77.44	70.12	70.00	70.43	5.98	7.27
Self-debugging	78.05	72.56	72.80	70.43	9.40	11.52	79.27	73.78	72.20	71.12	8.55	13.33
INTERVENOR	77.44	69.51	73.40	71.93	8.55	9.09	79.88	72.56	72.60	72.43	7.69	10.30
Reflexion	69.57	-	67.76	-	-	-	81.99	-	74.31	-	-	-
LDB†	82.90	-	76.00	-	-	-	-	-	-	-	-	-
PairCoder (ours)	**87.80**	**77.44**	**80.60**	**77.69**	**17.95**	**15.15**	**85.37**	**76.22**	**78.80**	**75.69**	**13.68**	**14.55**
Relative improvement ↑	29.73%	26.99%	20.66%	16.97%	162.43%	150.00%	12.00%	12.45%	18.67%	17.51%	128.76%	118.14%

- PairCoder outperforms all baselines.
- Up to **87.80%** pass@1 on HumanEval with GPT-3.5.
- Up to **162.43%** improvement over prompting baselines.

Ablation Studies

- **Without multi-plan:** 6–8% accuracy drop.
- **Without feedback refinement:** 10–12% drop.
- Confirms both components are crucial.

Comparison: Guided Code Generation

- Hierarchical planning tree using Generalist, Code, and Tester agents.
- Modular and interpretable.
- Strong on compositional logic, but complex to manage.
- PairCoder is better for fast, flexible tasks.

Comparison: MapCoder

- 4 agents: Retrieval, Planning, Coding, Debugging.
- Simulates full software development lifecycle.
- Adds plan scoring and algorithm tutorials.
- Strong modularity but dependent on high-quality examples.

Threats to Validity

- Dataset leakage (e.g., HumanEval overlap)
- Prompt sensitivity and model size bias
- Benchmark limitations (single-function only)
- Pass@1 does not reflect partial correctness
- Reproducibility challenges noted in GitHub testing

Example: Bug Fixing and Plan Refinement

Original buggy implementation:

```
def solve_buggy(arr):
    # Bug: Possible IndexError due to arr[i+1] access
    operations = 0
    for i in range(len(arr)):
        if arr[i] > i + 1 and arr[i+1] < arr[i]:
            operations += 1
    return operations
```

Example: Bug Fixing and Plan Refinement

Refined (corrected) version:

```
def solve_fixed(arr):
    arr = arr[:]
    i = 0
    ops = 0
    while i < len(arr):
        if arr[i] > i + 1:
            arr.insert(i, i + 1)
            ops += 1
        else:
            i += 1
    return ops
```

Discussion and Limitations

- PairCoder mimics human problem-solving effectively.
- Relies heavily on good initial plan diversity.
- Computationally more expensive than single-pass methods.
- Real-world usability depends on further optimization and robustness.

Future Work

- Combine PairCoder's refinement with Guided Code's hierarchy.
- Human-in-the-loop selection and interpretation.
- Cross-domain application (e.g., theorem proving).
- Dynamic test suite generation.

Conclusion

- Multi-agent systems like PairCoder outperform single-agent LLMs.
- Iterative refinement and plan diversity are key strengths.
- Hybrid systems may offer even better performance.
- Continued work needed on implementation, reproducibility, and scalability.