Large Language Models and Grammar-Guided Genetic Programming for Complex Code Generation on HumanEval

Assignee Research

June 4, 2026

Abstract

This report synthesises findings from 13 peer-reviewed papers addressing the following research question: How do large language models compare to Grammar Guided Genetic Programming in solving code generation tasks involving complex, overlapping data structures on the HumanEval benchmark. 18 claims were extracted from source literature; 2 were independently verified against retrieved documents. An automated multi-reviewer quality assessment produced a score of 4.5/10. 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: How do large language models compare to Grammar Guided Genetic Programming in solving code generation tasks involving complex, overlapping data structures on the HumanEval benchmark?.

2 Methodology

Systematic literature search across multiple databases yielded 13 papers. Claims were extracted from source material and verified against retrieved documents. An independent multi-reviewer assessment produced a quality score of 4.5/10.

3 Results

13 papers retrieved. 18 claims extracted; 2 independently verified. Quality review score: 4.5/10.

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
In Code Building Genetic Programming (CBGP), once an individual's genome is translated into Push code and compiled into	×	0.14
If the Push compilation process in CBGP does not produce a program DAG that returns the required type for the given prob	×	0.04
In CBGP, if a program DAG raises an exception during run-time evaluation on a training case, the exception is caught and	×	0.02
In CBGP, program DAGs only need to be compiled once before the individual can be evaluated on all training cases.	×	0.02
PushGP requires a separate execution of the stack-based interpreter for every training case.	×	0.02
The computation graph representation of programs produced during CBGP is faster to compute than the stack-based executio	×	0.04
Program DAGs synthesized by CBGP are roughly analogous to abstract syntax trees.	×	0.02
Source code can be produced from a CBGP program DAG using knowledge of the host-language's syntax.	×	0.09
Program DAGs evolved by CBGP contain expressions that wrap existing functions and methods.	×	0.04
The CBGP system was configured to run 31 runs per benchmark problem.	×	0.01
The CBGP system was configured to run for 300 generations per benchmark problem.	×	0.01
The CBGP system was configured with a population size of 1000.	×	0.02
The CBGP system used Lexicase selection.	×	0.02
The CBGP system used UMAD variation.	×	0.05
The CBGP system used 100 random training cases per benchmark problem.	×	0.02
The CBGP system used 1000 test cases per benchmark problem.	×	0.02
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

References

- http://arxiv.org/abs/2306.08568v2
- http://arxiv.org/abs/2008.03649v1
- http://arxiv.org/abs/2510.21391v1