Capstone Project Phase A

Interactive Mathematical Proof Verification System

24-2-R-15

Authors: Daniel Armaganian, Tzahi Bakal

Supervisors: Dr. Dan Lemberg, Mrs. Elena Kramer

Table of Contents

Abstract. Recent advancements in artificial intelligence, particularly large language models (LLMs) such as ChatGPT, have demonstrated their capacity to generate mathematical proofs with remarkable skill. Nevertheless, ChatGPT is not yet capable of verifying the accuracy of its proofs, which presents a major obstacle to their integration into formal mathematical studies. To solve this problem, this project aims to create an interactive proof verification system. Our suggested method builds a framework in which the correctness of proofs produced by ChatGPT may be verified using a formal proof language, Agda. The system features an API that lets ChatGPT, and the formal proof environment interact, ensuring that proofs are both generated and validated. This integration aims to enhance the reliability of AI-assisted mathematical proofs, and by that accelerating mathematical research and improving the overall trustworthiness of AI-generated results.

Keywords: Artificial Intelligence · Large Language Models · ChatGPT · Formal Verification · Formal Proof Languages · Agda.

1 Introduction

In current years, the sphere of artificial intelligence has witnessed superb advancements, specifically within the domain of LLMs. These models, exemplified by using structures like ChatGPT, have validated an outstanding capability to interact in complicated reasoning obligations, together with the method of mathematical proofs [1], [5]. However, a difficult persists: at the same time that ChatGPT can generate proofs, it lacks the capability to verify the correctness of the proofs it generates [2], [5]. This gap between proof generation and verification presents a challenge in fields where accuracy is crucial. As LLMs are increasingly adopted in research, it becomes critical to develop methods that ensure the trustworthiness of their outputs.

This project aims to address this gap by developing an interactive proof verification system that bridges the gap between the creative potential of the LLM and the difficult foundation of formal mathematics. Using Per Martin-Löf and Sambin's [3] concept of intuitive types and Vladimir Voevodsky's [4] concept of grounded Homotopy types, we recommend a framework that can communicate with LLMs to solve and verify mathematical proofs

It is impossible to exaggerate how important this task is. As LLMs are increasingly included into mathematical and scientific research, it is crucial to guarantee or at least significantly improve the accuracy of their results. By integrating between human intuition, artificial intelligence, and formal verification, a strong verification system could potentially speed up mathematical research while simultaneously improving the trustworthiness of AI-assisted mathematical discoveries.

Our method is based on Voevodsky's Homotopy type theory [4], which provides tools for managing higher-dimensional structures and equalities, and Martin-Löf and Sambin's intuitionistic type theory [3], which provides a framework for constructive reasoning. We build a framework in which proofs produced by ChatGPT may be validated by putting these theories into practice in formal proof languages like Agda.

The remainder of this paper proceeds as follows. First, we will explore the technical details of our proposed system, starting with an overview of the relevant type theories. We will then describe the architecture of our verification system, including the design of the API for communication with ChatGPT. Finally, we will present results from our research and discuss the challenges and future directions for this research.

2 Literature Review

2.1 ChatGPT as an Expert Tool

Azaria, Azoulay, and Reches (2024) discuss the potential of ChatGPT as a powerful assistant for experts, highlighting its capability to generate complex solutions across various subjects. However, they emphasize that its outputs may require validation to ensure accuracy, pointing to the need of integrating some kind of a verification system when deploying LLMs in fields like mathematics [1]. This observation shows the importance of our system, which seeks to develop a mechanism that ensures the reliability of AI-generated proofs.

2.2 LLMs and Software Verification

Janßen, Richter, and Wehrheim (2024) explore the application of LLMs like ChatGPT in the domain of software verification. They find that while ChatGPT can produce valid and useful invariants that assist formal verifiers, the tool alone is insufficient for full verification of logical correctness. The study highlights the necessity of integrating these LLM-generated outputs with robust formal verification tools and human oversight [2]. This finding is relevant to our research, as it underscores the limitations of relying solely on LLMs for verifying the correctness of mathematical proofs.

2.3 Foundational Theories in Proof Verification

Our approach draws on the foundational theories of Intuitionistic Type Theory (ITT) and Homotopy Type Theory (HoTT), which provide the theoretical foundations for constructing and verifying proofs.

Martin-Löf and Sambin’s (1984) ITT offers a framework for constructive reasoning,

where mathematical proofs are treated as algorithms that must be explicitly constructed [3]. This aligns with the need for a verification system that can confirm the correctness of AI-generated proofs through constructive methods.

Voevodsky's HoTT introduces tools for reasoning about higher-dimensional structures and equalities, offering a robust framework for formalizing complex mathematical objects [4]. By integrating these theories into our system, we aim to create a verification framework that not only validates proofs but also accommodates the intricate structures that AI-generated proofs may involve.

2.4 Agda in Formal Verification

The implementation of these type theories in practical proof assistants has led to the development of formal proof languages such as Agda. Stump (2016) emphasizes the role of Agda, a dependently typed programming language, in formal verification. Agda's support for constructive proofs and its ability to represent complex mathematical structures with precision [9], make it an ideal choice for our proposed verification system. By employing Agda, our system will be able to formalize and verify the correctness of proofs generated by ChatGPT, ensuring their reliability and consistency.

```
{- Definition of natural numbers (ℕ) with zero and successor constructors -}
data ℕ : Set where
  zero : ℕ
  succ : ℕ → ℕ

{- Definition of equality (≡) for natural numbers with reflexivity constructor (n≡n) -}
data _≡_ : ℕ → ℕ → Set where
  n≡n : {n : ℕ} → n ≡ n

{- Definition of addition for natural numbers -}
_+_ : ℕ → ℕ → ℕ
n + zero = n
n + (succ m) = succ (n + m)

{- Proves that if m == n, then m + 1 == n + 1 -}
m≡n→sm≡sn : {m n : ℕ} → m ≡ n → (succ m) ≡ (succ n)
m≡n→sm≡sn n≡n = n≡n

{- Transitive relation. Helps for chaining equations into the final result -}
k≡m∧m≡n→k≡n : {k m n : ℕ} → k ≡ m → m ≡ n → k ≡ n
k≡m∧m≡n→k≡n n≡n n≡n = n≡n

{- Proof that zero + n ≡ n + zero for any natural number n -}
z+n≡n+z : (n : ℕ) → (zero + n) ≡ (n + zero)
z+n≡n+z zero = n≡n
z+n≡n+z (succ n) = m≡n→sm≡sn (z+n≡n+z n)

{- Proves that for any number m and n the equality succ m + n = succ (m +n) holds -}
sm+n≡s[m+n] : (m n : ℕ) → (succ m + n) ≡ succ (m + n)
sm+n≡s[m+n] m zero = n≡n
sm+n≡s[m+n] m (succ n) = m≡n→sm≡sn (sm+n≡s[m+n] m n)

{- Main proof: Commutativity of addition -> or any a and b, the equality a + b = b + a holds.-}
m+n≡n+m : (m n : ℕ) → (m + n) ≡ (n + m)
m+n≡n+m zero n = z+n≡n+z n
m+n≡n+m (succ m) n = k≡m∧m≡n→k≡n (sm+n≡s[m+n] m n) (m≡n→sm≡sn (m+n≡n+m m n))
```

Figure 1. Commutativity proof of addition in Agda. Created by Carbon https://carbon.now.sh/

2.5 LLMs in Mathematical Problem Solving

Plevris, Papazafeiropoulos, and Rios (2023) conducted a comparative study on the performance of various LLMs, including ChatGPT, in solving mathematical and logical problems. Their research highlights the variability in the logical clearness of solutions provided by these models, showing the need for verification systems [5]. While ChatGPT can generate mathematical questions and explanations across various difficulty levels, it struggles with consistently producing correct proofs, especially for complex problems. Similarly, Shakarian et al. [7] conducted an independent evaluation of ChatGPT's capabilities in solving mathematical word problems, highlighting significant challenges when faced with complex problems. These studies support our approach of integrating a formal verification framework with ChatGPT to ensure the correctness of its generated proofs.

2.6 Unifying Type Theory and Formal Verification

Pelayo and Warren (2014) explore Homotopy Type Theory and its univalent foundations, providing a unified approach to reasoning about mathematical structures. Their work highlights the potential of HoTT to serve as a robust foundation for formal verification, particularly in the context of complex mathematical proofs [4]. This theoretical foundation is critical to our project, as it informs the development of a verification system that leverages both intuitionistic and homotopy type theories to ensure the correctness of AI-generated proofs.

3 Background

Our work builds upon two fundamental theories in type theory: Intuitionistic Type Theory (ITT) and Homotopy Type Theory (HoTT). We begin by discussing ITT, followed by HoTT, the use of the Agda programming language, and finally, our system's API.

3.1 Intuitionistic Type Theory

Intuitionistic Type Theory, developed by Per Martin-Löf and Sambin, represents a significant advancement in the formalization of mathematical reasoning. The primary goal of ITT was to construct a formal system that could represent informal mathematical reasoning more accurately than previous foundational systems [3].

A key feature of ITT is the distinction between propositions and judgments. Propositions are statements that can be combined using logical operations and can be true or false, while judgments are assertions about the truth of propositions. This distinction forms the basis for the system's rules of inference, which are justified by explaining how conclusions can be drawn from premises known to be true.

ITT introduces four fundamental forms of judgment:

1. A is a set
2. A and B are equal sets
3. a is an element of set A
4. a and b are equal elements of set A

In this system, sets are defined by specifying rules for forming canonical elements and equal canonical elements. An element of a set is conceptualized as a method or program that yields a canonical element of that set when executed.

One of the distinguishing features of ITT is its intuitionistic approach to propositions. Rather than defining propositions in terms of truth values, ITT defines them in terms of what constitutes a proof of the proposition. This aligns with the intuitionistic interpretation of logical operations and allows for a more constructive approach to mathematics.

The theory pays particular attention to specific set operations, including the Cartesian product, disjoint union, and natural numbers. These operations and their associated rules are used to interpret various logical connectives and quantifiers. Notably, ITT provides a proof for the axiom of choice within its framework, demonstrating the system's expressive power.

To extend the system beyond finite types, ITT introduces the concept of universes. These allow for the construction of transfinite types, which are necessary for dealing with certain concepts in category theory that cannot be adequately captured with just finite types.

Overall, ITT aims to provide a rich formal system that serves a dual purpose: as a rigorous foundation for mathematics and as a practical programming language. This dual nature makes ITT particularly relevant to our work, as it bridges the gap between theoretical foundations and practical implementations in computer science.

3.2 Homotopy Type Theory

Homotopy Type Theory (HoTT) represents a combination of abstract homotopy theory and type theory. It emerged from the discovery of deep connections between Martin-Löf and Sambin's constructive type theory and abstract homotopy theory [4].

The key insight of HoTT is the interpretation of types as spaces (or homotopy types) and terms of a type as points in that space.

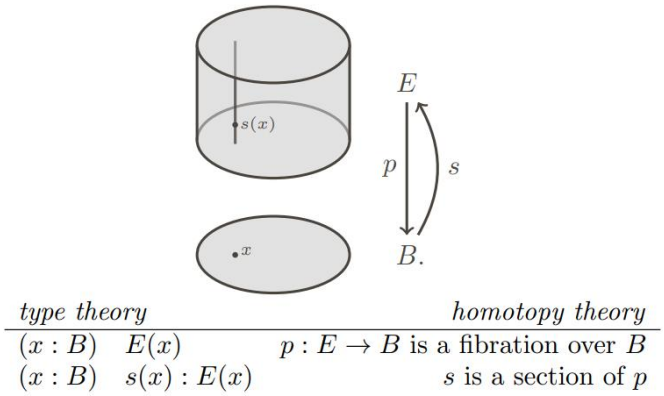

Figure 2. Illustration of how dependent types in type theory can be interpreted geometrically as fibrations in homotopy theory [4].

A central concept in HoTT is the Univalence Axiom, introduced by Vladimir Voevodsky (2014). This axiom states that identity between types is equivalent to equivalence of types. Formally: (A = B) ≃ (A ≃ B) where ≃ denotes equivalence. This principle allows for more flexible reasoning about equality.
"In other words, identity is equivalent to equivalence. In particular, one may say that 'equivalent types are identical'. [6]

HoTT also introduces higher inductive types, which allow for the direct definition of many important spaces and constructions from homotopy theory within the type theory itself.

The univalent perspective, proposes adopting HoTT as a foundation for mathematics, offering several key advantages. It provides a constructive foundation that aligns with homotopy theoretic concepts, enabling direct formal verification of proofs in varies proof assistants such as Agda. Furthermore, this approach yields fresh insights and techniques that benefit both type theory and homotopy theory. By linking the logic of type theory with the geometric intuitions of homotopy theory, HoTT creates a powerful framework that not only enhances our understanding of mathematical structures but also facilitates the development of machine-checkable proofs, potentially revolutionizing how we approach mathematical reasoning and verification.

3.3 Agda

In our project, we utilize Agda, a dependently typed programming language rooted in Martin-Löf and Sambin's ITT [9]. This theoretical foundation provides a framework for constructive reasoning, where mathematical proofs are approached as algorithms that need to be explicitly formulated and verified.

Agda is a dependently typed programming language primarily used for writing and verifying formal proofs in a functional programming setting. As Sitnikovski [8] explains, dependent types provide a powerful way to encode program proofs, offering a robust foundation for formal verification in Agda.

As an advanced tool for verified programming, Agda offers a unique approach where the correctness of programs is ensured through mathematical proofs written in the language itself. The proofs are checked by Agda's compiler, which verifies that the program adheres to its specifications across all possible inputs.

Agda is grounded in constructive logic, where types are viewed as logical propositions, and programs that inhabit these types serve as proofs of the propositions. This approach is based on the Curry-Howard correspondence, a fundamental concept in type theory that equates types with propositions and programs with proofs.

Agda's type system is highly expressive, supporting features like dependent types, where types can depend on values. This allows for precise specifications of program properties, such as the length of a vector being encoded in its type. Additionally, Agda supports both internal and external verification of functions, offering a framework for proving the correctness of complex algorithms and data structures.

The language's strong emphasis on pure functional programming, where functions behave like mathematical functions without side effects, aligns well with the goals of verified programming. Agda also enforces termination, ensuring that all programs eventually produce a result, which is crucial for maintaining logical consistency in proofs.

Agda's interactive development environment, allows users to write, type-check, and interactively develop proofs and programs. This environment supports Unicode, enabling the use of mathematical symbols directly in code, which enhances the readability and expressiveness of proofs.

Overall, Agda represents a powerful tool for researchers and developers interested in formal methods, providing a platform to explore and apply advanced concepts in type theory and functional programming.

3.3.1 More on Agda

Agda's power as a proof assistant and programming language stems from several key features:

Inductive Types: Agda allows the definition of inductive data types, which are fundamental for representing recursive structures.

```
data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
```

Figure 3. Agda code of inductive definition of the type N for the Peano natural numbers [9].

Pattern Matching and Recursion: Agda uses pattern matching and structural recursion for function definitions, allowing for clear and concise code.

```
_+_ : ℕ → ℕ → ℕ
zero  + n = n
suc m + n = suc (m + n)
```

Figure 4. Agda code of recursive definition of addition [9].

3.4 Our System's API and Expected Achievements

Our system features an API designed to bridge the gap between natural language mathematical statements and formal verification processes. The API serves as an interface that allows interaction between AI-driven proof generation and formal verification methods, aiming to streamline the iterative process of proof generation and validation. It is structured to facilitate seamless communication between ChatGPT and the Agda-based formal verification system, ensuring that proofs are generated correctly.

The API accepts natural language input, translating it into a semi-formal proof structure that is subsequently validated by Agda. The system is designed for iterative refinement, where failed proofs are analyzed, and feedback is provided to the proof generation module, enabling continuous improvement until convergence is achieved. This architecture allows for efficient proof discovery and enhances the reliability of AI-generated mathematical proofs.

4 Expected Achievements

Our primary objective is to achieve convergence during the iterative verification process of proofs by Agda. This convergence is crucial as it represents the system's ability to refine AI-generated proofs to a point where they were verified by Agda and represent a correct proof. The project aims to create a reliable integration between ChatGPT, and formal verification method, Agda. By successfully merging these technologies, the system will allow AI-generated proofs to be verified for correctness, ensuring that the API facilitates effective communication between these components. This integration will result in a proof verification cycle, where AI-generated proofs can be checked and validated.

Another key goal is to develop an iterative process where initial proof attempts are systematically refined based on feedback from the verification module. The system is designed to achieve a high rate of convergence, meaning it should reliably verify the proof after a series of iterations. This iterative refinement process is essential for enhancing the accuracy and reliability of the proofs, as it allows for continuous improvement until the proofs meet formal verification standards.

Moreover, the project aims to enhance the efficiency of proof generation by automating significant portions of the verification process. By integrating AI with formal methods, the system can reduce the time and effort required for mathematical proof verification, demonstrating the potential to accelerate mathematical research.

Finally, the project seeks to improve the trustworthiness of AI-generated proofs. By establishing a verification process, the system will increase the reliability of proofs generated by ChatGPT. This increased trustworthiness is crucial for the broader adoption of AI in mathematical research. The framework developed in this project could also be extended to other LLMs, formal proof languages and verification systems, potentially broadening its applicability in the field of mathematical logic. These expected achievements represent a significant advancement in the integration of AI with formal verification methods, offering a powerful tool for mathematicians and researchers.

5 Research/Engineering Process

5.1 Problem Statement

The primary challenge addressed in this research is the inability of ChatGPT to verify the mathematical proofs they generate. While these models are capable of producing proofs with a high degree of complexity, there remains no mechanism to validate their correctness independently. Our research aims to develop an interactive proof verification system that leverages ITT and HoTT theories, implemented through formal proof language, Agda, to close this verification gap.

5.2 Research Objectives

Our research has the following objectives:

1. Develop an API that enables real-time interaction between ChatGPT and formal verification system, Agda.
2. Design and implement a framework that iteratively refines AI-generated proofs to achieve convergence.
3. Test the system’s efficiency and scalability in generating verified proofs across a variety of mathematical problems.

5.3 System Architecture

The proof verification system consists of three main components:

1. User's mathematical prompt in natural language.
2. ChatGPT for generating semi-formal proofs.
3. Agda for verifying those proofs.

4. An API that facilitates communication between the two systems. The API translates ChatGPT's natural language output into Agda-compatible code, receives feedback from Agda, and feeds it back to ChatGPT for iterative refinement.

5.4 Proof Refinement Algorithm

The main part of our solution, is to create an algorithm that integrates ChatGPT and Agda within our system. Our approach is to enable ChatGPT to refine the proof iteratively based on feedback provided by Agda.

```
function refine_proof(proof)
    start_time = current_time()
    feedback = agda_verify(proof)

    while feedback is not valid and (current_time() - start_time) < max_duration do
        proof = chatgpt_refine(feedback)
        feedback = agda_verify(proof)
    end while

    if feedback is valid then
        return proof
    else
        return None
    end if
end function
```

Figure 5. Pseudo code the algorithm. Created by Carbon https://carbon.now.sh/

The algorithm begins by receiving a mathematical proof request from the user. ChatGPT generates an initial proof, which is then verified using Agda. If the proof is found to be invalid, feedback is sent back to ChatGPT to refine and improve the proof. This process of refinement continues iteratively, with the proof being updated and re-verified, until it either becomes valid or the time limit is reached. Finally, the results, whether successful or not, are displayed to the user.

5.5 Challenges

- Non-Convergence: Some proofs may not converge within the allowed iteration limit, requiring manual intervention or additional refinement strategies.
- Efficiency: Optimizing the API to handle large and complex proofs efficiently will be a key technical challenge.
- Integration: Ensuring smooth communication between ChatGPT and Agda, while maintaining robustness across different types of proofs.
- Agda code: Generating Agda code with ChatGPT can be difficult as it requires from ChatGPT to constantly meet all the complex syntactic rules with Agda.

5.6 Requirements

To ensure the success of our proof verification system, it is essential to define both functional and non-functional requirements that align with the project's objectives. These requirements provide a comprehensive framework for the system's development and evaluation.

5.6.1 Functional Requirements

1. The system must be able to interact with ChatGPT's API.
2. The system must generate semi-formal mathematical proofs using ChatGPT.
3. The system must convert semi-formal proofs to formal Agda code.
4. The system must support iterative refinement of proofs.
5. The system must provide feedback on failed proofs.
6. The system must facilitate interaction between AI and verification.
7. The system must derive proofs from natural language input.
8. The system must store proofs and verification statuses.
9. The system must provide an option to manually interrupt the iterative refinement.
10. The system must limit the time of the iterative process.
11. The system must allow exporting verified proofs.

5.6.2 Non-Functional Requirements:

1. Should perform proof generation and verification promptly.
2. Should efficiently handle complex proofs.
3. Ensure high reliability in proof verification.
4. Include error detection and recovery mechanisms.
5. Should be intuitive and easy to use.
6. Must have mathematical statements, generated proof and verification result.
7. Should facilitate easy maintenance and updates.
8. Limit proof refinement to a maximum of user set time
9. Should support a dark mode for user interface accessibility.

5.7 Product Diagrams and GUI

In this section, we describe our work process and a first draft of the proposed GUI for the system. As the first step we plan to develop a desktop application in order to realize our research. Figure 6 shows a use case diagram that summarize details of the system and the user within it, and Figure 7 shows the flow of the system.

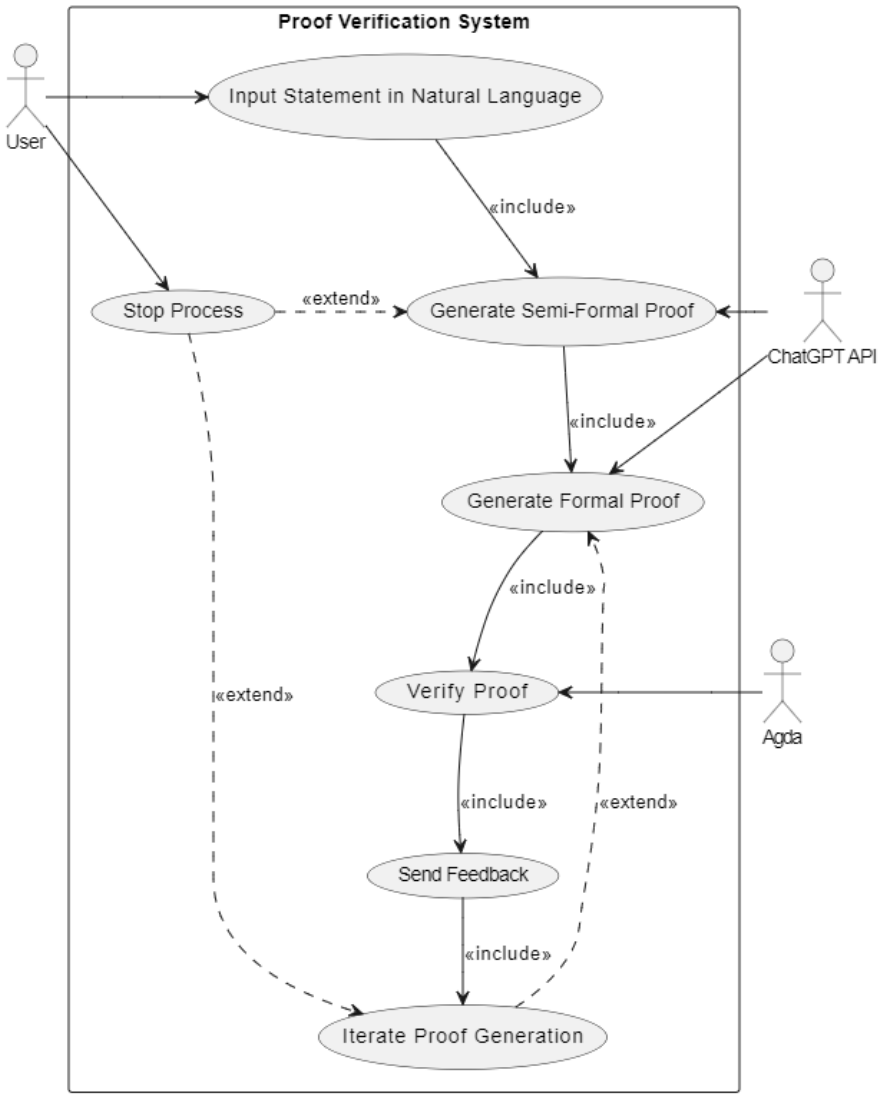

Figure 6. Use case diagram of the system. Created by http://www.draw.io

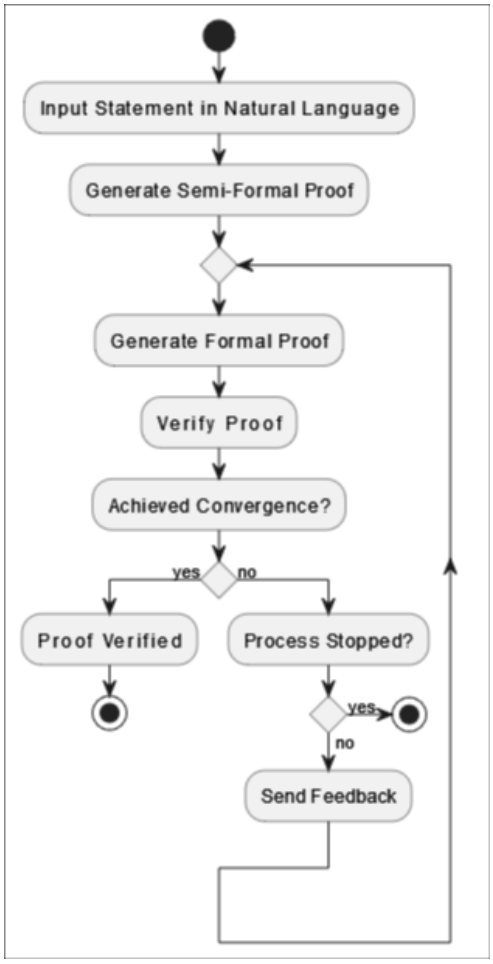

Figure 7. Activity diagram of the system. Created by http://www.draw.io

Figure 8 demonstrates the main GUI of the system. Within it, we can see that the screen is divided into three parts:

1. Area for user to enter his statement.
2. Area for the proof generated by ChatGPT.
3. Area for the status of verification.

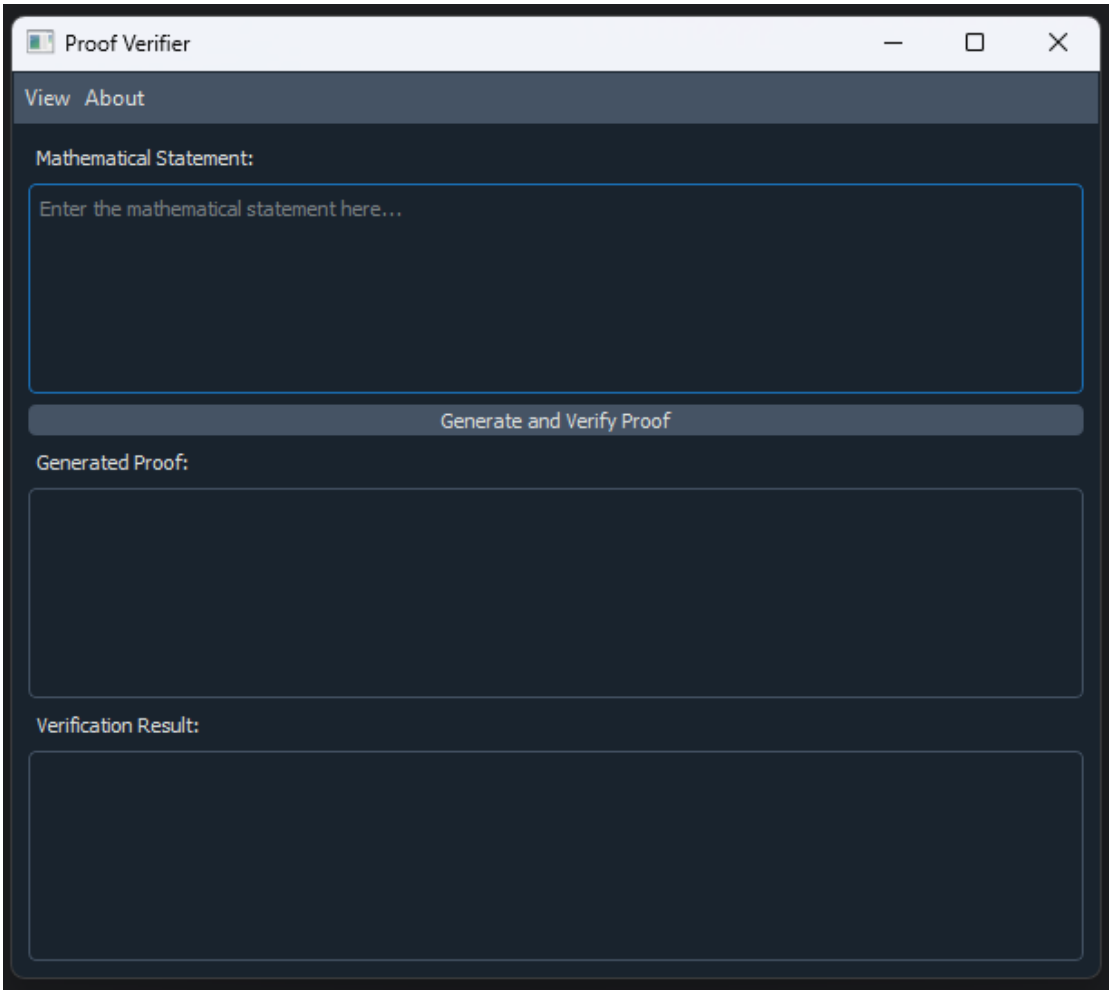

Figure 8. GUI of the system.

6 Evaluation/Verification Plan

In order to evaluate and verify the functionality of our proof verification system, we have established several key use cases and corresponding test cases to ensure the system performs as expected under various conditions. The system will be tested for accuracy, efficiency, and scalability using both automated and manual verification methods.

6.1 Test Scenarios

The test cases outlined in Table 1 are designed to evaluate the core functionalities of the system, ensuring that it accurately processes natural language inputs, generates correct proofs, and handles various edge cases. Each test case simulates real-world usage scenarios to verify the reliability of the system.

Test Case	Test Case Description	Expected Outcome
1	Input a valid natural language mathematical statement into ChatGPT	ChatGPT generates a semi-formal proof, which is translated into Agda code for verification
2	Proof verification fails due to incorrect input	The system provides feedback, and ChatGPT refines the proof
3	Input a large and complex proof statement	The system efficiently handles the input and completes verification within an acceptable timeframe
4	Attempt to verify an invalid or incomplete proof	The system detects the errors, provides feedback, and does not falsely confirm validity
5	User interrupts the iterative refinement process	The system safely halts the refinement process, saving the current state of the proof
6	The maximum duration of the iterative process is reached	The system safely halts the refinement process, saving the current state of the proof

Table 1: Suggested test cases for our program validation.

6.2 Automated Testing

For automated testing, we will implement a set of unit and integration tests to verify individual components of the system:

- Unit Tests: Testing the interaction between the API, ChatGPT, and Agda to ensure correct responses for a variety of mathematical inputs.
- Integration Tests: Verifying the seamless integration of the proof generation, refinement, and verification processes.
- Performance Tests: Measuring the system's response time and memory usage for complex proofs to ensure scalability.

6.3 User Acceptance Testing (UAT)

In collaboration with mathematicians and proof verification experts, we will conduct UAT to ensure the system meets user expectations. The feedback from experts will focus on:

1. Accuracy of Proof Verification: Assessing the correctness of the verified proofs.
2. Ease of Use: Evaluating the intuitiveness of the user interface and refinement process.
3. Error Handling: Ensuring that the system provides clear and actionable feedback for erroneous proofs.

6.4 Evaluating Success and Project Outcomes

The success of our proof verification system will be evaluated based on several key metrics, including accuracy, reliability, scalability, efficiency, and user experience. A critical measure of success is the system's ability to accurately generate and verify mathematical proofs, ensuring that ChatGPT's outputs are translated into valid Agda code and that any errors are handled through iterative refinement. The system should be able to process increasingly complex proofs without a significant drop in performance, highlighting its scalability. Efficiency is another important factor, with the system expected to perform proof generation, refinement, and verification within acceptable time frames, while also maintaining low latency during interactions. Furthermore, user experience will be assessed through an intuitive and user-friendly interface, enabling seamless interaction between users and the system. This includes the ability to input natural language statements, track proof generation progress, and view verification results clearly. Moreover, the system's ability to improve the trustworthiness of AI-generated proofs will be key to its broader adoption, ensuring that ChatGPT's mathematical solutions can be rigorously verified for correctness. Passing these evaluations will confirm that the system meets its goals of enhancing the reliability, efficiency, and scalability of AI-driven proof verification.

7 Bibliography

[1] Azaria, A., Azoulay, R., & Reches, S. (2024). ChatGPT is a remarkable tool—for experts. *Data Intelligence*, *6*(1), 240-296. https://doi.org/10.1162/dint_a_00235

[2] Janßen, C., Richter, C., & Wehrheim, H. (2024, April). Can ChatGPT support software verification? In *International Conference on Fundamental Approaches to Software Engineering* (pp. 266-279). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-57259-3_13

[3] Martin-Löf, P., & Sambin, G. (1984). *Intuitionistic type theory* (Vol. 9, p. 136). Naples: Bibliopolis. https://archive-pml.github.io/martin-lof/pdfs/Bibliopolis-Book-retypeset-1984.pdf

[4] Pelayo, Á., & Warren, M. (2014). Homotopy type theory and Voevodsky's univalent foundations. *Bulletin of the American Mathematical Society*, *51*(4), 597-648. https://doi.org/10.1090/S0273-0979-2014-01456-9

[5] Plevris, V., Papazafeiropoulos, G., & Rios, A. J. (2023). Chatbots put to the test in math and logic problems: A preliminary comparison and assessment of ChatGPT-3.5, ChatGPT-4, and Google Bard. *arXiv preprint arXiv:2305.18618*. https://doi.org/10.48550/arXiv.2305.18618

[6] Program, T. U. F. (2013). Homotopy type theory: Univalent foundations of mathematics. arXiv preprint arXiv:1308.0729. https://doi.org/10.48550/arXiv.1308.0729

[7] Shakarian, P., Koyyalamudi, A., Ngu, N., & Mareedu, L. (2023). An independent evaluation of ChatGPT on mathematical word problems (MWP). https://doi.org/10.48550/arXiv.2302.13814

[8] Sitnikovski, B. (2023). *Introduction to dependent types with Idris: Encoding program proofs in types*. Apress. https://doi.org/10.1007/978-1-4842-9259-4

[9] Stump, A. (2016). *Verified functional programming in Agda*. Morgan & Claypool. https://doi.org/10.1145/2841316