IsabeLLM: Automated Theorem Proving Applied to Formally Verifying Consensus

ELLIOT JONES, Imperial College London, United Kingdom

WILLIAM KNOTTENBELT, Imperial College London, United Kingdom

Advances in Artificial Intelligence (AI) have led AI for Theorem Proving to become a promising means of formally verifying computer systems. Whilst formal verification is traditionally reserved for safety-critical systems due to the required amount of expertise and effort, AI can help to automate a large amount of this workload and make it far more accessible. Blockchain-based systems are becoming increasingly popular and are frequently targeted by malicious actors, often resulting in huge financial losses, highlighting the need to better verify these systems and mitigate vulnerabilities. Arguably the most important component of these systems is the consensus protocol, which allows nodes to agree on decisions in a potentially adversarial environment. In this paper, we improve upon IsabeLLM, the automated theorem proving tool in Isabelle. Namely, we implement a Retrieval-Augmented Generation framework, Error tracing and counterexample generation for improved context supplied to the Large Language Model. Compatibility with the latest version of Isabelle and Sledgehammer is also implemented for improved efficiency. We compare the performance of the two versions of IsabeLLM in their ability to complete the verification of Bitcoin's Proof of Work consensus.

CCS Concepts: • **Security and privacy → Logic and verification**.

Additional Key Words and Phrases: Theorem Proving, Formal Verification, Blockchain, Consensus, Artificial Intelligence

ACM Reference Format:

Elliot Jones and William Knottenbelt. 2026. IsabeLLM: Automated Theorem Proving Applied to Formally Verifying Consensus. 1, 1 (June 2026), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

A blockchain enables peer-to-peer digital transactions without the need for a trusted intermediary. This is only possible because of its consensus protocol, which allows nodes within the system to agree on the state of the blockchain, even in the presence of adversaries. For this reason, it is paramount that consensus is designed and implemented correctly to prevent the system from reaching unwanted states that can be exploited by adversaries. The most famous example of this is Bitcoin's Proof of Work (PoW) consensus protocol and its susceptibility to a 51% attack, where adversaries control the majority of the compute power in the system and giving them the potential to double spend. Infamous examples of such attacks include Ethereum Classic [18], Bitcoin Gold [17], and Vertcoin [16], totalling losses of over $30 million.

Other key components of modern blockchain systems are bridging protocols for cross-chain data transfer and smart contracts for automated agreement execution. These components are also not without their exploits, with infamous

Authors' Contact Information: Elliot Jones, Imperial College London, London, United Kingdom, e.jones24@imperial.ac.uk; William Knottenbelt, Imperial College London, London, United Kingdom.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

examples such as the Poly Network [55], Wormhole Bridge [30], Binance Smart Chain [20], and Qubit Finance [14], totalling losses of over $1.5 Billion. These failures further underscore the need to ensure correctness across the domain.

Formal verification is the process of formalising a system and then mathematically proving its correctness. However, it is often underutilised in the software development process because of the large amount of effort and expertise it requires. Whilst organisations like the Ethereum Foundation actively fund formal verification work in the space, it continues to see the huge financial losses discussed previously. Furthermore, blockchain systems cannot rely on the traditional 'test and patch'model for their consensus protocols as patching would require a hard fork, such as the Ethereum/Ethereum Classic split [15]. Hard Forks are disruptive and controversial as they challenge blockchain's core principle of immutability. Furthermore, patching smart contracts is often impossible once they have been deployed on a blockchain as they are usually immutable [31] with some exceptions [1]. This outlines the need for formal verification, as it can be used for correctness-by-construction [12] and minimise the need for costly post-deployment fixes.

In recent years, the field of Artificial Intelligence has made incredible progress, particularly within the realm of Large Language Models (LLMs) like OpenAI's ChatGPT and High-Flyer's DeepSeek. This advancement has opened up new opportunities across all domains, including the formal verification space. In particular, AI for theorem proving has gained traction and has started to see applications outside of purely mathematical statements and instead for program verification. An example of this is FVEL [42], which is used to assist in automated verification of C/C++ programs in the Isabelle proof assistant. This reduces the entry barrier for the formal verification of such programs, making it more accessible and less time-consuming.

In our original paper, we presented IsabeLLM [35], a tool that integrates the proof assistant Isabelle with a Large Language Model to assist and automate proofs. We demonstrated the effectiveness of IsabeLLM by using it to prove the correctness of a novel model for Bitcoin's Proof-of-Work consensus protocol.

We expand on the original paper by introducing IsabeLLM-RAG, an improved version of IsabeLLM. Specifically, we made the following improvements:

(1) Implemented a Retrieval-Augmented Generation (RAG) database of proofs from the binary tree model in [36] for improved domain-specific context.

(2) Implemented the Nitpick Counterexample Generator [10] to identify logically false proof steps.

(3) Implemented an error trace that captures all modifications between LLM calls, providing the history of changes as context and highlighting which proof steps have been corrected and should be preserved.

(4) Compatibility with Isabelle 2025 and general efficiency improvements.

(5) Changed LLM from DeepSeek R1 to DeepSeek R1T2 Chimera for improved efficiency.

We discuss these changes in more detail in Section 5.3.

2 Background

2.1 Blockchain

A blockchain is a decentralised ledger that allows multiple parties to carry out transactions without the need of a trusted intermediary, eliminating the need for trust. This is only possible through a blockchain's consensus protocol, which allows all parties to agree on the current state of the blockchain and the transactions recorded on it. The most popular consensus protocol is Proof of Work (PoW) used by Bitcoin's blockchain, which has around 1.2 billion recorded transactions [11] with Bitcoin's market capitalisation sitting around $1.8 trillion [19]. The core idea of PoW is that the longest blockchain is correct since it assumes the majority of computing power within the system is honest and therefore should be able to solve hashes and add blocks faster than adversaries [46].

2.2 Isabelle

Isabelle is a proof assistant written in Scala and ML that uses Higher-Order Logic (HOL). It is used to write and verify formal proofs with high assurance due to the mechanisation of these proofs [48]. Isabelle's Isar proof language allows these proofs to be more readable than the traditional approach to theorem proving by repeatedly applying tactics. Isabelle also makes use of automation tools like Sledgehammer, which uses external automated theorem provers (ATPs) to help you complete proofs. Outside of the proof assistant itself, the Scala library Scala-Isabelle provides the functionality to interact with an Isabelle process inside of a Scala application [61].

2.3 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is the technique of using external data to improve the performance of LLMs. Although LLMs are great generalist tools, they often struggle in niche domains where there is little available data, resulting in the LLM hallucinating an incorrect answer. RAG aims to combat this by providing a database of domain-specific data to help give the LLM more context and understand the problem.

A RAG framework consists of a retrieval mechanism and a generative LLM. Data is first mapped into a vector space such that the closer data points are to one another, the more similar they are. The retriever searches this vector space to find the most relevant information to a given query and then includes it in the prompt fed to the LLM [40].

In the context of AI for Theorem Proving, RAG is used to identify similar statements to what we are currently trying to prove. By seeing how similar statements have been proven in the past, it helps the LLM understand how to prove the new statement. RAG-based techniques have been used in both Lean [58] and Isabelle [22] for purely mathematical proofs but has yet to be used in the context of formal verification.

3 Related Work

Isabelle has been used extensively in the last 20 years to carry out numerous verifications. Some of the most notable verifications include the seL4 Microkernel [38], the ML compiler [29], and numerous protocol and program verifications [23, 26, 44]. Outside of verification, Isabelle has been used to formalise a large amount of mathematics that can be found in the Archive of Formal Proofs (AFP) [32]. Some of the most notable formalisms in the AFP include Gödel's incompleteness theorems [52], Jordan curve theorem [59], and Ramsey's Theorem [51]. In recent years, Isabelle has been used for verifications and formalisms of blockchain systems, including the Ethereum Virtual Machine [5] and a framework to verify solidity smart contracts [45].

Outside of Isabelle, various other theorem provers have been used to carry out verifications in the blockchain domain. To name a few, Agda [2, 3, 56], Coq [4, 57, 67], and Lean [47, 53] have been used for the formalisms of blockchain. The field has also started to see formalisms of Decentralised Finance (DeFi) components [7–9] but are yet to be mechanised. Other major works within the space include a formalism of the Ethereum Virtual Machine in the K [28], Certora Prover [13] for semi-automated smart contract verification, and fuzzer Echidna [27]. Most recently, PropertyGPT [43] used LLMs trained on Certora auditing reports to formally verify smart contracts. Zero-Knowledge Proof circuits, which are widely used in blockchain systems, have also seen increased interest in formal analysis [63].

The field of AI for theorem proving has seen the development of major data sets for proof assistants in recent years, including IsarStep and PISA for Isabelle [33, 41], LeanDojo for Lean [66], and GamePad and CoqGym for Coq [65]. Using these datasets has allowed for the development of various theorem proving models, including LEGO-Prover [62], LISA [33] and DeepSeek-Prover [64]. Artificial Intelligence for formal verification has seen limited use, with the aforementioned FVEL [42] being the major work in this area. As for AI for formal verification of blockchain, the literature is sparse and has only seen research into extracting smart contract specifications from natural language [39].

4 Model

Our consensus model builds on previous work [36] by generalising the blockchain structure from a binary tree to an n-ary tree. This extension enables the model to account for an arbitrary number of forks at any given block, reflecting a more realistic view of a blockchain. We prove that consensus holds in a majority honest network using the common prefix and chain quality properties outlined in the Bitcoin Backbone Protocol [24], where they are discussed in more detail. We make the same assumptions of majority honesty and synchronisation in the network, meaning that the majority of the computing power in the network is honest and that everyone shares the same view of the blockchain. Moreover, we assume block difficulty remains constant, simplifying Bitcoin's heaviest chain rule to the longest chain rule. We make these assumptions to reduce complexity in our model and prevent the need to switch to a computational model. Whilst such models offer a more realistic verification, the objective of this research is to assess the efficacy of LLMs in verification. This model serves as a sufficient baseline for future verifications of increased complexity.

As in the previous work, we can omit the chain quality property under our majority honesty assumption, leaving us with the common prefix property which states that all honest parties agree on a common chain up to the last k blocks in a chain. This is a safety property, showing honest nodes do not diverge except near the tip of the chain. We first define some preliminaries before specifying our consensus theorem.

Definition 4.1 (n-ary Trees and Height Function). Let $\mathcal{T}$ be the set of n-ary trees. A tree $t \in \mathcal{T}$ is defined as a tuple $t = (x, ts)$ where x is the node ID and $ts = [t_1, t_2, \ldots, t_k]$ is a finite list of subtrees, such that each $t_i \in \mathcal{T}$ and $k \geq 0$. Let $Height : \mathcal{T} \to \mathbb{N}$ be a function returning the height of a tree.

Definition 4.2 (Depth Check Function). For depths $n, d \in \mathbb{N}$, the function $Check : \mathbb{N} \times \mathbb{N} \times \mathcal{T} \to \mathbb{B}$ is defined as follows:

$$Check(0, d, t) = True$$

$$Check(n+1, d, (x, ts)) = \exists p \in ts : \left(Height(p) > d + \max_{p' \in ts \setminus \{p\}} Height(p') \wedge Check(n, d, p) \right)$$

This function ensures that no other branch is less than d nodes away from the longest chain up to depth n of the tree.

Definition 4.3 (Events). An event $e \in \mathcal{E}$ is defined as a tuple $(b, t) \in \mathbb{B} \times \mathcal{T}$, where b is a boolean flag indicating if the event was honest, and t is the state of the tree immediately after the event. $\mathcal{L}(\mathcal{E})$ denotes the set of all finite sequences of events, representing the traces of the blockchain.

Definition 4.4 (Traces). Let $add_h : \mathcal{T} \to \mathcal{T}$ and $add_d : \mathcal{T} \to \mathcal{T}$ be the transition functions for honest (adding to the longest chain) and dishonest (adding to any branch) block additions, respectively. Let $State : \mathcal{E} \to \mathcal{T}$ extract the tree from an event, $hd : \mathcal{L}(\mathcal{E}) \to \mathcal{E}$ return the most recent event in a trace, and $C_{bool} : \mathcal{L}(\mathcal{E}) \to \mathbb{N}$ return the number of events in a trace matching the specified boolean flag.

Given a starting tree t_0 under the assumption that $Check(n, Height(t_0) - n + 1, t_0)$ holds, the inductive set of valid blockchain traces $\mathcal{S} \subseteq \mathcal{L}(\mathcal{E})$ is defined by the following rules:

(1) **Honest Base Case:**

$$[(True, add_h(t_0))] \in \mathcal{S}$$

(2) **Dishonest Base Case:**

$$[(False, add_d(t_0))] \in \mathcal{S}$$

(3) **Honest Inductive Step:** For any existing trace $tr \in \mathcal{S}$:

$$(True, add_h(State(hd(tr)))) :: tr \in \mathcal{S}$$

(4) **Dishonest Inductive Step:** For any existing trace $tr \in \mathcal{S}$, given the condition $C_F(tr) < C_T(tr) + (Height(t_0) - n)$ holds:

$$(False, add_d(State(hd(tr)))) :: tr \in \mathcal{S}$$

Theorem 4.5 (Consensus). *Let $LongestPaths : \mathcal{L}(\mathcal{E}) \to \mathcal{P}(\mathcal{T})$ return the set of the longest branches, and let the function $take : \mathbb{N} \times \mathcal{T} \to \mathbb{N}$ return the prefix of a branch up to the specified depth n. For all valid traces, all longest paths share the same prefix up to depth n:*

$$\forall tr \in \mathcal{S}, \forall p, p' \in LongestPaths(State(hd(tr))) : take(n, p) = take(n, p')$$

This statement is identical to the consensus statement for the binary tree model. However, the generalisation to an n-ary tree significantly increases the complexity of the proof. In the binary tree case, inductive arguments typically require only two cases (e.g., left and right subtrees), whereas the n-ary setting necessitates reasoning over an arbitrary number of branches, complicating case distinctions and inductive reasoning. To show this, Table 1 gives a description for each lemma and the Lines of Proof (LoP) required to complete the verification of each model. We only list the lemmas that were more than one LoP in at least one of the models. In the binary tree column, "N/A" means that the lemma was not required for the verification. For the rows with $x + y$, x is the LoP that are 'original' and y is the LoP that are symmetric to x and are just repeated for the different cases. With this in mind, it is clear that the n-ary tree model has more than double the original LoP when compared to the binary tree model.

Lemma Name	Binary Tree	N-ary Tree	Description
subtree_height	N/A	15	The height of all trees in a set are less than or equal to the highest tree in the set.
height_mono	1+1	23	Monotonicity of tree heights.
obtain_max	N/A	23	In a set of trees, there exists a tree that is at least as high as all other trees.
foldr_max_eq	N/A	37	The maximum height of a set of trees is equal to the height of the tallest tree.
branch_height	N/A	30	The height of a branch of a tree is less than or equal to the height of the tree.
sub_longest	N/A	28	Given the longest branch in a tree, then there exists a subtree where its prefix is a longest branch.
sub_branch	N/A	41	Given the longest branch in a tree, then its prefix is a longest branch for the prefix of the tree.
weaken_distance	1	18	For any tree $t \in \mathcal{T}$, $Check(n, d+1, t), \rightarrow Check(n, d, t)$
weaken_depth	1	15	For any tree $t \in \mathcal{T}$, $Check(n+1, d, t), \rightarrow Check(n, d, t)$
common_prefix	25+12	38	If two branches are longest branches in a tree and no other node exists at a distance less than d with a common ancestor on the path to the root, then the two branches share a common prefix up to depth n.
height_add_mining	10+5	36	When mining on a tree, either the maximum height of the tree increases (added to the longest branch) or it stays the same (added to other branch).
check_add_mining	49+158	1	Given $Check(n, d+1, t)$, then there are 3 cases when mining on the tree: - Add to longest chain ($Height(t') + 1 = Height(t)$) - Add to next longest chain ($Height(t) - Height(t') = d+1$) - Add to any other chain ($Height(t) - Height(t') > d+1$)
height_add_honest	10+5	32	Mining honestly on a tree always results in the tree height increasing (mining on longest chain).
check_add_honest	22+13	36	For tree $t \in \mathcal{T}$, $Check(n, d, t), \rightarrow Check(n, d+1, add_h(t))$
bounded_check	56	17	For any trace $tr \in \mathcal{L}(\mathcal{E})$ and $t_0 \in \mathcal{T}$, $Check(n, C_T(tr) + Height(t_0) - n - C_F(tr), State(hd(tr)))$
consensus	1	5	As stated in Theorem 4.5.
Total	**175+193**	**395**	

Table 1. Lines of Proof (LoP) and Lemma Descriptions for each tree model.

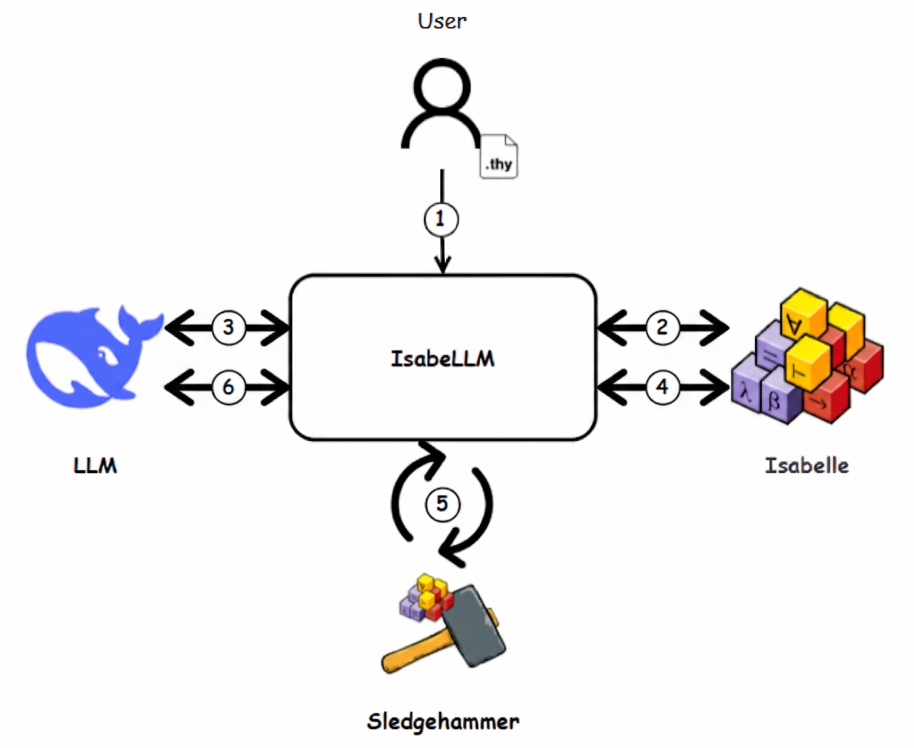

Fig. 1. IsabeLLM Architecture.

5 IsabeLLM

IsabeLLM is an interface between the Isabelle proof assistant and an LLM. It is designed for general purpose and so can be used to prove any kind of statements within Isabelle. It should be noted that if you are using bespoke imports for your theory file, then they should be given to the LLM as context for it to understand. In our models, we are only importing Isabelle's Main library, and everything is contained within the single theory file, meaning we do not need to provide extra context.

Error	Description
"Sorry" Detected in Proof	The sorry keyword is used to mark incomplete proofs. IsabeLLM uses sorry to detect on which part of the theory file to work on.
Failed Proof	The proof failed. IsabeLLM attempts to Sledgehammer before retrying the LLM.
Undefined Fact/Method	LLM hallucinated a fact or method. Hallucinations are removed before rebuilding.
Lexical/Syntax Error	Bad syntax injected into the theory file. IsabeLLM has various methods to detect these issues but will return the proof back to the LLM if it fails.
Timeout	Build exceeds time limit, often due to hanging tactics (e.g., metis, blast). IsabeLLM verifies these tactics using Sledgehammer.

Table 2. Isabelle build errors.

5.1 Architecture

The high-level architecture for IsabeLLM can be seen in Fig. 1. The main idea is that we use an LLM to understand the high-level structure of a proof and then use Isabelle's Sledgehammer tool to solve the intermediate steps that the LLM failed (if any). The general workflow for IsabeLLM is as follows:

1. The user uploads their Isabelle theory file (.thy) to their working directory, along with a ROOT file so that the Isabelle server knows which files to look at. The user starts IsabeLLM.
2. IsabeLLM first uses the Isabelle server to try and build the theory file. If there are no issues with the file and all statements have been proven, then the build completes, and we are done. If not, then IsabeLLM captures the errors raised to identify the unproven statements and extracts them.
3. IsabeLLM sends the context of the theory file and the unproven lemma to the LLM via its API. The LLM tries to prove the lemma and returns a proof of the statement.
4. IsabeLLM injects the new proof into the theory file and tries to build it again. If this fails, we send the file to Isabelle's Sledgehammer tool.
5. Sledgehammer tries to solve each unproven line within the proof. If some are left unproven, then IsabeLLM extracts these lines and their errors, along with the rest of the updated theory file.
6. IsabeLLM returns the current proof state to the LLM and asks it to resolve the remaining errors. The LLM returns a proof of the statement.
7. Steps 4–6 are repeated until the theory file is successfully built or IsabeLLM reaches a set number of iterations.

Fig. 2 shows an example workflow in IsabeLLM. In this example, we prove the lemma subtree_height which states that the height of a tree in a set of trees is always greater than or equal to the maximum height of the set of trees. The LLM generates a proof that fails a proof step, which we then correct with Sledgehammer. The proof is by induction over the list of trees and splitting the inductive step into the cases of whether the tree is at the head of the list or not.

For this paper, we opted to use variations of the DeepSeek R1 model as our LLM due to its strong coding benchmarks and free access to its API. Claude Sonnet was also considered, but was ultimately decided to be too expensive. Other models like OpenAI's GPT-4 and Mistral's Le Chat were also considered but showed poor performance during manual testing. As for our choice of proof assistant, we chose Isabelle due to its existing automation tool Sledgehammer and integration library Scala-Isabelle. Although not integral, the Isar language also helps to understand the logic of the proofs and the dialogue between IsabeLLM and LLM.

5.2 Implementation

Almost all of IsabeLLM is written in Scala, with some Python to access the LLM API using the OpenAI library. The main reason for choosing Scala is to be able to use the Scala-Isabelle library, which offers the functionality to control an Isabelle process from a Scala application. In particular, we make use of Scala-Isabelle for calling Sledgehammer in our theory file. All of our code, including the IsabeLLM source code and theory files, can be found at [34]. We list IsabeLLM's features below:

(1) Interface between Isabelle and a LLM API.

(2) Code extraction from a theory file, including lemmas, definitions, and proofs.

(3) Injection of code into a theory file, including lemmas, definitions, and proofs.

(4) Sledgehammer functionality with a timeout control and option to select which provers it uses. In this paper, we set this timeout to 60 seconds and use the default provers.

(5) Handling of errors in the build process. An Error trace is maintained between LLM calls for improved context.

(6) Records and updates to the LLM chat history.

(7) LLM Prompt Generation.

(8) RAG Database for improved context sent to LLM.

(9) Nitpick counterexample generator to identify logically false statements.

(10) Controls for the maximum number of LLM iterations before timing out to prevent endless loops.

Due to the stochastic nature of LLMs, the most challenging part of automating proof with IsabeLLM is ensuring the output from the LLM has the correct syntax to be injected into Isabelle. Generally speaking, most models understand the syntax well enough to give a coherent response. However, most outputs will trigger at least one error in the build process and must be handled accordingly. Table 2 highlights the general types of error that are encountered when trying to build the theory file after injecting a generated proof.

When sending requests to the LLM, IsabeLLM automatically builds the required prompts to make the context clear. When we first initialise a proof, we send a prompt that includes the context of the lemma we are trying to prove and everything in the theory file before it. After the initialisation prompt, we send prompts that include only the current proof state of the lemma and error trace. Similar proofs extracted from the RAG database are appended to the end of both messages. The templates for these prompts can be seen in Table 3. IsabeLLM also maintains the chat history with the LLM. To minimise the size of our context, we reset the history after successfully proving a lemma, then update the initial context to the theory file with the updated lemma.

Prompt	**Text**
Initialisation	I am trying to complete a proof in Isabelle. Here is my theory file so far: (.thy file). I am trying to prove the following lemma: (lemma). Please prove this lemma. Return only the raw code without any additional text, explanations, formatting, or commentary. Do not include “‘ or language tags. Just the pure code.
Error	Your proof is incorrect. The current proof state is: (proof state). The line: (error line) produced the following error message: (error). Please amend the proof to deal with this error. Since your last output, the system attempted the following automated fixes which resulted in the current state: (error trace). Return only the raw code without any additional text, explanations, formatting, or commentary. Do not include “‘ or language tags. Just the pure code.

Table 3. IsabeLLM prompts.

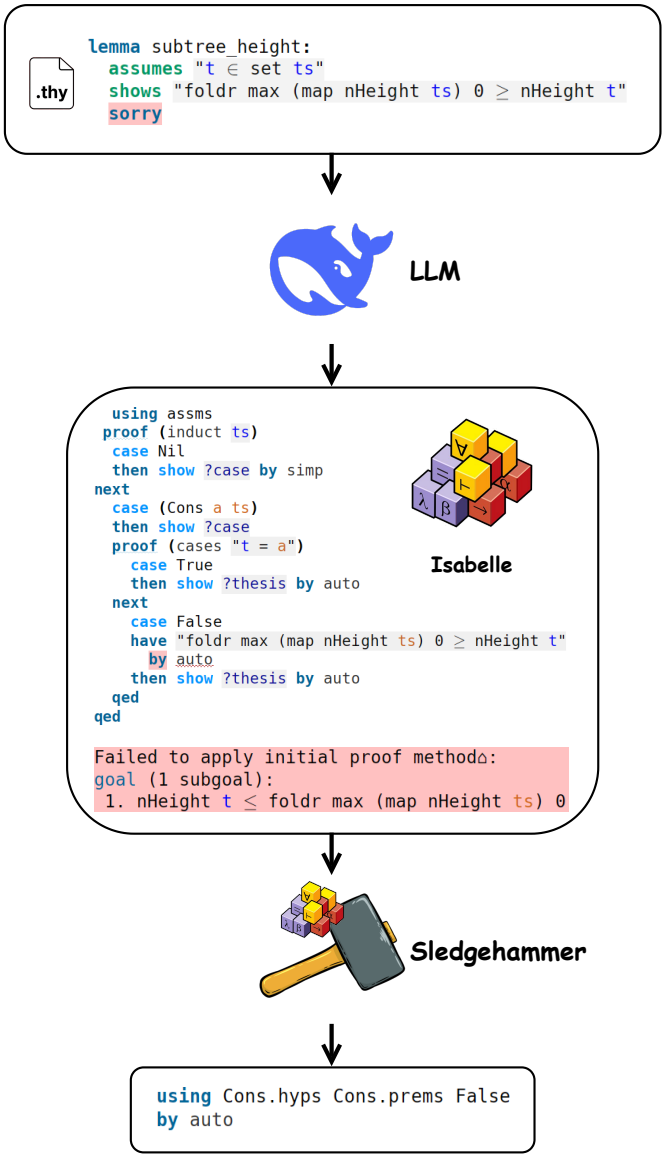

Fig. 2. IsabeLLM example workflow.

5.3 Improvements

This paper improves on the original IsabeLLM tool by making a variety of changes to improve performance and efficiency, most notably by enabling stronger context in our prompts. The first of these is the implementation of a Retrieval-Augmented Generation (RAG) framework. To ensure the LLM receives self-contained context, we process the Isabelle .thy files using regular expressions to isolate definitions and lemmas. We then build a dependency graph and perform a breadth-first search to identify all transitive dependencies (e.g., datatypes and functions) that a lemma relies on. These dependencies are recursively bundled with the proof text before vectorisation. Embeddings are generated using the SentenceTransformers all-MiniLM-L6-v2 model [54] and stored in a local ChromaDB instance. The database consists of all 22 proofs (trivial and non-trivial) from the previous binary tree model in [36]. We query the vector database to find the three most similar lemmas using Euclidean distance as our metric. These lemmas are then injected into the LLM prompt under a dedicated reference header.

We also integrated the Nitpick counterexample generator [10]. In the previous paper we found that LLMs would repeatedly try to prove statements that were logically false. Nitpick identifies these statements which we can then provide as context to the LLM to help it understand why the proof does not work and adapt to a new approach. Fig. 3 shows an example counter example. Here, the LLM is trying to prove the intermediate step that the height of tree t is strictly less than the maximum height of all trees in the set ts. However, nitpick identifies a counterexample where t has the maximum height and so the statement must be false.

```
Nitpicking goal:
  t ∈ set ts ⟹
  (t ∈ set ys ⟹ nHeight t ≤ foldr max (map nHeight ys) 0) ⟹
  t ∈ set (y # ys) ⟹ t ≠ y ⟹ nHeight t < foldr max (map nHeight ts) 0
Nitpick found a counterexample for card 'a = 3:
  Free variables:
    t = nNode a₁ []
    ts = [nNode a₁ []]
    y = nNode a₂ []
    ys = [nNode a₁ []]
```

Fig. 3. Nitpick Example

Lastly, we implemented an error trace that is added context to the LLM. In the previous paper we noted the LLM would often override changes we made between LLM calls and get stuck in a loop by repeatedly changing proof steps we have already corrected. The trace provides the exact changes that have been made since the last LLM call and highlights what should be preserved, enabling the LLM to build on our improvements instead of trying to start from scratch each call.

Next, we aimed to improve the efficiency of IsabeLLM. First, we made the tool compatible with Isabelle 2025. The original tool used Isabelle 2022 as this was the version that the Scala-Isabelle package [61] supported. Since then, the package has been updated to be compatible with the latest Isabelle versions and its associated tools, including Sledgehammer. In the previous paper we noted that the older version of Sledgehammer failed to find proofs in the allotted time but the Isabelle 2025 sledgehammer succeeded. Updating to the latest version enables us to use these more efficient tools, improving our ability to find proofs. Moreover, the original IsabeLLM ran different processes for the Isabelle Server and Sledgehammer. We have updated this to consolidate the whole tool into a single process, massively improving the speed of computation and significantly reducing the required memory.

5.4 Model Selection

In this paper, we test IsabeLLM-RAG with 3 different models. Namely, these are DeepSeek R1T2 Chimera [60], NVIDIA-Nemotron-3-Super-120B-A12B [49] and OpenAI's gpt-oss-120b [50]. We summarise their architectures in Table 4. The original IsabeLLM paper used DeepSeek R1 as its model, using a standard Mixture-of-Experts (MoE) transformer architecture with 671B total parameters and around 37B active parameters, optimised via reinforcement learning for pure reasoning. In contrast, DeepSeek R1T2 Chimera maintains the same parameter footprint but employs an Assembly of Experts (AoE) methodology [37], combining the R1, V3-0324, and R1-0528 models to stabilise outputs. It also offers a slightly larger context window. NVIDIA Nemotron 3 Super introduces a MoE Hybrid Mamba-Transformer architecture with 120B total and 12B active parameters, leveraging LatentMoE routing [21] and Multi-Token Prediction (MTP) [25] to achieve a 1M token context window. Finally, GPT-OSS-120B uses a traditional MoE transformer with 117B total and 5.1B active parameters, with MXFP4 quantisation and a 128K context window.

Our DeepSeek models are by far the largest in terms of total parameters, roughly 5 times the size of our NVIDIA and OpenAI models. From this, we would expect our DeepSeek models to outperform the others. Nemotron and GPT-OSS are of similar size, with Nemotron being the most recent model and making use of recent innovations to enable a massive context window. Moreover, GPT-OSS is quantised with MXFP4, meaning it operates at lower precision to significantly reduce its memory footprint, while activating far fewer parameters (5.1B) than our other models.

While the Chimera model is arguably less advanced compared to the pure reasoning of the R1 model, it is much more stable and efficient. The reason for this switch is that we noted one of the main efficiency bottlenecks of the tool is the slow response time of the LLM. In this case, the speed increase is worth the slight decrease in performance, especially when our other improvements help mitigate this decrease.

While the baseline R1 model is highly optimised for pure mathematical reasoning, we hypothesised that the AoE methodology in the Chimera architecture would provide greater syntax stability and inference efficiency. As noted, a primary bottleneck of IsabeLLM is the latency of LLM generation. Reviewing our OpenRouter API logs, DeepSeek R1's speed typically ranged between $10-20$ tokens per second (tok/s). In contrast, Chimera's speed typically ranged between $30-50$ tok/s, representing a significant latency improvement. Furthermore, as demonstrated in Section 6, Chimera also minimised the number of corrective loops required, achieving the lowest average number of LLM iterations per successful proof (1.06 compared to 1.31 for R1). Both Nemotron and GPT-OSS exhibited a similar generation speed to the Chimera model.

Table 4. Model Architectures

Model	Architecture	Total Parameters	Active Parameters	Context Window
DeepSeek R1	MoE Transformer	671B	~37B	128K
DeepSeek R1T2 Chimera	AoE Transformer	671B	~37B	163.8K
NVIDIA Nemotron 3 Super	MoE Hybrid Mamba-Transformer	120B	12B	1M
GPT-OSS-120B	MoE Transformer	117B	5.1B	128K

6 Results

We tested IsabeLLM and IsabeLLM-RAG by using them to prove each of the 16 non-trivial lemmas listed in Table 1. We attempt each proof 10 times with a maximum of 3 iterations per attempt, not counting instances when the LLM would return an empty output as this is an API failure rather than a methodological failure. We reduced the maximum number of iterations from 5 in the first paper as we discovered that the additional iterations were almost always ineffective. We consider an attempt to be a failure if it exceeds the maximum number of iterations or exits prematurely (often due to syntax issues). It should be noted that the LoP specified for each lemma can vary as the LLM can generate different proofs for the same statement.

The original IsabeLLM used DeepSeek R1 as its LLM. To evaluate the impact of underlying model architecture on our retrieval-augmented framework, IsabeLLM-RAG was tested using DeepSeek R1T2 Chimera, NVIDIA-Nemotron-3-Super-120B-A12B, and OpenAI's gpt-oss-120b. As discussed in Section 5.3, IsabeLLM-RAG benefits from the RAG database, error trace, counterexample generator, and access to the Isabelle 2025 Sledgehammer. All experiments were conducted on a workstation running Windows 11. The system was equipped with an AMD Ryzen 7 5825U processor (8 cores, 16 threads) and 16.0 GB of RAM.

Table 5 shows the results of each version of IsabeLLM. Recall that in the previous paper, for the lemma branch_height, we had to manually use the 2025 Sledgehammer to succeed. These manual interventions have been removed so that we can more clearly see the automated improvements in IsabeLLM-RAG.

Generally speaking, IsabeLLM-RAG outperforms the original across all lemmas when using the Chimera and Nemotron models. It matches or improves the success rate for nearly every lemma while decreasing the average number of iterations per successful attempt. Overall, the Chimera model achieved a 94.4% success rate with a significant reduction in LLM iterations. Nemotron achieved a 87.5% success rate, outperforming IsabeLLM despite it using a much larger model. The most notable improvement is for the lemma branch_height, which previously failed without manual Sledgehammer intervention. IsabeLLM-RAG (Chimera) achieved a 90% success rate here, needing only 1 LLM call in almost every successful attempt. Conversely, the GPT-OSS model struggled, yielding a 67.5% average success rate and frequently requiring the maximum number of iterations. However, this is not far behind how IsabeLLM performed with DeepSeek R1, a much larger model without the precision loss of quantisation.

Similar to the original paper, IsabeLLM-RAG rarely obtained the correct answer on the very first LLM generation for complex lemmas; IsabeLLM had to apply various error corrections and leverage Sledgehammer to fully complete the proofs. Interestingly, where the original paper saw some variation in proof approaches, particularly for larger proofs, IsabeLLM-RAG produced highly deterministic structural approaches across iterations.

7 Discussion

We first examine the variance in success rates across the models. The five lemmas that IsabeLLM-RAG (Chimera) failed to prove with a 100% success rate have the largest LoP, showing that increased logical complexity inherently reduces success rate. Exceptions to this are the honest mining variants of height_add and check_add, which both require a high number of LoP yet achieved perfect completions across Chimera and Nemotron. This success is likely attributable to the RAG system. As discussed in Section 4, the two tree models have different approaches to proof; however, the logic of adapting the proofs from the mining variant to the honest mining variant remains consistent in both models. The

Table 5. Performance comparison between IsabeLLM Versions.(SR = Success Rate, Iter. = Avg. Iterations)

		IsabeLLM		**IsabeLLM-RAG**					
		R1		**Chimera**		**Nemotron**		**GPT-OSS**	
Lemma Name	**Lines**	**SR**	**Iter.**	**SR**	**Iter.**	**SR**	**Iter.**	**SR**	**Iter.**
subtree_height	15	100%	1.0	100%	1.0	100%	1.0	100%	1.0
height_mono	23	100%	1.0	100%	1.0	100%	1.0	100%	1.0
obtain_max	23	90%	1.4	100%	1.0	100%	1.0	100%	1.0
foldr_max_eq	37	50%	2.0	100%	1.0	80%	1.1	30%	3.0
branch_height	30	0%	N/A	90%	1.1	50%	1.2	60%	2.0
sub_longest	28	70%	1.1	90%	1.1	100%	1.0	40%	1.0
sub_branch	41	50%	1.8	80%	1.1	50%	1.4	40%	1.0
weaken_distance	18	100%	1.0	100%	1.0	100%	1.0	100%	1.0
weaken_depth	15	100%	1.0	100%	1.0	100%	1.0	100%	1.0
common_prefix	38	60%	1.5	70%	1.4	50%	1.2	0%	N/A
height_add_mining	36	60%	2.0	80%	1.1	70%	2.0	20%	2.0
check_add_mining	1	100%	1.0	100%	1.0	100%	1.0	100%	1.0
height_add_honest	32	80%	1.7	100%	1.2	100%	1.5	100%	2.5
check_add_honest	36	90%	1.2	100%	1.0	100%	1.0	40%	2.0
bounded_check	17	70%	1.0	100%	1.0	100%	1.0	50%	1.1
consensus	5	100%	1.0	100%	1.0	100%	1.0	100%	1.0
Average	**-**	**76.3%**	**1.31**	**94.4%**	**1.06**	**87.5%**	**1.15**	**67.5%**	**1.44**

LLM identifies this structural symmetry from the RAG context and applies it to prove these lemmas reliably. On the other hand, three of the failed proofs were unique to the n-ary tree model, meaning there was no analogous proof in the binary tree database. It is evident that the RAG context is less effective here as it cannot provide a good template.

As expected, our largest model Chimera performed best, achieving the highest success rate and lowest average iterations. Interestingly, Nemotron performed exceedingly well despite its small size. Despite having a large context window, our prompt lengths rarely exceeded 12K tokens, indicating its success does not stem from context capacity. We attribute its performance primarily to its usage of MTP, which forces the model to learn the broader structural syntax of Isar proofs rather than prioritising myopic next-token prediction. GPT-OSS's weaker performance highlights its architectural limitations. Its lower active parameter count and quantisation appear to restrict its capacity for deep reasoning, context retention and syntax consistency.

The lower variability observed in LLM responses is attributable to RAG. The use of RAG effectively prunes the potential proof space, providing a strict template on how to approach the proof and drastically reducing the chance of hallucination. Despite the original DeepSeek R1 having an advantage in pure mathematical reasoning, we saw no instance where it outperformed the Chimera model. However, this is also influenced by other improvements in IsabeLLM-RAG.

In terms of overall efficiency, IsabeLLM-RAG exceeded its predecessor. One of the highlighted issues in the original paper was the LLM getting stuck in a proof loop, repeatedly failing on the same step. Proof loops were effectively mitigated across the Chimera and Nemotron tests due to the stronger context supplied by Nitpick and the error trace, allowing the LLM to learn from immediate failure. It was also noted that the API interactions were far more reliable.

As before, IsabeLLM is still limited to proof completion, unable to generate the initial properties to be proven. This means that the user must specify a statement before it can be verified, including setting up functions, locales, and sets. However, this issue will be largely mitigated if IsabeLLM is used in conjunction with existing verification frameworks rather than building from the ground up. For example, Isabelle/Solidity [45] builds most of the model automatically, leaving only the invariant properties to be specified. A remaining challenge is ensuring the LLM understands the bespoke calculus of such frameworks, given the scarcity of relevant proof corpora.

While the primary objective of IsabeLLM-RAG is the full automation of the verification process, the system inherently facilitates proof discovery. All proofs use Isabelle's Isar proof language that is highly readable, allowing users to follow the logic of the proof and gain insight into potentially new approaches. Furthermore, the integration of Nitpick enables the discovery of previously unconsidered edge cases through counterexample generation, effectively highlighting logical flaws in proof steps.

While our current RAG implementation provides static templates based on prior successful proofs, scaling IsabeLLM to larger verifications involving multiple Isabelle theory files proves to be a challenge. A large context becomes infeasible to include in our prompt due to the limited context window, causing our RAG methodology to break down. In this scenario, integrating the Model Context Protocol (MCP) [6] presents a complementary solution to our RAG strategy. Whereas RAG effectively anchors the overarching architecture of a proof by matching structural templates, a MCP-based approach would act as a dynamic retrieval mechanism. By equipping the LLM with MCP tools, the model could actively query Isabelle's environment during inference. This dynamic retrieval would mitigate the context window bottleneck by fetching only the necessary facts.

8 Conclusion

In this paper, we improved the original IsabeLLM tool with IsabeLLM-RAG, implementing a RAG database, counterexample generator (Nitpick), improved conversational error trace, and compatibility with Isabelle 2025 and its associated tools. We evaluated IsabeLLM-RAG's performance using three models - DeepSeek R1T2 Chimera, NVIDIA Nemotron 3 Super, and GPT-OSS-120B. With these improvements, we were able to outperform the original tool across the board using the Chimera and Nemotron architectures, increasing success rates and decreasing the average number of LLM iterations required. Although GPT-OSS showed weaker performance, it was not far behind the performance of the original IsabeLLM tool, which used a much larger model.

While our results highlight IsabeLLM's capacity to evolve alongside state-of-the-art AI and theorem-proving infrastructure, several avenues for future work remain. IsabeLLM could be parallelised to query different LLMs simultaneously,

generating a distributed proof tree to maximise the probability of finding a valid proof path. Furthermore, the LLMs could be fine-tuned on domain-specific data, such as the Archive of Formal Proofs or bespoke consensus verifications. Moreover, IsabeLLM could be integrated with semantic embeddings of programming languages within Isabelle, the most prominent in the blockchain space being Isabelle/Solidity. This would transition IsabeLLM from verifying abstract state-machine models to verifying concrete smart contract bytecode, solidifying its viability in real-world, safety-critical applications. Conversely, because IsabeLLM remains a general-purpose proof assistant wrapper, its methodologies could be readily translated to formal verifications in domains entirely outside the blockchain space.

References

[1] Alchemy. 2024. What are upgradeable smart contracts? https://docs.alchemy.com/docs/upgradeable-smart-contracts [Accessed April 2025].

[2] Fahad Alhabardi and Anton Setzer. 2023. A simulator of Solidity-style smart contracts in the theorem prover Agda. In *Proceedings of the 2023 6th International Conference on Blockchain Technology and Applications (ICBTA)*. IEEE, Xi'an, China, 1–11.

[3] Fahad F Alhabardi, Arnold Beckmann, Bogdan Lazar, and Anton Setzer. 2022. Verification of Bitcoin Script in Agda using weakest preconditions for access control. arXiv:2203.03054 [cs.CR] Preprint.

[4] Musab A Alturki, Jing Chen, Victor Luchangco, Brandon Moore, Karl Palmskog, Lucas Peña, and Grigore Roşu. 2019. Towards a verified model of the Algorand consensus protocol in coq. In *International Symposium on Formal Methods (Lecture Notes in Computer Science, Vol. 11800)*. Springer, Cham, 362–367.

[5] Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. 2018. Towards verifying Ethereum smart contract bytecode in Isabelle/HOL. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*. ACM, New York, NY, USA, 66–77.

[6] Anthropic. 2024. *Model Context Protocol (MCP)*. https://modelcontextprotocol.io Accessed: April 2026.

[7] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. 2021. Towards a theory of decentralized finance. In *Financial Cryptography and Data Security Workshops (FC 2021)*. Springer, Berlin, Heidelberg, 227–232.

[8] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. 2022. A theory of automated market makers in DeFi. *Logical Methods in Computer Science* 18, 4 (2022), 12:1–12:35.

[9] Massimo Bartoletti and Roberto Zunino. 2023. A theoretical basis for blockchain extractable value. arXiv:2302.02154 [cs.CR] Preprint.

[10] Jasmin Christian Blanchette and Tobias Nipkow. 2010. Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder. In *Interactive Theorem Proving (ITP) (Lecture Notes in Computer Science, Vol. 6172)*. Springer, Berlin, Heidelberg, 131–146.

[11] Blockchain.com. 2025. BTC Total Number of Transactions. https://www.blockchain.com/explorer/charts/n-transactions-total [Accessed April 2025].

[12] Tabea Bordis, Tobias Runge, Alexander Kittelmann, and Ina Schaefer. 2023. Correctness-by-Construction: An Overview of the CorC Ecosystem. *Ada Letters* 42, 2 (2023), 75–78.

[13] Certora. 2024. Certora Prover. https://www.certora.com/prover [Accessed April 2025].

[14] Chainalysis. 2023. The $80 Million Qubit Hack Likely the Work of North Korea-linked Cybercriminals. https://www.chainalysis.com/blog/qubit-hack-north-korea/ [Accessed April 2025].

[15] CoinBase. 2016. Ethereum Classic and the Ethereum hard fork. https://help.coinbase.com/en/coinbase/getting-started/crypto-education/eth-hard-fork [Accessed April 2025].

[16] CoinDesk. 2019. The Vertcoin Cryptocurrency Just Got 51% Attacked – Again. https://www.coindesk.com/tech/2019/12/02/the-vertcoin-cryptocurrency-just-got-51-attacked-again [Accessed April 2025].

[17] CoinDesk. 2020. Bad Actors Rent Hashing Power to Hit Bitcoin Gold With New 51% Attacks. https://www.coindesk.com/tech/2020/01/27/bad-actors-rent-hashing-power-to-hit-bitcoin-gold-with-new-51-attacks [Accessed April 2025].

[18] CoinDesk. 2021. Ethereum Classic Hit by Third 51% Attack in a Month. https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month [Accessed April 2025].

[19] CoinMarketCap. 2025. BTC Market Capitalisation. https://coinmarketcap.com/currencies/bitcoin/ [Accessed April 2025].

[20] Decrypt. 2024. BNB Chain Hits Record-High Sandwich Attacks Exposing $1.5 billion in Trades. https://decrypt.co/294648/bnb-smart-chain-blocks-hits-record-high-sandwich-attacks [Accessed April 2025].

[21] Venmugil Elango, Nidhi Bhatia, Roger Waleffe, Rasoul Shafipour, Tomer Asida, Abhinav Khattar, Nave Assaf, Maximilian Golub, Joey Guman, Tiyasa Mitra, et al. 2026. LatentMoE: Toward Optimal Accuracy per FLOP and Parameter in Mixture of Experts. *arXiv preprint arXiv:2601.18089* (2026).

[22] Emily First, Markus Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-Proof Generation and Repair with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, New York, NY, USA, 1229–1241.

[23] Simon Foster, Jonathan Julián Huerta y Munive, Mario Gleirscher, and Georg Struth. 2021. Hybrid systems verification with Isabelle/HOL: Simpler syntax, better models, faster proofs. In *Formal Methods: 24th International Symposium (FM 2021) (Lecture Notes in Computer Science, Vol. 13047)*. Springer, Cham, 367–386.

[24] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT) (Lecture Notes in Computer Science, Vol. 9057)*. Springer, Sofia, Bulgaria, 281–310.

[25] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737* (2024).

[26] Victor BF Gomes, Martin Kleppmann, Dominic P Mulligan, and Alastair R Beresford. 2017. Verifying strong eventual consistency in distributed systems. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–28.

[27] Gustavo Grieco, Will Song, Artur Cygan, Josselin Feist, and Alex Groce. 2020. Echidna: effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*. 557–560.

[28] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. 2018. Kevm: A complete formal semantics of the Ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, Oxford, UK, 204–217.

[29] Lars Hupel and Tobias Nipkow. 2018. A verified compiler from Isabelle/HOL to CakeML. In *European Symposium on Programming (ESOP) (Lecture Notes in Computer Science, Vol. 10801)*. Springer, Cham, 999–1026.

[30] Investopedia. 2022. Crypto Worth Over $320 Million Taken in Wormhole Hack. https://www.investopedia.com/crypto-theft-of-usd320-million-wormhole-hack-5218062 [Accessed April 2025].

[31] Investopedia. 2024. What Are Smart Contracts on the Blockchain and How Do They Work? https://www.investopedia.com/terms/s/smart-contracts.asp [Accessed April 2025].

[32] Isabelle/HOL. 2004. Archive of Formal Proofs. https://www.isa-afp.org/ [Accessed April 2025].

[33] Albert Q. Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. 2021. LISA: Language Models of ISAbelle Proofs. In *6th Conference on Artificial Intelligence and Theorem Proving (AITP)*. University of Innsbruck, Innsbruck, Austria, 378–392.

[34] Elliot Jones. 2025. IsabeLLM. https://github.com/EllbellCode/IsabeLLM GitHub Repository.

[35] Elliot Jones and William Knottenbelt. 2025. Towards Automating Blockchain Consensus Verification with IsabeLLM. In *International Conference on Blockchain Technology and Emerging Applications (Communications in Computer and Information Science, Vol. 2100)*. Springer, Cham, 3–18.

[36] Elliot Jones and Diego Marmsoler. 2024. Towards Mechanised Consensus in Isabelle. In *5th International Workshop on Formal Methods for Blockchains (FMBC 2024) (OASIcs, Vol. 117)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 4:1–4:22.

[37] Henrik Klagges, Fabian Klemm, Benjamin Merkel, Daniel Klingmann, David A. Reiss, and Dan Zecha. 2025. Assembly of Experts: Linear-time construction of the Chimera LLM variants with emergent and adaptable behaviors. arXiv:2506.14794 [cs.CL] Preprint.

[38] Gerwin Klein, Thomas Sewell, and Simon Winwood. 2010. Refinement in the Formal Verification of the seL4 Microkernel. In *Design and Verification of Microprocessor Systems for High-Assurance Applications*, David S. Warren (Ed.). Springer, Boston, MA, 323–339.

[39] Gabriel Leite, Filipe Arruda, Pedro Antonino, Augusto Sampaio, and AW Roscoe. 2024. Extracting formal smart-contract specifications from natural language with LLMs. In *International Conference on Formal Aspects of Component Software (Lecture Notes in Computer Science, Vol. 14330)*. Springer, Cham, 109–126.

[40] Patrick Lewis et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33. Curran Associates, Inc., New York, NY, USA, 9459–9474.

[41] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. 2020. Isarstep: a benchmark for high-level mathematical reasoning. arXiv:2006.09265 [cs.LO] Preprint.

[42] Xiaohan Lin, Qingxing Cao, Yinya Huang, Haiming Wang, Jianqiao Lu, Zhengying Liu, Linqi Song, and Xiaodan Liang. 2024. FVEL: Interactive Formal Verification Environment with Large Language Models via Theorem Proving. arXiv:2406.14408 https://arxiv.org/abs/2406.14408 Preprint.

[43] Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, and Yang Liu. 2024. Propertygpt: LLM-driven formal verification of smart contracts through retrieval-augmented property generation. *arXiv preprint arXiv:2405.02580* (2024).

[44] Filip Marić. 2010. Formal verification of a modern SAT solver by shallow embedding into Isabelle/HOL. *Theoretical Computer Science* 411, 50 (2010), 4333–4356.

[45] Diego Marmsoler and Achim D Brucker. 2021. A denotational semantics of Solidity in Isabelle/HOL. In *International Conference on Software Engineering and Formal Methods (SEFM) (Lecture Notes in Computer Science, Vol. 13085)*. Springer, Cham, 403–422.

[46] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf Whitepaper.

[47] Nethermind. 2024. Clear–Prove Anything About Your Solidity Smart Contracts. https://www.nethermind.io/blog/clear-prove-anything-about-your-solidity-smart-contracts [Accessed April 2025].

[48] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Vol. 2283. Springer, Berlin, Heidelberg.

[49] NVIDIA. 2026. NVIDIA-Nemotron-3-Super-120B-A12B. https://build.nvidia.com/nvidia/nemotron-3-super-120b-a12b/modelcard

[50] OpenAI. 2025. gpt-oss-120b. https://huggingface.co/openai/gpt-oss-120b

[51] Lawrence C Paulson. 2004. Theory Ramsey. https://isabelle.in.tum.de/website-Isabelle2021-1/dist/library/HOL/HOL-Library/Ramsey.html Isabelle/HOL Library.

[52] Lawrence C Paulson. 2015. A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. *Journal of Automated Reasoning* 55, 1 (2015), 1–37.

[53] Daniele Pusceddu and Massimo Bartoletti. 2024. Formalizing Automated Market Makers in the Lean 4 Theorem Prover. arXiv:2402.06064 [cs.LO] Preprint.
[54] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://arxiv.org/abs/1908.10084
[55] Reuters. 2021. How hackers stole $613 million in crypto tokens from Poly Network. https://www.reuters.com/technology/how-hackers-stole-613-million-crypto-tokens-poly-network-2021-08-12/ [Accessed April 2025].
[56] Anton Setzer. 2018. Modelling bitcoin in Agda. arXiv:1804.06398 [cs.CR] Preprint.
[57] Tianyu Sun and Wensheng Yu. 2020. A formal verification framework for security issues of blockchain smart contracts. *Electronics* 9, 2 (2020), 255.
[58] Amitayush Thakur, Yuhuai Wu, Kellin Pelrine, Brett Holmgren, Janice Ahn, Zong Zhang, Jean-Stanislas Denain, Kevin Clark, Noah Smith, Alexandre Drouin, and Edward Grefenstette. 2023. A Language-Agent Approach to Formal Theorem-Proving. arXiv:2305.07641 [cs.AI] https://arxiv.org/abs/2305.07641
[59] René Thiemann and Akihisa Yamada. 2016. Formalizing Jordan normal forms in Isabelle/HOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP)*. ACM, New York, NY, USA, 88–99.
[60] TNG Technology Consulting GmbH. 2025. DeepSeek-TNG R1T2 Chimera. https://huggingface.co/tngtech/DeepSeek-TNG-R1T2-Chimera Model version 1.0.
[61] Dominique Unruh. 2022. scala-isabelle – a Scala library for controlling Isabelle/HOL. https://dominique-unruh.github.io/scala-isabelle/ [Accessed April 2025].
[62] Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, et al. 2023. Lego-prover: Neural theorem proving with growing libraries. arXiv:2310.00656 [cs.LG] Preprint.
[63] Hongbo Wen, Jon Stephens, Yanju Chen, Kostas Ferles, Shankara Pailoor, Kyle Charbonnet, Isil Dillig, and Yu Feng. 2024. Practical security analysis of Zero-Knowledge proof circuits. In *33rd USENIX Security Symposium (USENIX Security 24)*. 1471–1487.
[64] Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. Deepseek-prover: Advancing theorem proving in LLMs through large-scale synthetic data. arXiv:2405.14333 [cs.CL] Preprint.
[65] Kaiyu Yang and Jia Deng. 2019. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*. PMLR, Long Beach, CA, USA, 6984–6994.
[66] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2023. LeanDojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems* 36 (2023), 21573–21612.
[67] Zheng Yang, Hang Lei, and Weizhong Qian. 2020. A hybrid formal verification system in coq for ensuring the reliability and security of ethereum-based service smart contracts. *IEEE Access* 8 (2020), 21411–21436.

Appendix

All relevant code for IsabeLLM can be found at:

https://github.com/EllbellCode/IsabeLLM

The repository includes:

- Source code for IsabeLLM.
- Isabelle theory files for the n-ary tree PoW model.
- Setup instructions.