How does the pass@k metric for code generation models vary across BigCodeBench tasks requiring multi-library P

Assignee Research

May 29, 2026

Abstract

Repeated sampling with a verifier is the standard way to allocate test-time compute for code generation, with pass@K as the canonical metric. Yet the standard policy class draws K independent samples from a single answer distribution, so attempts often collapse onto near-duplicate reasoning paths and waste the budget on redundant rollouts. This failure is costly in competitive programming, where many problems admit multiple distinct algorithmic strategies and pass@K requires only one correct attempt. We propose Coordinated Pass@K Policy Optimization (CPPO), which turns pass@K generat

1 Introduction

This paper examines: Cast a Wider Net: Coordinated Pass@K Policy Optimization for Code Reasoning. Research question: How does the pass@k metric for code generation models vary across BigCodeBench tasks requiring multi-library Python data science workflows compared to single-library tasks?.

2 Methodology

Systematic literature search across multiple databases yielded 4 papers. Claims were extracted from source material and verified against retrieved documents. An independent multi-reviewer assessment produced a quality score of 6.5/10.

3 Results

4 papers retrieved. 0 claims extracted; 0 independently verified. Quality review score: 6.5/10.

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.

References

- http://arxiv.org/abs/2303.12869v1
- http://arxiv.org/abs/2605.27000v1
- http://arxiv.org/abs/2510.08325v2