# Genetic Programming and Language Features in Solving Competition-Level Software Engineering Problems

Assignee Research

June 4, 2026

**Abstract**

This report synthesises findings from 16 peer-reviewed papers addressing the following research question: What techniques enable language models to solve competition-level software engineering problems. 13 claims were extracted from source literature; 4 were independently verified against retrieved documents. An automated multi-reviewer quality assessment produced a score of 4.7/10. This report is a machine-generated literature synthesis and does not constitute original research.

## 1 Introduction

This paper examines: Code Building Genetic Programming. Research question: What techniques enable language models to solve competition-level software engineering problems.

## 2 Methodology

Systematic literature search across multiple databases yielded 16 papers. Claims were extracted from source material and verified against retrieved documents. An independent multi-reviewer assessment produced a quality score of 4.7/10.

## 3 Results

16 papers retrieved. 13 claims extracted; 4 independently verified. Quality review score: 4.7/10.

## 4 Limitations

This report is a machine-generated literature synthesis and does not constitute original research. Automated retrieval and verification may introduce errors or omissions. Review scores reflect automated assessment, not human peer review. Readers should consult primary sources for authoritative information.

## 5 Extracted Claims

| Claim | Verified | Confidence |
|---|---|---|
| CBGP assigns penalty errors to individuals if the Push compilation process does not produce a program DAG that returns t | × | 0.05 |
| CBGP assigns penalty errors to individuals if a program DAG raises exceptions during run-time. | × | 0.02 |
| CBGP program DAGs only need to be compiled once and can then be evaluated on all training cases. | × | 0.03 |
| PushGP requires a separate execution of the stack-based interpreter for every training case. | × | 0.03 |
| The computation graph representation of the programs produced during CBGP are much faster to compute than the stack-base | × | 0.05 |
| CBGP has the potential to dramatically reduce the cost of evolution for program synthesis tasks. | × | 0.08 |
| Program DAGs synthesized by Code Building Genetic Programming are roughly analogous to abstract syntax trees. | ✓ | 0.17 |
| Using knowledge of the host-language's syntax, it is possible to produce source code from a program DAG in CBGP. | × | 0.09 |
| Program DAGs evolved by CBGP contain expressions that wrap existing functions and methods. | × | 0.05 |
| CBGP was configured to use 31 runs, 300 generations, a population size of 1000, Lexicase selection, UMAD variation, 100 | × | 0.03 |
| CBGP leverages programming language features such as reflection and first-class specifications. | ✓ | 0.20 |
| CBGP produces a computational graph that can be executed or translated into source code of a host language. | ✓ | 0.26 |
| CBGP can synthesize programs that use arbitrary data types, data structures, and specifications drawn from existing code | ✓ | 0.34 |

## References


- `http://arxiv.org/abs/2303.12869v1`

- http://arxiv.org/abs/2008.03649v1
- http://arxiv.org/abs/1206.3111v1