LEAN 4 in the AI Software Domain

Langze Ye, Xuan Zhang, Jiaheng Xiong
Politecnico di Milano, Italy

***Abstract*—The intersection of artificial intelligence (AI) and programming languages has yielded significant advancements in software development. As AI continues its evolution, the demand for programming languages capable of robust reasoning, formal verification, and efficient modeling grows. LEAN 4, an advanced formal theorem proving language, emerges as a language of profound interest in this context. This paper extensively explores the applications of LEAN 4 within AI software, focusing on its potential to enhance logic-based reasoning, machine learning, and natural language processing. By synergizing formal verification and AI, LEAN 4 paves the way for more dependable, transparent, and secure AI systems.**

***Index Terms*—LEAN 4, artificial intelligence (AI), formal theorem proving language**

I. Introduction

LEAN 4 [1], [2] is an advanced formal theorem proving language that provides developers with powerful tools for formal reasoning and modeling. It is based on Dependent Type Theory, a powerful type system that allows developers to express rich logical relationships in code. Unlike traditional programming languages, LEAN 4's type system focuses not just on the structure and data types of the code, but also on the logical nature of the code.

An important feature of LEAN 4 is its support for higher-order logic. This means that developers can express more complex logical relationships in LEAN 4, including first-order logic, predicate logic, and more advanced forms of logic. This gives LEAN 4 an advantage when dealing with complex mathematical theorems and logic problems.

Formal proofs are another core feature of LEAN 4. It allows developers to express and verify mathematical theorems, algorithmic properties, and software behavior in a formal manner. By using formal proofs, developers can accurately prove the correctness of code, thus greatly reducing the likelihood of errors and defects. Different from other programming language, LEAN 4 supports interactive theorem proving as Fig. 1, which means that developers can build proofs step-by-step, interacting with the system by the 'Lean Infoview' window to build complex arguments over time. This interactive approach helps developers gain a deeper understanding of the problem and ensures the accuracy of the proof [3], [4].

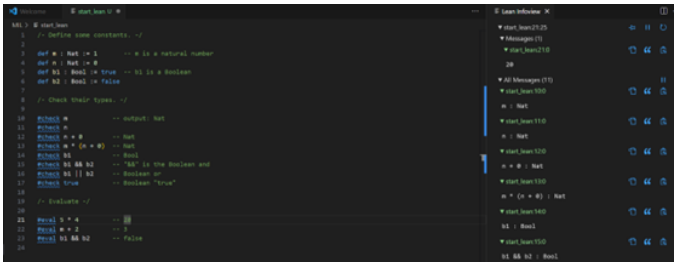

Fig. 1. Interactivity in LEAN4

In the field of AI software, these features of LEAN 4 provide powerful tools for formal reasoning and modeling. It can be used to express the nature of complex algorithms and models, to verify logical relationships in the reasoning process, and to ensure the correctness and security of AI systems [5].LEAN 4's capabilities in formal reasoning and proofs make it an indispensable tool in AI software development.

II. LEAN 4 in Logical Reasoning - Mathematical Theorem Proving

A. the law of exchange for the addition of natural numbers

In order to demonstrate the formal theorem proving power of LEAN 4, we can consider a simple mathematical theorem, such as the law of exchange for the addition of natural numbers as Fig. 2. In LEAN 4, we first need to represent the problem formally. We can define a function that represents the addition of natural numbers and state this exchange law theorem. Here, we can find the first advantage of LEAN4: We formalize the problem almost directly in mathematical language. So that it's very clear to understand the question even though you can't understand anything about the code.

```
theorem add_comm : ∀ (a b : N), a + b = b + a :=
begin

end
```

Fig. 2. Represent the law of exchange for the addition of natural numbers in lean4

Then we need to fill in the proof steps between 'begin' and 'end'. We need to represent the math method in code. In this example, we need to use the addition rule and mathematical induction. In LEAN 4, proofs are constructed by step-by-step logical reasoning and application of known theorems. Each step of the proof requires sound reasoning based on logical rules. The proof of the exchange law theorem is based on the axioms of Peano. In brief, we should prove that it works for b=0 firstly. Then we assume it holds for b=k, and we need to prove that it's also true for b=succ(k). (With axioms of Peano, $succ(k) = k + 1$).

The following codes are as Fig. 3:

In this case, at first, natural numbers a and b are introduced and they are the most fundamental assumptions needed for

```
theorem add_comm : ∀ (a b : ℕ), a + b = b + a :=
begin
  -- Introducing natural numbers a and b
  intros a b,
  -- Analyze b by induction
  induction b with b' ih,
  -- Base case: b = 0
  { rw [nat.add_zero, nat.zero_add] },
  -- Inductive case: b = b' + 1
  { rw [nat.add_succ, ih, nat.succ_add] }
end
```

Fig. 3. Proof steps for the exchange law theorem

the proof process. Then we use induction to analyze the variable b. Section I is used to make a mathematical induction that will analyze various scenarios of the variable b and will apply the appropriate reasoning steps in each case. $rw[nat.add_zero, nat.zero_add]$ means when b is 0, we use the rw (rewrite) strategy to apply a known mathematical equation by replacing the terms on both sides of the equation. Here, we use the properties of addition of natural numbers, where $nat.add_zero$ means that the result of adding 0 to a natural number, and $nat.zero_add$ means that the result of adding 0 to any natural number is still that natural number. Finally, when b is $b' + 1$. We use the rewrite strategy again, applying the properties of addition of natural numbers and the induction hypothesis (ih), respectively. here, $nat.add_succ$ denotes the definition of the successor of addition of natural numbers, and $nat.succ_add$ denotes the result of adding the successor number to the other.

Then, we are done with proof. With this example, we can find that each step must be logically rigorous and requires a proof based on a theorem already in the library. Normally, you need to know the math to prove this theorem in order to prove it in lean. However, it would be very cumbersome if every theorem required such a complex proof. With the continuous updating of LEAN language versions and its libraries, many common math problems no longer require repeated proofs. Now that LEAN4 supports the Mathlib library, there are many mathematical proofs that LEAN will automatically do for us.

Now we will try again to prove the law of exchange of addition using library Mathlib. It's a library which contains most of mathematical language and mathematical theorems. (In Lean4 we can use 'by' instead of 'begin' and 'end'.)

B. Infinity of Primes

We can also try to prove the infinity of primes using lean4. We need to prove that for every natural number n, there is a prime number greater than n. Our proof procedure can be based on Euclid's theorem. We can let p be any prime factor of $n!+1$. If p is less than n, it divides $n!$. Since it also divides $n!+1$, it divides 1, a contradiction. Hence p is greater than n. In LEAN 4, we need to formalize the problem first, after which we write the proof process and logic for each step. [6]

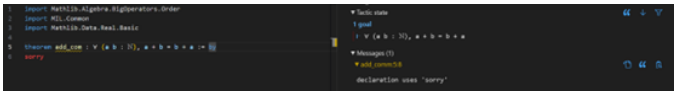

Fig. 4. Prove exchange of addition by Mathlib(step 1)

Suppose we now have no idea how to prove the law of exchange of addition by mathematical induction so now there's only one red 'sorry' in the proof process. We can just use the code 'library_search' to replace the 'sorry' to find something useful in the library as Fig. 4. After typing 'library_search', we

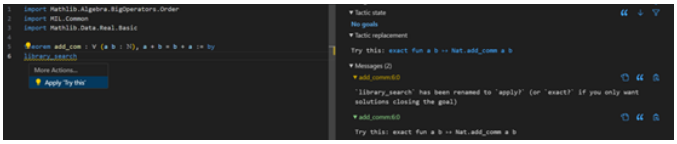

Fig. 5. Prove exchange of addition by Mathlib(step 2)

can find that it gives us a hint: Apply try this. In the right Lean infoview, we can find the function which it suggests us use. So that what we need to do is just click "Apply" as Fig. 5. Now,

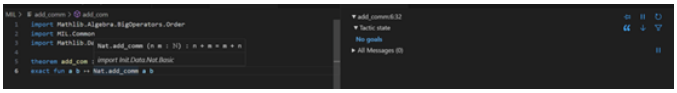

Fig. 6. Prove exchange of addition by Mathlib(step 3)

the proof is done. If we can't read the code, we just mouse over the function to consult the explanation of the function. Most simple math theorems can be proved automatically by 'library_search' with Mathlib as Fig 6.

We can try a slightly more complicated mathematical proof. In fact, many complex mathematical theorems can be broken down into smaller steps using the LEAN4 Theorem Prover. For example, we can try to prove the infinity of prime numbers. We need to write the entire framework of the proof based on Euclid's theorem. Due to the lack of theorem support for the proof step, in many places we can only temporarily indicate with 'sorry' as Fig 7.

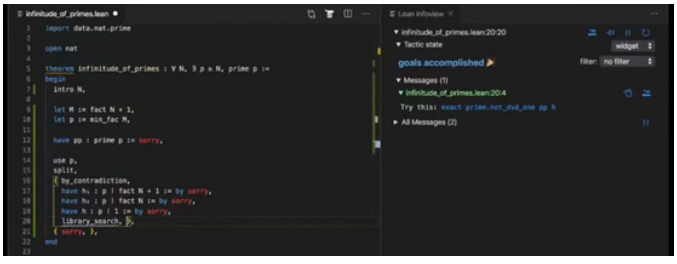

Fig. 7. Prove infinity of primes in LEAN 4

Then, we can try to complete it with the function 'library_search'. Once you've completed a theorem, you can also refer to it directly when needed. Of course, this automatic proof method has its limitations. Firstly, it can only look up

relevant theorems already in the library. Secondly, we need enough math knowledge to support us in breaking down the complex problem into several smaller problems that can be proved in almost one step. Nevertheless, sometimes even if lean4 does not give the correct theorem directly, we can get inspiration from the advice it gives us. This interactivity is useful for math theorem proving.

In addition to this, there are other helpful functions in mathlib. For example, a simple function 'ring' can directly verify that the expansion of both sides of the equation is consistent as Fig. 8.

```
example : c * b * a = b * (a * c) := by
  ring

example : (a + b) * (a + b) = a * a + 2 * (a * b) + b * b := by
  ring

example : (a + b) * (a - b) = a ^ 2 - b ^ 2 := by
  ring
```

Fig. 8. Function 'ring' in Mathlib

Many similarly useful functions also greatly simplify the mathematical proof process. As the library continues to be refined and updated in the future, more and more basic math theorems will be added to the library. Theoretically, LEAN4 can be used to assist in proving all existing mathematical theorems. There will also be more and more mathematical theorems that will be proved automatically in the future. The main application of LEAN4 at the moment is also used as a machine assistant prover in the field of mathematics.

III. Future Steps

LEAN4 also has powerful definitional capabilities. For example, we can use the function 'inductive' to define a list of weekdays. Then we can simply create a function to export 'next day' or 'previous day' as Fig. 9. Or we can also convert

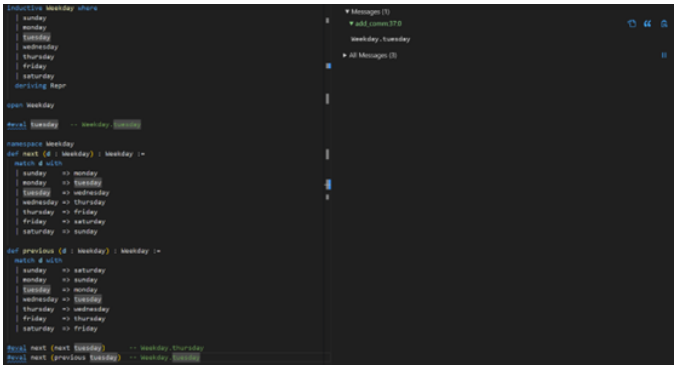

Fig. 9. Inductive 'weekday' in LEAN4

text to numbers as needed. We can use the 'match' expression to define a function from 'Weekday' to the natural numbers as Fig. 10.

At the same time, because we define new functions, we can try to prove the logic between words like a math problem as

```
def numberOfDay (d : Weekday) : Nat :=
  match d with
  | sunday    => 1
  | monday    => 2
  | tuesday   => 3
  | wednesday => 4
  | thursday  => 5
  | friday    => 6
  | saturday  => 7

#eval numberOfDay Weekday.sunday   -- 1
#eval numberOfDay Weekday.monday   -- 2
#eval numberOfDay Weekday.tuesday  -- 3
```

Fig. 10. Convert text to numbers

Fig. 10. Therefore, based on these functions and features of

```
example : next (previous tuesday) = tuesday :=
  sorry
```

Fig. 11. Prove logic between words

LEAN4, we can see the potential of LEAN4 in other areas of AI.

A. Natural Language Processing

Theoretically, we could create a library on natural language as we did with mathlib. For example, at its simplest, we can define some words as Fig. 12. And Lean4 can handle natural

```
inductive word : Type
| hello : word
| world : word
| lean : word

-- Defining Syntactic Structures
inductive sentence : Type
| simple : word → sentence
| compound : word → sentence → sentence

-- Defining Semantic Relationships
def semantics : word → string
| word.hello =>"Hello"
| word.world => "World"
| word.lean => "Lean"
```

Fig. 12. Defining Natural Language with LEAN4

language like a math problem! We can define functions as needed, such as defining a function that can determine whether sentences are consistent in meaning or not. Of course, this all needs to be supported by the full library. But it's enough to see the potential of LEAN4 in natural language processing.

B. Designing and optimizing machine learning algorithms

LEAN4 has strong formalization capabilities lean4 has a powerful formalization capability that, in theory, can formalize any machine learning algorithm. For example, we can try to define the VGG convolutional neural network. Fig. 13 above

```
def conv311 {n m} (k : Nat) (w : NDVector [3,1,1,k]) (x : NDVector [3,n,m]) : NDVector [n,m,k] := sorry
def conv33  {n m} (k : Nat) (l : Nat) (w : NDVector [3,3,k*l]) (x : NDVector [n,m,k]) : NDVector [n,m,k*l] := sorry
def fully_connected {dims} (n : Nat) (w : NDVector [dims.product, n]) (x : NDVector dims): NDVector [n] := sorry
```

Fig. 13. VGG convolutional neural network VGG convolutional neural network

is the definition of a convolutional layer function. It takes the following parameters:

- k: The number of convolutional kernels.
- w: The weight tensor of the convolutional layer.
- x: The input tensor.
- $[3, 1, 1, k]/[3, 3, k * l]$: Indicates the dimensions of each layer.

Each function returns a tensor of the corresponding dimension to represent the output of that layer. However, these definitions and formalizations require libraries to underpin them. 'Sorry' also indicates that this function has not been implemented yet. Currently, mathlib does not support definitions of vectors and dimensions. Therefore, if one is to fully implement algorithms for formalizing or designing machine learning on LEAN4, more knowledge in related fields is needed to build libraries.

C. Automated theorem proving

"Given the statement of a theorem, simply push a button, and the proof comes out." This is the goal that most LEAN language learners are pursuing today. If automated theorem proving (ATP) can come true, it will revolutionize the field of mathematics and machine learning. [7], [8]However, proof assistants (like LEAN) currently available are far inferior to human experts. To make the leap from semi-automated theorem proving (interactive theorem proving) to fully automated theorem proving, you need proof assistants and machine learning together. For example, the first version of "Lean Chat"

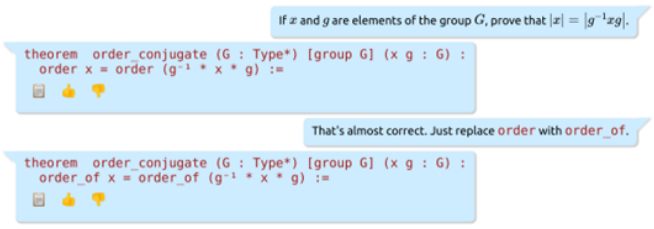

Fig. 14. Lean Chat

has already been designed [9]. By extracting mathematical theorems and proof processes from Mathlib as a dataset to train the model. We can chat with it by asking some mathematical theorems (in Latex) as Fig. 14, it can transfer the theorem to Lean code and try to prove it. However, for instant, ie can only give answers to uncomplicated theorems and sometimes there are still some mistakes. The datasets that Mathlib can provide are relatively limited. Achieving fully automated proofs in a wider range of areas is still a huge challenge.

Furthermore, a number of tools for extracting datasets from lean are also being developed and refined. For example, the 'LeanDojo' as Fig. 15, which is a a Python library for learning–based theorem provers in Lean. It can help us leverage lean and machine learning for automated theorem proving [10].

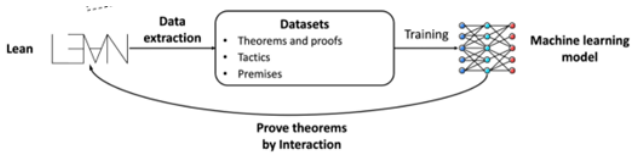

Fig. 15. The role of leandojo

IV. Conclusion and Analysis

With the above example, we can see that LEAN 4's impact on the AI domain lies in its formalization capabilities and interactivity. Firstly, LEAN 4 can be used as an interactive theorem prover to prove complex mathematical problems, while the interactivity it provides can deepen one's thinking about the problem. What's more, thanks to its formalization capabilities, LEAN 4 can be used to solve complex problems in other domains as well. For example, in the field of machine learning, LEAN 4 can be used to represent neural network structures, formalize machine learning algorithms, verify model properties, etc. Similarly, LEAN 4 can formalize natural language, analyze semantics, etc., which is helpful in the field of natural language processing (NLP).

In conclusion, LEAN 4 has a number of advantages for applications in the AI software space like its formalized capacity, rigor and credibility, and its interactive nature also helps reduce manual labor.

However, every coin has 2 sides, we also realized its limitations in the process of learning LEAN. For example, LEAN 4 has a relatively steep learning curve and takes some time and effort to master. At the same time, the configuration of the compilation environment and the import of libraries are complex. In practical applications, specialized training and resources may be required. What's more, although LEAN 4 supports formalization, facing complex problems will require more expertise and programming ability. LEAN's interactive proof system also has some limitations in solving complex problems.

All in all, in my opinion, the potential for LEAN 4 to be used in the AI software space remains huge. Although LEAN4 is still mostly used in math, thanks to the continuous

improvement of Mathlib. As technology evolves, LEAN 4 may further enhance the ability to automate formalisms, thereby reducing manual intervention and better accommodating the needs of large-scale systems. Specific LEAN 4 libraries may also emerge for the AI domain, including formal machine learning algorithms, model validation tools, etc., to simplify applications. If future versions of LEAN can be stabilized with constant updates, it will be one of the most promising tools of the future!

REFERENCES

[1] "Lean official website," https://leanprover.github.io/about/.

[2] "Lean 4 quick start," https://github.com/leanprover/lean4/blob/master/doc/quickstart.md.

[3] "Lean 4 grammar tutorial," https://www.bilibili.com/video/BV12m4y1x7CA/?spm_id_from=333.337.search-card.all.click&vd_source=e1e93e424f66b968a618cde73640d75a.

[4] "Numbergames," https://adam.math.hhu.de/#/g/djvelleman/STG4/world/Subset/level/3.

[5] "The lean 4 theorem prover and programming language," https://link.springer.com/chapter/10.1007/978-3-030-79876-5_37.

[6] "Prove the infinity of primes by lean 4," https://www.bilibili.com/video/BV1wU4y1475g/?spm_id_from=333.337.search-card.all.click&vd_source=e1e93e424f66b968a618cde73640d75a.

[7] "Machine learning for throrem proving," https://leanprover.zulipchat.com/#narrow/stream/219941-Machine-Learning-for-Theorem-Proving/topic/ML.20for.20Theorem.20Proving.20Tutorial.20-.20NeurIPS.202023.

[8] "Neurips tutorial on machine learning for theorem proving," https://machine-learning-for-theorem-proving.github.io/.

[9] "Lean prover zulipchat room," https://leanprover.zulipchat.com/.

[10] "Leandojo," https://github.com/lean-dojo/LeanDojo.