Reproducibility Report: DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation

Ifeanyi Agu
Zakaria Bouzada
Kai Larinen

Abstract

This report critically examines the reproducibility of the paper *DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation* by Lai et al. The authors propose DS-1000, a benchmark featuring 1000 realistic data science problems curated from StackOverflow. Key contributions include:

1. Problems representing real-world coding scenarios.
2. A multi-criteria evaluation framework ensuring solution correctness and efficiency.
3. Measures to mitigate memorization risks in models.

This report evaluates the reproducibility of the methodology, analysis, and results, while also identifying challenges and proposing enhancements for broader adoption.

Introduction

Modern advancements in code generation aim to reduce programming barriers, especially in domains like data science that require domain-specific expertise. Benchmarks play a crucial role in evaluating these systems, but existing ones often fail to capture realistic scenarios or lack robust evaluation metrics.

The authors address these gaps with DS-1000, emphasizing its use of:

- Diverse, realistic problem sets.
- Reliable evaluation metrics combining functional correctness and surface-form constraints.
- Modifications to defend against memorization by models pre-trained on web data.

This reproducibility report critically analyzes the feasibility of replicating their contributions and validating their findings.

Benchmark Construction

Problem Sourcing

The problems in DS-1000 were derived from StackOverflow under seven library tags (e.g., NumPy, Pandas). High-quality problems were selected based on votes, views, and accepted answers. Further refinements included filtering duplicates, stratifying by creation year, and curating 451 base problems for expansion.

Challenges in Replication:

1. **StackOverflow Metadata**: Access to problem creation timestamps, votes, and views is critical but may vary depending on scraping tools and API limitations.
2. **Scoring Criteria**: While the scoring rubric (e.g., representativeness, usefulness) is clear, annotator bias could affect reproducibility.

Problem Modification

To align with DS-1000's requirements, problems were rewritten for clarity and unambiguity. The process involved:

- Adding executable contexts, such as pre-imported libraries and variable initialization.
- Rewriting Matplotlib problems to exclude visual content.
- Fixing inaccuracies in high-vote StackOverflow solutions.

Strengths:

- The step-by-step methodology for problem preparation is detailed and replicable.
- Adjustments to ensure consistency across library versions are forward-thinking.

Limitations:

- Context rewriting and reference solution creation rely heavily on expert judgment, requiring annotators with domain expertise.

Evaluation Framework

Multi-Criteria Evaluation

DS-1000 introduces a novel evaluation framework combining:

1. **Functional Correctness**: Solutions are tested against input-output test cases.
2. **Surface-Form Constraints**: Checks ensure solutions adhere to specified efficiency standards (e.g., avoiding loops).

The test cases are manually curated and validated, with edge cases handled using statistical methods, such as the Kolmogorov-Smirnov test for randomness.

Key Challenges

1. **Floating-Point Arithmetic**: Variability in floating-point results could lead to false negatives in equivalence checks.
2. **Subjectivity in Constraints**: The thresholds for "efficient" implementations depend on subjective criteria, which may vary across replicators.

Replication Feasibility: The framework is well-documented and includes illustrative examples, making it feasible to reimplement. However, expertise in handling statistical tests and edge cases is necessary for accurate replication.

Perturbation for Memorization Defense

Categories of Perturbations

The authors employed two types of perturbations to modify the problems:

1. **Surface Perturbations**: Paraphrasing problem descriptions, changing input-output examples, etc., without altering the underlying logic.
2. **Semantic Perturbations**: Modifying problem requirements, such as replacing "min" with "max" or changing the result type.

Experimental Results

The authors demonstrated that perturbations effectively reduce model accuracy, e.g., Codex-002's performance dropped from 72.5% on the unmodified numpy-100 dataset to 40.6% after perturbation.

Strengths

- The perturbation taxonomy is detailed, with concrete examples to guide replication.
- Semantic perturbations add depth, requiring models to demonstrate generalization beyond memorized solutions.

Limitations

- The perturbation process is labor-intensive, requiring annotators to rewrite problems consistently while maintaining test case integrity.

Replication Feasibility: Recreating perturbations is feasible with sufficient manual effort. Automating parts of this process could improve scalability.

Benchmarking Models

Evaluated Models

The authors benchmarked five pre-trained code generation models, including Codex (three variants), CodeGen, and InCoder. Codex-002 achieved the highest accuracy (43.3%), while others performed significantly worse.

Prompt Formats

Two formats were tested:

1. **Insertion Format**: Includes both left and right contexts for infilling tasks.
2. **Completion Format**: Uses left context only, with right context provided as instructions.

Results showed that the insertion format consistently improved performance, highlighting the importance of bidirectional context in code generation.

Key Insights:

1. **Accuracy Variations Across Libraries**: Model performance varied significantly across libraries, suggesting a need for holistic benchmarks.
2. **Room for Improvement**: Even the best-performing model achieved <50% accuracy, underscoring the benchmark's difficulty.

Replication Challenges

- Proprietary Access: Codex models are not publicly available, limiting external reproducibility.
- Controlled Environment: Maintaining library versions and configurations identical to the original study is critical for fair comparisons.

Results Analysis

Perturbation Impact

The results convincingly demonstrate that DS-1000's perturbations reduce reliance on memorization:

- Accuracy drops were consistent across surface and semantic perturbations.
- The difficult rewrites introduced significant challenges, further lowering model accuracy.

Evaluation Metrics

The evaluation framework was robust, with only 1.8% false positives and 0.5% false negatives among predictions deemed correct/incorrect. These metrics validate DS-1000's reliability.

Limitations

1. **Expert Dependency**: Many steps, including problem modification and evaluation, require significant expertise, limiting replicability without skilled annotators.
2. **Proprietary Tools**: Codex models are proprietary, restricting their use in independent studies.
3. **Labor-Intensive Process**: Perturbation and evaluation implementation are time-consuming, posing scalability challenges.

Recommendations for Future Work

1. **Automated Tools**: Develop semi-automated systems for perturbation and evaluation to reduce manual effort.
2. **Open Models**: Encourage benchmarking using open-source models to enhance reproducibility.
3. **Expanded Libraries**: Include additional libraries and domains to increase DS-1000's coverage and utility.

Conclusion

DS-1000 represents a significant advance in benchmarking code generation models for data science. Its emphasis on realistic problems, robust evaluation, and memorization defense sets a high standard for future benchmarks. While the methodology is reproducible with sufficient expertise, scalability and accessibility improvements could further enhance its impact.

This report concludes that DS-1000 is a valuable resource for evaluating code generation systems, offering meaningful insights into their capabilities and limitations.