LLM4Docq: Bootstrapping Documentation for MathComp with LLMs and Expert Feedback

Théo Stoskopf* Jules Viennot† Cyril Cohen*

Introduction

The **MathComp** [1] library of the Rocq Prover contains thousands of definitions and lemmas, yet many currently lack detailed docstrings or explanatory comments [4, 14]. This scarcity of documentation creates steep learning curves for newcomers and makes it difficult to locate the right lemmas or understand formal statements [3, 15]. Recent advances in large language models (LLMs) offer a promising solution: LLMs have demonstrated an ability to generate code documentation and even assist with formal proofs in interactive theorem provers [10, 2, 7, 8, 9]. However, directly applying general-purpose LLMs to Rocq code is non-trivial—the scarcity of training data presents unique challenges [4, 12]. Addressing this gap requires *domain-specific training data* and a strategy to align LLM outputs with expert knowledge [6].

In this extended abstract, we present **LLM4Docq**, a project that aims to use LLMs and human feedback to augment MathComp with docstrings [10]. Our approach focuses on two key goals. First, we aim to improve *user accessibility* to the MathComp library by enabling natural language retrieval of formal content [3]. Second, we fine-tune an off-the-shelf LLM for Rocq—a model trained on Rocq code paired with docstrings—for tasks such as code auto-completion, documentation generation, and even auto-formalization [11]. This strategy combines a *user-centric application* (search by natural language) with *deep learning research* (fine-tuning on a specialized corpus). In the following, we describe our two-step methodology for dataset creation.

Methodology: Human-in-the-Loop Documentation

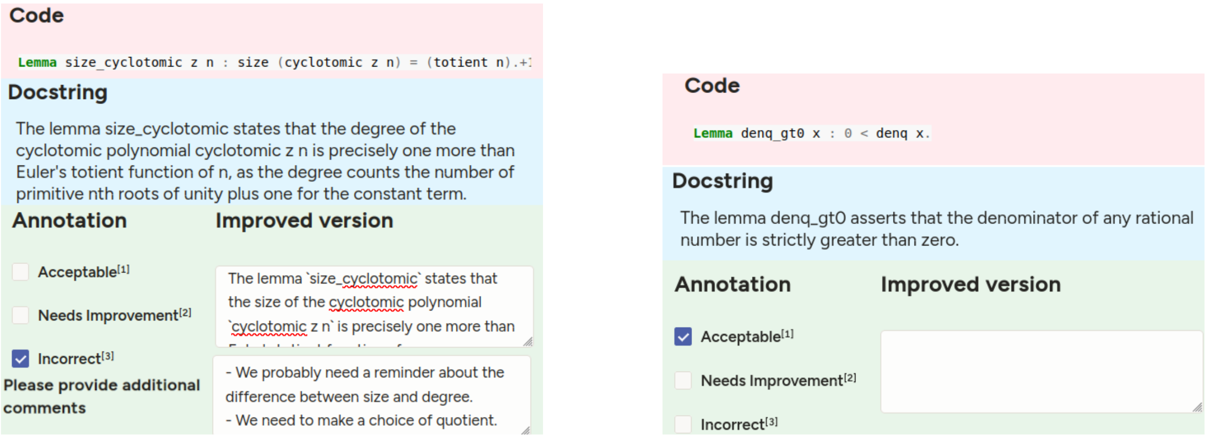
Our process consists of an **iterative human-in-the-loop annotation pipeline** to efficiently bootstrap a large docstring dataset [13]. In **Step 1**, an LLM is used to *automatically generate initial docstrings* for every definition, lemma, theorem, etc., in MathComp (over 30,000 items) [10]. While modern models can produce plausible documentation, outputs vary in quality—some descriptions are accurate, others may be incomplete or even incorrect [15]. Rather than fully trusting these generations, we plan to incorporate *expert review* to refine them [6].

In **Step 2**, human experts will *review and provide feedback* on a subset of the LLM-generated docstrings. Using a collaborative annotation platform [13], each candidate docstring is labeled as **Acceptable**, **Needs Improvement**, or **Incorrect**, along with corrections or suggestions when applicable (See Figure 1 for examples of LLM-generated docstrings, illustrating both a typical failure and a success case as displayed in the collaborative annotation platform.). Instead of doing a full audit of the dataset, we leverage feedback from partial auditing to *regenerate or improve* the remaining docstrings. For any entries annotated as "Needs Improvement" or "Incorrect" corresponding to some systemic issue, we will update the LLM prompts with additional

*Inria, CNRS, ENS de Lyon, LIP, UMR 5668, France
†Inria, CNRS, Université Paris Cité, IRIF, UMR 8243, France

instructions or examples derived from the expert suggestions, and have the LLM produce new versions. This targeted re-generation uses the *latest feedback as guidance*, allowing the LLM to avoid past mistakes and align with human-preferred style and accuracy [6]. We plan to iterate this process to progressively converge on a high-quality documentation corpus with minimal human effort on each round.

Our human-in-the-loop approach is inspired by alignment techniques like reinforcement learning from human feedback, where models learn from preference data to better satisfy user intent [6]. However, instead of learning a reward model, we apply feedback *directly as constraints and examples* for the next generation cycle. By leveraging LLM suggestions as a starting point and expert knowledge for correction, we expect to rapidly produce a comprehensive set of docstrings that would have otherwise required more time to author manually [15].



(a) Failure case

(b) Success case

Figure 1: Illustration of LLM-generated docstrings: (a) a failure case and (b) a success case. Both cases are shown as they appear on the collaborative annotation platform.

LLM Fine-Tuning on MathComp augmented with Docstrings

The curated dataset of MathComp code annotated with reviewed docstrings enables us to train a **domain-specific LLM** for Rocq [4, 10]. We are fine-tuning a code-oriented model (Qwen 2.5 32b coder base [5]) on our augmented codebase, effectively performing continued pre-training on Rocq's formal language enriched with natural language explanations [7, 8, 2]. This technique—*training on source code together with documentation*—has precedent in software engineering: fine-tuning an LLM on code-comment pairs can greatly improve its ability to generate docstrings and understand code intent [14]. In our case, the model is encouraged to associate Rocq definitions/lemmas with their natural-language descriptions [7, 8].

Recent work has combined fine-tuned LLMs on large Rocq corpora, such as CoqStoq, with retrieval-augmented generation for automatic proof synthesis [12]. While Rango focuses on proof generation, it validates the strategy of domain-specific fine-tuning on formal code. Other systems, such as *RocqStar*, also aim to improve the proof search in Rocq using augmented retrieval techniques [16].

We plan to evaluate the final model on two complementary tasks:

- **Docstring Generation (Formal → NL):** Given a formal MathComp statement and a source code context, the model must generate an explanatory docstring [10].
- **Statement Reconstruction (NL → Formal):** Given a docstring and source code context, the model must produce the corresponding Rocq formal statement (definition or lemma). This is a form of *auto-formalization* on a small scale [11].

Acknowledgments

This work has received funding from the Inria "Défi LLM4Code". We also thank Guillaume Baudart and Marc Lelarge for their valuable input.

References

[1] Mathematical Components Team. Mathematical Components library. https://github.com/math-comp/math-comp, 2007.

[2] Peiyang Song, Kaiyu Yang, Anima Anandkumar, "Towards Large Language Models as Copilots for Theorem Proving in Lean" *arXiv preprint* 2404.12534, (2025).

[3] Jialin Lu, Kye Emond, Weiran Sun, Wuyang Chen, "Lean Finder: Semantic Search for Mathlib That Understands User Intents." *AI4Math Workshop @ ICML 2025.*

[4] Andreas Florath, "Enhancing Formal Theorem Proving: A Comprehensive Dataset for Training AI Models on Coq Code." *arXiv preprint* 2403.12627, (2024).

[5] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-Coder technical report. *arXiv preprint* 2409.12186, (2024).

[6] Long Ouyang , Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, Ryan Lowe "Training language models to follow instructions with human feedback." *NeurIPS 2022.*

[7] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, Sean Welleck. "Llemma: An Open Language Model For Mathematics." *ICLR 2024.*

[8] Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, Tong Zhang, "TheoremLlama: Transforming General-Purpose LLMs into Lean4 Experts." *arXiv preprint* 2407.03203, (2024).

[9] Andrei Kozyrev, Gleb Solovev, Nikita Khramov, Anton Podkopaev, "CoqPilot: a plugin for LLM-based generation of proofs." *AITP 2024* (poster abstract).

[10] Sayak Chakrabarty, Souradip Pal, "ReadmeReady: Free and Customizable Code Documentation with LLMs - A Fine-Tuning Approach" *Journal of Open Source Software* (2025).

[11] Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, Zhicheng Yang, Jing Tang, Zhijiang Guo, "Process-Driven Autoformalization in Lean 4.". *arXiv preprint* 2406.01940, (2024).

[12] Kyle Thompson, Nuno Saavedra, Pedro Carrott, Kevin Fisher, Alex Sanchez-Stern, Yuriy Brun, João F. Ferreira, Sorin Lerner, Emily First, "Rango: Adaptive Retrieval-Augmented Proving for Automated Software Verification." *ICSE 2025.*

[13] Label Studio, "HumanSignal/label-studio", GitHub, 2025.

[14] William Macke, Michael Doyle, "Testing the Effect of Code Documentation on Large Language Model Code Understanding" *Findings-NAACL*, (2024).

[15] Shubhang Shekhar Dvivedi, Vyshnav Vijay, Sai Leela Rahul Pujari, Shoumik Lodh, Dhruv Kumar, "A Comparative Analysis of Large Language Models for Code Documentation Generation," *First ACM International Conference on AIware*, (2024).

[16] Nikita Khramov, Andrei Kozyrev, Gleb Solovev, Anton Podkopaev, "RocqStar: Leveraging Similarity-driven Retrieval and Agentic Systems for Rocq generation." *arXiv preprint* 2505.22846, (2025).