Mistral-Large-2 Scaling Effects on MBPP Code Generation and Evaluation Metrics

Assignee Research

May 30, 2026

Abstract

This report synthesises findings from 15 peer-reviewed papers addressing the following research question: How does the performance of Mistral-Large-2 in generating code solutions on MBPP scale with model size, and how does this scaling affect both functional correctness and human evaluation scores. Although large language models (LLMs) have been largely successful in generating functionally correct programs, conditioning models to produce efficient solutions while ensuring correctness remains a challenge. Further, unreliability in benchmarking code efficiency is a hurdle. 13 claims were extracted from source literature; 2 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: ECCO: Can We Improve Model-Generated Code Efficiency Without Sacrificing Functional Correctness?. Research question: How does the performance of Mistral-Large-2 in generating code solutions on MBPP scale with model size, and how does this scaling affect both functional correctness and human evaluation scores?.

2 Methodology

Systematic literature search across multiple databases yielded 15 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

15 papers retrieved. 13 claims extracted; 2 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
ECCO supports two optimization paradigms: (i) history-based code editing and (ii) NL-based code generation.	✓	0.24
ECCO collects over 50k Python solution pairs, spanning 1.3k competitive programming problems, with an average of 3.1 pub	×	0.03
ECCO uses JUDGE0, a cloud-hosted code execution engine, for reliable and reproducible executions.	×	0.06
JUDGE0 supports up to 66 programming languages.	×	0.06
Spurious optimization can reduce runtime but add errors that cause the program to be incorrect.	×	0.02
A true optimization reduces runtime and remains correct.	×	0.03
Execution information and fine-tuning help maintain functional correctness.	✓	0.22
NL-involved prompting often yields higher efficiency improvements.	×	0.06
The linear search algorithm has a runtime of 500 ms and passes the test.	×	0.01
The spurious binary search optimization has a runtime of 120 ms but fails the test.	×	0.01
The correct binary search optimization has a runtime of 150 ms and passes the test.	×	0.01
The execution results for the linear search algorithm on Test1: [1, 2], 1 are , 200ms, 172 KB.	×	0.02
The execution results for the linear search algorithm on Test2: [5, 6, 7], 1 are , 200ms, 172 KB.	×	0.02

References

- http://arxiv.org/abs/2401.13565v3
- http://arxiv.org/abs/2407.14044v2
- http://arxiv.org/abs/2310.06825v1