Adaptive Theorem Proving Benchmarks for Robust LLM Evaluation

Will Thomas, Logan Schmalz, Bryan Richlinski, and Jack Reynolds

Institute for Information Sciences
The University of Kansas
Lawrence, KS 66045
{30wthomas, loganschmalz, b748r023, jackreynolds}@ku.edu

Abstract. The application of Large Language Models (LLMs) to complex reasoning tasks, particularly in the domain of formal methods and theorem proving, holds significant promise. However, evaluating the true deductive capabilities of these models is challenging due to their potential exposure to existing benchmarks during training, leading to issues of memorization rather than genuine understanding. This project addresses this challenge by developing and evaluating a dynamic benchmark suite for the Coq proof assistant. We systematically generate test cases by applying semantic-preserving transformations—specifically, the renaming of identifiers—to theorems sourced from the well-known *Logical Foundations* dataset.

The performance of two contemporary LLMs, `llama3.1` and `phi4`, was assessed by tasking them to generate proofs for both original and transformed theorems. Proof attempts were first evaluated for structural usability and then compiled using the Coq compiler (`coqc`) to verify correctness.

This approach aims to determine the extent to which LLM performance is affected by superficial syntactic changes, thereby providing insights into their reliance on learned surface patterns versus deeper semantic reasoning. This work contributes a methodology for creating more robust benchmarks for LLM-based theorem proving and offers initial findings on the sensitivity of modern LLMs to such perturbations.

Keywords: LLMs · Theorem Proving · Coq · Benchmarking · Formal Methods

1 Introduction

1.1 Background and Motivation

Large Language Models (LLMs) have demonstrated remarkable capabilities across a diverse array of natural language processing tasks and are increasingly being explored for their potential in specialized domains requiring complex reasoning, such as software development, mathematics, and formal verification. Within the realm of computer science, formal methods provide rigorous, mathematically-grounded techniques for specifying, developing, and verifying software and hardware systems. Theorem provers, such as the Coq proof assistant, are pivotal tools in formal methods, enabling users to construct machine-checked proofs of correctness. The prospect of leveraging LLMs to assist or even automate aspects of the formal proof development process in Coq could significantly lower the barrier to entry for formal methods and enhance productivity for experienced users.

However, the effective integration of LLMs into formal reasoning workflows necessitates a clear understanding of their actual deductive capabilities, as distinct from their ability to recall or replicate patterns seen during training. As LLMs are trained on vast swathes of publicly available text and code, including potentially many formal proofs and tutorials, their performance on standard benchmarks may not accurately reflect true generalization or reasoning prowess.

1.2 Problem Statement

A critical issue in evaluating LLMs for theorem proving is the risk of benchmark contamination or "memorization." Many established benchmarks for formal methods consist of static sets of problems that may have been part of the LLMs' training data. Consequently, an LLM might successfully "solve" a benchmark problem not by deducing the solution, but by recalling it, or parts of it, from its training set. This phenomenon makes it difficult to:

1. Reliably compare the true reasoning abilities of different LLMs.
2. Ascertain whether LLMs genuinely "understand" the underlying semantics of formal statements and proof strategies.
3. Gauge their robustness when faced with novel-looking but semantically equivalent problems.

There is, therefore, a pressing need for evaluation methodologies that can mitigate the effects of data leakage and assess an LLM's ability to generalize its reasoning capabilities beyond an exact replication of previously encountered examples.

1.3 Proposed Solution and Contributions

This project addresses the aforementioned challenges by developing and evaluating a dynamic benchmark generation strategy for Coq theorem proving tasks. The core idea is to create new test cases by applying programmatic transformations to existing, well-understood theorems, specifically those from the widely recognized *Logical Foundations* series by Pierce et al. [1]. These transformations are designed to alter the syntactic appearance of the theorems while preserving their underlying logical semantics and provability. In this study, the primary transformation implemented is the systematic renaming of identifiers (such as variables and local definitions) within the Coq theorem statements and their contexts.

The experimental methodology involved the following steps:

1. **Data Collection:** Coq files were sourced from the *Logical Foundations* dataset.
2. **Theorem Mutation:** A transformation process was applied to these files to automatically rename identifiers, creating a parallel set of mutated (perturbed) files.
3. **Task Generation:** From both the original and mutated files, individual theorem statements were extracted.
4. **LLM Evaluation:** Two contemporary LLMs, `llama3.1` and `phi4`, were prompted to generate proofs for each extracted theorem (both original and its renamed counterpart).
5. **Verification:** The LLM-generated outputs were first assessed for structural usability (i.e., whether they formed a coherent Coq proof script). Usable scripts were then compiled using the Coq compiler (`coqc`) to rigorously verify their correctness.

By comparing the LLMs' success rates and the nature of their outputs on original versus renamed theorems, this study aims to shed light on their sensitivity to superficial syntactic variations. A significant drop in performance on renamed theorems would suggest a reliance on memorized surface patterns, whereas comparable performance would indicate a more robust, semantic understanding.

The **primary contributions** of this work are:

- An implemented methodology and tool for generating perturbed Coq theorem proving tasks based on identifier renaming, aimed at reducing benchmark leakage in LLM evaluations.
- An empirical evaluation of two modern LLMs (`llama3.1` and `phi4`) on this dynamic benchmark, providing initial data on their performance and consistency when faced with original and syntactically varied theorems.
- Insights into the current capabilities and limitations of LLMs in the context of formal theorem proving, specifically concerning their ability to generalize beyond surface-level syntax.

This research represents a step towards developing more authentic and reliable assessments of LLM reasoning abilities in the demanding domain of formal methods.

1.4 Report Outline

The remainder of this report is structured as follows:

- **Chapter 2 (Background and Related Work):** Discusses relevant literature on LLMs in code and formal reasoning, existing Coq benchmarks, program transformation techniques, and challenges in LLM evaluation.
- **Chapter 3 (Dataset Collection):** Details the data corpus, its significance in the Coq community, and the process of extracting theorems from the *Logical Foundations* dataset.
- **Chapter 4 (Methodology):** Explains the design of the theorem transformation engine focusing on identifier renaming, and the overall benchmark generation process, as well as the LLM prompting strategy.
- **Chapter 5 (Experimental Setup):** Describes the LLMs tested, the precise task definition, the evaluation metrics employed (usability and Coq compilation), and the experimental environment.
- **Chapter 6 (Results):** Presents the quantitative and qualitative findings from the experiments, comparing LLM performance on original versus perturbed theorems.
- **Chapter 7 (Conclusion and Future Work):** Summarizes the key contributions and suggests avenues for future research in this area.

2 Related Work

3 Dataset

4 Methodology

5 Experiment

6 Results

7 Conclusion

Bibliography

[1] Pierce, B.C., Casinghino, C., Greenberg, M., Sjöberg, V., Yorgey, B.: Logical Foundations, Software Foundations, vol. 1. Electronic textbook, version 6.4 edn. (January 2024)