

# Mining Tactics for Automated Theorem Proving

Jian Fang

Key Laboratory of High Confidence  
Software Technologies (Peking  
University), Ministry of Education;  
School of Computer Science, Peking  
University  
Beijing, China  
fangjian@stu.pku.edu.cn

Yixun Yao

Key Laboratory of High Confidence  
Software Technologies (Peking  
University), Ministry of Education;  
School of Computer Science, Peking  
University  
Beijing, China  
yaoyixun@stu.pku.edu.cn

Yingfei Xiong\*

Key Laboratory of High Confidence  
Software Technologies (Peking  
University), Ministry of Education;  
School of Computer Science, Peking  
University; Zhongguancun  
Laboratory  
Beijing, China  
xiongyf@pku.edu.cn

## Abstract

Automated theorem proving is essential for the formal verification of safety-critical systems. As the corpus of formal proofs grows, a natural paradigm is to learn from existing proofs. However, current learning-based approaches predominantly train Large Language Models (LLMs) as end-to-end provers, which yields resource-intensive, opaque systems. Conversely, while traditional symbolic provers are computationally efficient, how to automatically improve these solvers from data remains an open challenge.

This paper bridges this gap by proposing LLM2LTAC, a novel approach that leverages the reasoning power of LLMs not as end-to-end provers, but as intelligent synthesizers to mine purely symbolic tactics from data. Given a corpus of formal proofs, LLM2LTAC asks an LLM to identify latent proof strategies and formalize them into reusable tactics. These tactics are verified for validity and generalizability, and finally integrated into symbolic provers to enhance their automated proving capabilities without the runtime cost of LLMs.

We implement LLM2LTAC on Rocq 8.20.0 and mine tactics from 11,725 theorems in the standard library. We evaluate our approach on 6,199 theorems from four large real-world verification projects, namely, CompCert, Coq-ART, EXT-LIB, and VFA. Results show that the mined tactics improve CoqHammer to prove 23.87% more theorems, and when integrating the improved CoqHammer with Claude Code, the overall proved theorems increases by 9.90%, indicating the effectiveness of LLM2LTAC.

## CCS Concepts

• **Theory of computation** → **Automated reasoning**; • **Software and its engineering** → *Formal software verification*.

## Keywords

Automated Theorem Proving, Large Language Models, Tactic Mining, Formal Verification.

\*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.  
ASE '26, Munich, Germany  
© 2026 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-2882-2/2026/10  
<https://doi.org/10.1145/3832783.3837528>

## ACM Reference Format:

Jian Fang, Yixun Yao, and Yingfei Xiong. 2026. Mining Tactics for Automated Theorem Proving. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26)*, October 12–16, 2026, Munich, Germany. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3832783.3837528>

## 1 Introduction

Automating formal verification is important for ensuring the correctness of safety-critical software [19, 22, 25] and hardware systems [37, 38]. Interactive theorem provers (ITPs) such as Rocq (Coq) [39], Isabelle [43], and Lean [13] provide rigorous mathematical frameworks for constructing machine-checked proofs. However, manual proof construction requires substantial expertise and effort, significantly limiting the scalability of formal verification to large-scale industrial applications.

The state-of-the-art approach to automating formal verification combines large language models (LLMs) with symbolic automated theorem provers (ATPs) [21, 26, 40]. Though many efforts are devoted to improving LLM performance in these proving agents [21, 23, 26, 30, 40], we argue that enhancing symbolic provers themselves remains equally important. First, as revealed in existing studies [10], symbolic provers complement LLMs effectively. If the improvement falls outside the proving capabilities of LLMs, the overall performance of the proving agent improves. Second, even if the improvement of symbolic provers falls within the capabilities of LLMs, we can reduce the expensive cost of invoking LLMs by offloading proving tasks to symbolic provers. Third, in many real-world scenarios, LLM-based approaches cannot be used at all. For example, small companies cannot afford the cost of their own LLM servers, and cannot use public services due to confidentiality concerns. Therefore, this paper aims to improve the performance of symbolic provers for automated theorem proving.

A key observation is that existing symbolic provers rely on the existing decision procedures [4, 14], and do not work well in domains that lack decision procedures. However, in many such domains, existing proofs have already been written by human developers, and these proofs embed reusable proof strategies that serve as practical alternatives to the missing decision procedures. Based on this observation, our basic idea is to enable symbolic provers to learn from existing proofs: we mine the proof strategies from existing proofs, formalize them as reusable tactics, and equip symbolic provers with this mined knowledge to handle cases where they would otherwise fail.

We propose LLM2LTAC, a novel approach that improves symbolic provers by using LLMs to mine reusable proof strategies from existing formal proofs and formalize them into executable tactics. Note that LLM2LTAC does not train or fine-tune any neural network model; rather, it uses LLMs as a mining tool to surface proof strategies that already exist in proofs and make them accessible to symbolic provers. Concretely, the workflow of LLM2LTAC is as follows: i) we collect a corpus of formal proofs from existing verification projects as the knowledge source; ii) the LLM analyzes each proof to understand the underlying reasoning process; iii) the LLM identifies generalizable proof strategies embedded in the proof; iv) the LLM formalizes each strategy as a symbolic tactic; v) the mined tactics are integrated into the symbolic prover, enabling it to apply this acquired knowledge for automated proving.

Key challenges in this approach are: i) How can we ensure that mined tactics are useful? Many tactics generated by LLMs may be syntactically incorrect, or they may be overfitted to the specific proof from which they were mined, failing to generalize to new problems. ii) How can we retrieve appropriate tactics for a given proof state? Traditional retrieval methods such as BM25 [34] rely on textual similarity. Since tactics are a structured representation with limited textual content, textual similarity measures are insufficient for identifying relevant tactics effectively.

To address the first challenge, we design a testing process that ensures: i) the tactic can be correctly represented in the symbolic system; and ii) the tactic is generalizable by demonstrating applicability to other theorems outside the mining dataset. To address the second challenge, we record the theorem from which each tactic was mined, and retrieve tactics by using BM25 to match the theorems that the tactics are mined from. By indexing tactics via their source theorems, we effectively enrich the text for retrieval, as theorem statements carry richer textual information than tactics themselves.

**Evaluation.** To evaluate LLM2LTAC, we collect 11,725 theorems from Rocq standard library as the dataset for tactic mining. We choose the standard library because it contains a substantial number of theorems on commonly used data structures such as natural numbers, lists, and real numbers. To evaluate generalization capability, we collect a validation set containing 6,462 theorems from C-CoRN [11], including common algebraic structures (e.g., setoids, monoids, groups). We randomly select 93,424 different positions across these theorems' proof scripts to assess how broadly the mined tactic applies under different proof contexts. To evaluate the enhancement to the symbolic prover, we select large, real-world open-source Rocq projects focused on software verification: COQ-ART [5], EXT-LIB [28], VFA [3], and COMPCERT [24], in total 6199 theorems. We collect a testing benchmark of 6199 theorems from these four benchmarks, and evaluate whether our approach can improve the proof capability of CoqHammer [12], the state-of-the-art symbolic prover in Rocq on these projects. Our results show that using the tactics mined by Deepseek-V3.2 [15], LLM2LTAC enhances the number of theorems proved by CoqHammer directly by 23.87%. When integrating the improved CoqHammer into the state-of-the-art proving agent, Claude Code [1], the overall number of theorems proved increases by 9.90%, and the token consumption is reduced by 10.51%.

**Contribution.** The contributions of this paper are as follows:

- We propose an approach that mines reusable proof strategies from existing formal proofs by an LLM, allowing the prover to apply this acquired knowledge at proof time without runtime LLM interaction and supporting deployment in constrained environments.
- We design a testing methodology that ensures mined tactics are generalizable to new proof goals outside the mining dataset.
- We implement and evaluate LLM2LTAC on multiple benchmarks, demonstrating that it enhances the automation capability of symbolic provers.

## 2 Background

**The Rocq Proof Assistant.** Rocq [39] (formerly known as Coq) is an interactive theorem prover based on the Calculus of Inductive Constructions (CIC) [31]. Users state theorems and construct machine-checked proofs interactively. At each step, the system maintains a *proof goal*: the proposition that remains to be proved. When a proof begins, the goal is the theorem statement itself. Users enter *tactics*, each of which transforms the current goal into zero or more subgoals. Once all subgoals are discharged, the Rocq kernel type-checks the proof term, guaranteeing correctness independently of the tactics that produced it. Rocq provides a rich set of built-in tactics for common proof steps, including introducing hypotheses (*intros*), applying known lemmas (*apply*), rewriting with equalities (*rewrite*), and performing structural induction (*induction*).

**Ltac.** Ltac [16] is the tactic scripting language built into Rocq. It allows users to compose built-in tactics into higher-level, reusable proof procedures. An Ltac definition binds a name to a tactic expression. The language supports pattern matching on the proof goal and hypotheses via the *match goal* construct, recursion, backtracking with the *+* and *||* operators, and looping via *repeat* and *do*. Ltac is widely used to encode domain-specific proof automation in Rocq projects. In this paper, LLM2LTAC produces Ltac tactics as its reasoning strategy representation, so that the mined proof strategies can be directly loaded and invoked by the Rocq system.

**CoqHammer.** CoqHammer [12] is an automated theorem proving tool for Rocq. It adapts the "hammer" methodology that has been effective in other proof assistants (e.g., Sledgehammer for Isabelle [8, 43]). The tool operates in three phases: Given a proof goal, CoqHammer performs *premise selection*, *translation* to first-order logic with external ATP invocation, and *proof reconstruction* inside Rocq to produce a fully checked proof term. CoqHammer is the state-of-the-art general-purpose symbolic prover for Rocq [12]. We use it as the baseline symbolic prover in our evaluation. Despite its strength, CoqHammer does not learn new proof strategies from existing proofs; it relies on the fixed set of ATPs and reconstruction tactics. LLM2LTAC addresses this limitation by mining reusable Ltac tactics from proof corpora and integrating them into CoqHammer's proof search.

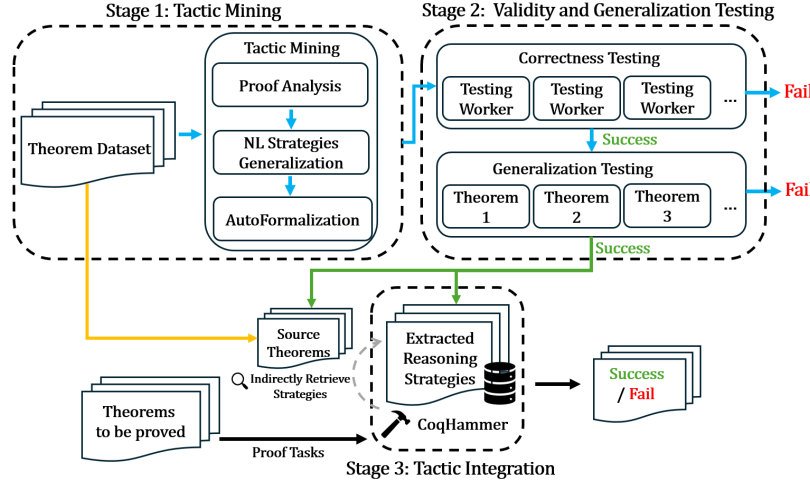


Figure 1: The workflow of LLM2LTAC

```

Definition divls (v1 v2: val): option val :=
  match v1, v2 with
  | Vlong n1, Vlong n2 =>
    if Int64.eq n2 Int64.zero
    || Int64.eq n1 (Int64.repr Int64.min_signed)
    && Int64.eq n2 Int64.mone
    then None
    else Some (Vlong (Int64.divs n1 n2))
  | _, _ => None
end.

```

```

Definition partial_binary_constructor_sound cstr sem :=
  forall le a x b y z,
  eval_expr ge sp e m le a x ->
  eval_expr ge sp e m le b y ->
  sem x y = Some z ->
  exists v, eval_expr ge sp e m le (cstr a b) v ^
  lessdef z v.

```

```

Theorem eval_divls_base:
  partial_binary_constructor_sound divls_base divls.

```

Figure 2: Theorem to be proved in benchmarks

### 3 Overview

In this section, we present an overview of LLM2LTAC, a framework designed to mine reasoning strategies. We will use some examples to demonstrate the workflow of our approach.

We consider the theorem `eval_divls_base` in Figure 2, which comes from the `CompCert` project. This theorem describes a fundamental property of instructions selection. Here, `partial_binary_constructor_sound` is a general correctness template for a binary function: `eval_expr` denotes the evaluation of an expression, `sem` denotes the abstract semantics of a binary operation over values, and `lessdef` denotes a refinement relation, under which  $z = v$

holds as long as  $z$  is not an undefined value. `divls_base` denotes 64-bit signed division on the target machine.

The theorem `eval_divls_base` states that for any two expressions  $a$  and  $b$  that evaluate to the 64-bit integer values  $x$  and  $y$ , as long as  $x$  divided by  $y$  is mathematically well-defined (i.e., `divls`  $x$   $y$  = `Some`  $z$ , with no division by zero and no overflow), the expression `divls_base a b` generated by the instruction selection phase evaluates to a result that agrees with  $z$ . This guarantees that the instruction selection step is semantics-preserving for 64-bit signed division.

The workflow of LLM2LTAC is shown in Figure 1. The entire framework is divided into three stages, where the first two steps are performed offline, similar to the training phase of machine learning, and the last step is performed online when proving new theorems.

- **Stage 1: Tactic mining.** In this stage, we use large language models to mine reasoning strategies from a set of theorems and their proofs.
- **Stage 2: Validity and generalization testing.** After mining the tactics, we need to ensure their validity and generalizability. We design specific tests to verify that the mined tactics can be applied in various scenarios.
- **Stage 3: Tactic integration.** We combine the mined tactics with symbolic provers to form a new prover with better theorem-proving capability. The prover uses these tactics to convert complex theorems into simpler goals until they can be automatically solved.

#### 3.1 Tactic Mining

*Proof analysis.* For each theorem and proof in the dataset, we ask the LLM to explain the purpose of each tactic in the proof, and what it means to use this tactic at the current position in natural language (NL). We analyze each tactic individually rather than asking the LLM to describe the proof as a whole, because the LLM tends to omit tactics whose effects are less apparent when summarizing an entire proof at once. Per-tactic analysis yields more detailed

descriptions that capture the specific effect of every proof step. For example, the following is a theorem from the dataset:

**Lemma** `hd_error_some_nil` `l` (`a` : `A`) :  
`hd_error l = Some a → l <> nil`.

**Proof.**

`unfold hd_error. destruct l; now discriminate.`

**Qed.**

where `A` is a polymorphic type that represents a list of elements.

The LLM provides the following natural language description for the effect of each tactic:

```
### Step-by-step explanation
1. `unfold hd_error`.
  **Meaning:** Replaces `hd_error` by its definition. ...
  **Purpose in this proof:** Unfolding turns the goal into a
  statement about a `match l with ... end`, making it possible to
  reason by cases on `l`.
  ...
2. `destruct l; now discriminate`.
  **Meaning:** Splits the goal into two cases based on the shape
  of `l`. ...
  **Purpose in this proof:** The property ``hd_error l = Some a`
  implies `l <> nil` is naturally proven by splitting into the
  two possible shapes of `l`.
  ...
```

We can observe that the LLM analyzed the roles of two tactics (`unfold` and `destruct`) in this proof and the impact on the goals when each tactic is applied. Then, by providing the proof process and the meaning of each proof step as input, we guide the LLM to give a general reasoning strategy.

*NL strategies generalization.* In the second step, we require the LLM to summarize the given proof analysis into generalizable strategies. After providing the above proof analysis to the LLM, the LLM gave the following general proof method:

```
General Proof Strategies
## 1) Unfold a definition to expose a match, then do case analysis
**Pattern:** `unfold f.` followed by `destruct x.` (or `cases x`)
  when `f x` reduces by pattern matching on `x`.
  ...
## 2) Split by constructors, then finish each branch with
  constructor-injectivity/disjointness (`discriminate`)
  ...
```

We can observe that the LLM provides general proof strategies. The first strategy unfolds a definition and then performs case analysis, which applies whenever a function reduces by pattern matching on its argument. The second strategy splits the goal by constructors and finishes each branch using constructor. We can observe that all strategies generalize beyond the current lemma and can be used in proving other theorems.

*Autoformalization.* We require the LLM to formalize the new general proof strategies described in natural language into the tactic language used by the symbolic system. Since the theorems we selected are obtained from Rocq projects, for consistency, we formalize the new proof strategies into tactics represented in Ltac in Rocq [16]. Note that the role of autoformalization here is solely to translate natural language strategies into a symbolic representation; the correctness and generalizability of the resulting tactics are not assumed but are instead verified by the subsequent testing stage. In this example, multiple tactics are formalized, and we show one tactic formalized from Strategy 1 as an example, whose formalized tactic is shown in Figure 3. The Ltac cannot know whether a

```
Ltac unfold_match_then_destruct :=
  match goal with
  | |- context [ ?f ?x ] =>
    unfold f; destruct x
  | H : context [ ?f ?x ] |- _ =>
    unfold f in H; destruct x
  end.
```

**Figure 3: One mined tactic from lemma `hd_error_some_nil` by LLM2LTAC**

function reduces by pattern matching on a given argument, so it uses syntactically matching a function application to represent the condition.

### 3.2 Validity and Generalization Testing

Due to LLM hallucinations, the generated code by LLM may not be valid, so we need to test whether it is formalized correctly in Rocq Ltac. In addition, LLM2LTAC also tests the generalization of tactics to ensure they can be used for proving other theorems. Note that a valid tactic is correct by construction in the Rocq system, and would never result in an incorrect proof.

To test validity, we employ a test-and-repair framework using the Rocq compiler. If compilation fails, we ask the LLM to repair the tactic. Tactics that fail after a fixed number of repairing attempts are discarded. To test generalizability, we use a validation set containing theorems that are not included in the mining dataset. We then apply the tactic into each position in the theorems' proving processes. If the proof statement changes after the application, we consider the tactic to have generalization capability and retain it; otherwise, we discard it. For the retained tactics, LLM2LTAC records the source theorem from which the tactic was mined, which will be used for retrieval when proving new theorems.

### 3.3 Tactic Integration

In this stage, LLM2LTAC combines the mined tactics with a symbolic prover. Given a proof goal that cannot be directly resolved by the symbolic prover, we retrieve relevant tactics. As mentioned before, tactics contain limited textual information and direct retrieval of tactics is ineffective. Our approach uses the theorems for mining tactics as the bridge for retrieval. We first use BM25 to identify the theorems that are most similar to the current proof goal from the dataset for tactic mining, and return the tactics mined from the theorems. Next, we apply each of the retrieved tactics to transform the current proof goal and then apply the symbolic prover again. We construct an And-Or tree [9] to try different combinations of the above process until the theorem is proved.

For the theorem in Figure 2, though CoqHammer cannot directly prove the theorem, the enhanced prover can retrieve the theorem `hd_error_some_nil` and use tactic `unfold_match_then_destruct` to convert the theorem into new goals that can be automatically solved by CoqHammer.

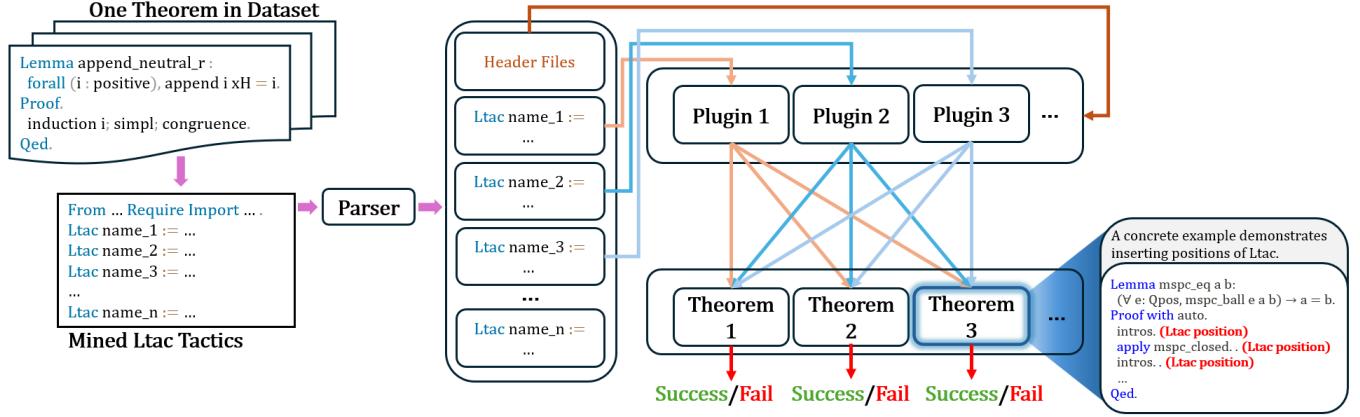


Figure 4: Workflow of generalization testing

## 4 Approach

In this section, we present the detailed design of LLM2LTAC. As illustrated in Figure 1, the workflow of LLM2LTAC consists of three main stages: i) **Tactic mining**. In this stage, LLM2LTAC mines potential tactics from LLMs. ii) **Validity and generalization testing**. The testing checks the tactics that are syntactically valid and applicable to other theorems. iii) **Tactic integration**. LLM2LTAC incorporates the mined tactics into the symbolic prover, employing a new retrieval method to select appropriate tactics based on the proof goal.

### 4.1 Tactic Mining

By inputting a dataset of theorems, we aim to mine the reasoning patterns that are already formed within proofs. This mining process consists of three steps: proof analysis, natural language strategy generalization, and autoformalization.

**Proof Analysis.** In the proof analysis step, we prompt LLMs to analyse the proof steps of given theorems. We require the LLM to perform two tasks. The prompts are shown in Figure 5. Specifically, the LLM performs two tasks per proof step: (1) explaining the semantic meaning of each tactic (e.g., what rewrite or induction does), and (2) describing the effect of each tactic on the current proof goal. To support accurate analysis, we supply the LLM with the types and definitions of inductive data referenced in the tactic parameters.

**Natural language strategies generalization.** In this step, we prompt the LLM to identify which steps from the previous proof analysis can be abstracted to a higher level. The prompt is shown in Figure 6. We use natural language descriptions for proof analysis and generalization, as LLMs are more powerful in understanding and generating natural language compared to symbolic language [42].

**Autoformalization.** In the autoformalization step [44], we use LLMs to convert the natural language descriptions of reasoning strategies into the formal language. The prompts used in the LLM are shown in Figure 7. During autoformalization, the LLM translates the high-level strategic descriptions into Ltac [16], the tactic language of Rocq. We provide the LLM with Ltac documentation

and examples to facilitate accurate translation, and instruct it to generate parameter-free tactics that match patterns in the proof goal.

### 4.2 Validity and Generalization Testing

**Validity Testing.** The purpose of this stage is to test the tactics mined from the LLM. In validity testing, we design a test-and-repair framework to test whether the mined tactics are valid. We use the symbolic verifier to compile each tactic. If the verifier finishes compilation without errors, we consider the tactic to be valid within the logical system. If the verifier reports errors, we provide both the error message and the current tactic back to the LLM, prompting it to repair the errors. If the number of repair attempts exceeds a threshold, we discard this tactic.

**Generalization Testing.** This stage evaluates whether the tactics can be applied to theorems in other projects. We collect a large number of theorems from other projects as the benchmark. Through generalization testing, we select the tactics whose generalizability falls within a certain range for the final symbolic prover enhancement. The workflow is shown in Figure 4. The overall workflow proceeds as follows: first, since a single theorem may yield multiple tactics, we employ a parser to decompose the tactic into individual tactic definitions along with their required header files and type definitions. This decomposition step is crucial as it isolates each tactic as an independent unit while preserving all necessary dependencies for compilation. Then, for each mined tactic, LLM2LTAC automatically generates a Rocq plugin that wraps the tactic with all its required dependencies and exposes it through a named interface. This plugin-based method allows each proof script to load a tactic via a single command, avoiding repeated script modifications and dependency conflicts across different tactics.

For generalization testing, we insert each mined tactic after every original tactic in the benchmark proof scripts (except for the last tactic that closes the goal), and check whether it successfully produces a new proof state. As shown in Figure 4 theorem 3, a concrete example demonstrates the inserting positions of the tactics (marked by "Ltac position"). A tactic is considered to be generalizable if, given the current proof state and theorem, it produces new

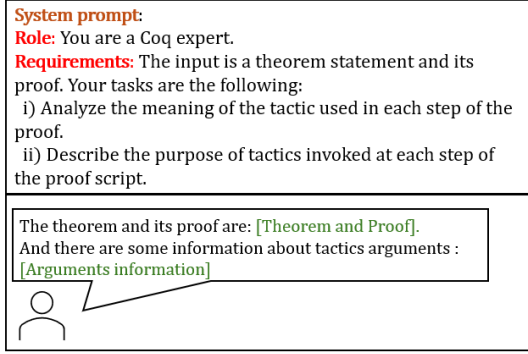


Figure 5: Prompts in proof analysis

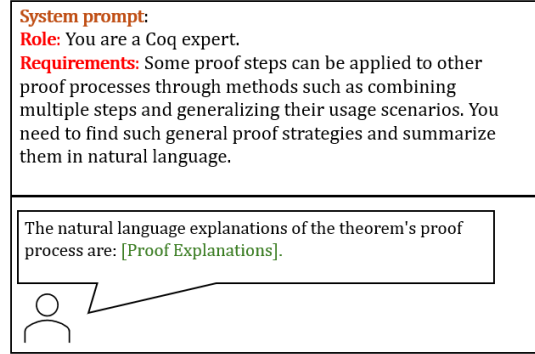


Figure 6: Prompts in natural language generalization

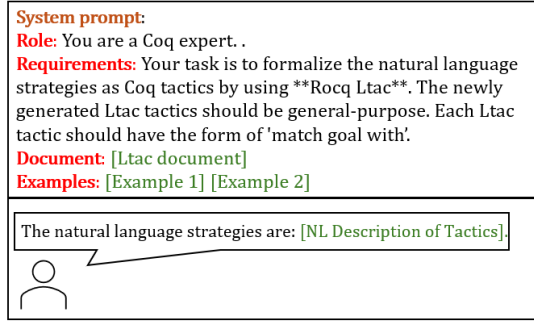


Figure 7: Prompts in Autoformalization

proof states without reporting errors (a tactic without errors but makes no changes to the proof state is discarded). We record the number of successful applications across all benchmark positions as a metric of generalization capability. In our approach, there are 93,424 test positions in total; if a tactic cannot make any change to the proof state across so many positions, we tend to regard it as lacking generalization capability. Specifically, we consider a tactic to have generalization capability if its application success rate across all benchmark positions falls within [10%, 100%] (details in Section 5.6, where we show that tactics from different intervals all improve the prover). A tactic with a success rate below 10% is considered too narrow to use reliably. Conversely, a tactic that succeeds in 100% of positions is considered over-general: such a tactic typically only introduces a fresh variable to alias an existing term in the proof context, which changes the proof state without making meaningful reasoning progress.

### 4.3 Tactic Integration

Tactics that pass the generalization testing are combined with a symbolic prover for better theorem proving. Since existing symbolic provers do not support the utilization of the tactics presented in the code, we design an algorithm to alternately invoke the symbolic prover and apply tactics, shown as algorithm SOLVE in Algorithm 1. The algorithm models proof search by constructing an And-Or

tree [9]. Each proof goal forms an OR-node: it is solved if *any* tactic succeeds. Each tactic application forms an AND-node: it requires *all* resulting subgoals to be proved.

---

#### Algorithm 1 And-Or Tree Proof Search with Tactic Retrieval

---

```

1: function SOLVE(goal)
2:   root ← ORNODE(goal)
3:   queue ← QUEUE(); queue.ENQUEUE(root)
4:   while queue is not empty ∧ ¬ root.solved do
5:     n ← queue.DEQUEUE()
6:     if APPLYATP(n.goal) then
7:       PROPAGATE(n); continue
8:     Thms ← RETRIEVETHEOREMS(n.goal, k)
9:     T ← GETTACTICS(Thms)
10:    for each t ∈ T do
11:      subgoals ← APPLY(t, n.goal)
12:      a ← ANDNODE(); n.ADDCHILD(a)
13:      if subgoals = ∅ then
14:        PROPAGATE(a)
15:      else
16:        for each g' ∈ subgoals do
17:          c ← ORNODE(g'); a.ADDCHILD(c)
18:          queue.ENQUEUE(c)
19:  return root.solved

```

---

Table 1: Impact of  $k$  value on Algorithm 1

| $k$ value | Proved | Time Consumption |
|-----------|--------|------------------|
| 10        | 77     | 2212.89s         |
| 20        | 85     | 3571.96s         |
| 30        | 87     | 4103.70s         |

Given a proof goal, SOLVE first attempts to close it with the automated theorem prover. If that fails, it retrieves the top- $k$  most similar theorems using RETRIEVETHEOREMS. Since Ltac is a structured representation, term-frequency algorithms such as BM25 [34] cannot be directly applied. To address this, LLM2LTAC records the

source theorem from which each tactic was mined during generalization testing. In our algorithm, the current proof goal is compared against these source theorems using a retrieval algorithm. Then, it locates the tactics mined from these theorems using the GETTACTICS function. Each tactic application creates an AND-node whose children are new OR-nodes for the resulting subgoals, which are enqueued for subsequent processing. When a node is solved, PROPAGATE walks upward through the tree: an OR-node is marked solved when any child succeeds; an AND-node is marked solved only when all its children are solved.

The parameter  $k$  controls how many similar theorems are retrieved at each node. A larger  $k$  increases the chance of finding a useful tactic but also expands the search space and increases time cost. We randomly select 50 samples from each benchmark (mentioned in Section 1), for a total of 200 samples. We set  $k$  to 10, 20, and 30, respectively, and record the number of proved theorems and the time consumed for each setting, and the result is shown in Table 1. We evaluate this trade-off and set  $k = 20$  for our approach.

## 5 Evaluation

We focus on the following research questions:

- **RQ1:** How effective is LLM2LTAC in mining tactics?
- **RQ2:** To what extent do the tactics mined from LLMs by LLM2LTAC improve the success rate of symbolic provers?
- **RQ3:** Do the mined tactics benefit LLM agents?
- **RQ4:** How does LLM2LTAC perform compared to existing extraction methods?
- **RQ5:** How do mining methods affect the mined tactics?
- **RQ6:** Do tactics with different generalization abilities impact the improvement of symbolic provers?

**Implementation.** We have implemented LLM2LTAC on Rocq 8.20.0. To obtain the argument information in Section 4.1, we modified parts of the Rocq compiler to extract argument types and type definitions. We selected CoqHammer, the state-of-the-art automated theorem prover in Rocq [12], as the symbolic prover for evaluating performance improvements.

**Dataset.** We selected different benchmarks from Rocq open-source projects for different stages in LLM2LTAC. We collected 11,725 theorems from the Rocq standard library for mining tactics. We collected 6,462 theorems from the C-CoRN [11] dataset, which contains theorems on common algebraic structures (e.g., setoids, monoids, groups) and large proof procedures for complex properties, for testing generalization. For each candidate tactic, we insert it at all 93,424 different positions across these 6,462 theorems' proof processes to assess how broadly the tactic applies under different proof contexts. To evaluate the improvement to symbolic provers, we selected the following four benchmarks to evaluate different verification tasks:

- **COQ-ART** [5]: a Rocq textbook covering well-established properties of data structures and algorithms.
- **EXT-LIB** [28]: a community-driven Rocq library that is widely reused across diverse Rocq projects.
- **VFA** [3]: the third volume of Software Foundations covering well-established properties of verified functional algorithms.
- **COMP CERT** [24]: an industrial-scale formally verified C compiler with machine-checked proofs of semantic preservation

across multiple compilation passes, memory models, and target architectures.

Together, they span varying levels of proof complexity and application domains, making them a rigorous testbed for evaluating the generalization of LLM2LTAC.

**Configurations.** We used OpenAI GPT-5.2 [29] and Deepseek-V3.2 [15] for evaluation experiments. For the tactic repair process in Section 4.2, we set the threshold to 3. The total API cost for tactic mining was \$1,528.65, of which Deepseek-V3.2 accounted for \$108.93 and GPT-5.2 for \$1,419.72, well below the cost of GPU infrastructure to host comparable models locally. We set CoqHammer's timeout to default value, which is 20 seconds, and for each theorem to be proved, we set the timeout to 600 seconds. We select mined tactics whose application success rate falls within [10%, 100%) (at least 10% but less than 100%) in generalization testing. For the retrieval algorithm, since our approach targets resource-constrained settings, embedding-based models that require high-performance GPUs are not applicable for tactic retrieval. We therefore choose BM25 to index tactics, which runs efficiently on CPUs [34]. Prior work has also shown that BM25 outperforms embedding-based models for retrieval in theorem proving [40]. All experiments were conducted on a server equipped with four 26-core Intel Xeon Platinum 8270 CPUs, 256 GB RAM, and running Ubuntu 24.04.4.

### 5.1 RQ1: How effective is LLM2LTAC in mining tactics?

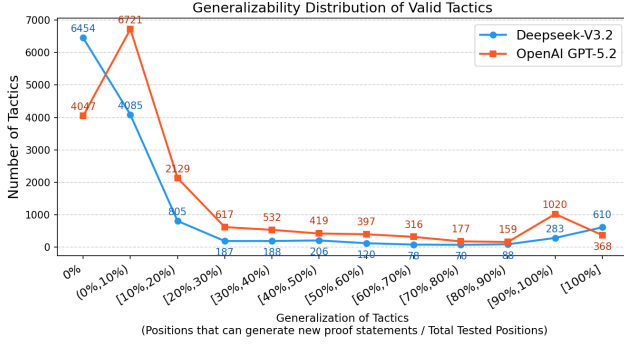
**Procedure.** This experiment evaluates how effectively LLM2LTAC mines tactics. We apply LLM2LTAC to mine tactics from 11,725 theorems in the Rocq standard library. For validity testing, we record the number of tactics obtained by OpenAI GPT-5.2 and Deepseek-V3.2 at each stage of the LLM2LTAC. For generalization testing, we count the generalizability distribution of valid tactics, and classify all valid tactics into groups based on their success rate with an interval of 10%, as shown in Figure 8.

**Table 2: Detailed Statistics for Tactic Mining**

|                          | OpenAI GPT-5.2 | Deepseek-V3.2  |
|--------------------------|----------------|----------------|
| Mined Tactics            | 49909          | 35408          |
| Valid Tactics            | 16902 (33.87%) | 13174 (37.21%) |
| Gen. Tactics ([10%,100)) | 5766 (11.55%)  | 2025 (5.72%)   |

**Results.** As shown in Table 2, “mined tactics” represents the number of natural language tactics obtained from LLMs, “valid tactics” represents the number of tactics that pass the Rocq compiler, and “Gen. Tactics” represents the number of tactics whose generalization testing success rate falls within [10%, 100%).

We observe that OpenAI GPT-5.2 mines more tactics (49,909) compared to Deepseek-V3.2 (35,408). (Valid tactics and Generalizable tactics counted here are syntactically distinct; tactics with different Ltac syntax may still be semantically equivalent, and the same tactic can behave differently across proof contexts.) After validity testing, the pass rates are comparable: 33.87% for OpenAI GPT-5.2 and 37.21% for Deepseek-V3.2. The autoformalization failures fall into three main categories: unresolved names, imports,



**Figure 8: Distribution of Tactic Generalizability**

libraries, or modules (about 70%); syntax, parsing, and lexical errors (about 23%); and notation or scope interpretation errors (about 5%). We merged and deduplicated the Valid Tactics and Generalizable Tactics ([10%, 100%]), and there are 25,416 unique valid tactics and 6,484 unique generalizable tactics. For generalization testing, we observe that LLMs still exhibit severe hallucinations in tactic mining: most generated tactics are incorrect and lack generalization capability, even though LLMs believe they are correct and generalizable. We further observe notable differences in the generalization distributions across models, and the majority of mined tactics fall below the 10% success rate. This also indicates the necessity of the testing process in LLM2LTAC.

*Example.* To illustrate the diversity of mined tactics, we present a representative example beyond the induction-based strategy shown in Figure 3. The following tactic shows a tactic mined by LLM2LTAC that operates in a fundamentally different way.

```
Ltac apply_constructor_auto_217 :=
  match goal with
  | [ |- ?R ?A ?B ] =>
    try (apply permA_swap; auto);
    try (intuition; auto)
  end.
```

This tactic first pattern-matches the current proof goal against a relational structure  $?R \ ?A \ ?B$ , then attempts to apply a lemma `permA_swap` and invokes built-in Rocq tactics `auto` and `intuition` to discharge remaining subgoals. Unlike the induction-based tactic in Figure 3, this example shows that LLM2LTAC can mine strategies that reason about the goal’s structure, apply project-specific lemmas, and compose multiple existing tactics in Rocq.

**Answer to RQ1:** LLM2LTAC is able to mine generalizable tactics from LLMs, and the testing process is necessary to filter out invalid or non-generalizable tactics.

## 5.2 RQ2: To what extent do the tactics mined from LLMs by LLM2LTAC improve the success rate of symbolic provers?

*Procedure.* We use `CompCert`, `EXT-LIB`, `Coq-ART`, and `VFA` four benchmarks, 6199 theorems in total, to test the automatic capability

of the prover. We compare the LLM2LTAC-enhanced CoqHammer against the original CoqHammer. For each experiment, we use the default configuration of CoqHammer and the timeout is set to 20 seconds.

*Results.* As shown in Table 3, integrating the tactics mined by LLM2LTAC significantly improves the performance of CoqHammer across all four benchmarks. Using tactics mined by OpenAI GPT-5.2, CoqHammer proves 1950 theorems, compared to 1592 theorems by the original CoqHammer, representing a 22.49% overall improvement. The improvements on individual benchmarks range from 17.27% on `Coq-ART` to 26.58% on `EXT-LIB`. Using tactics mined by Deepseek-V3.2, CoqHammer proves 1972 theorems, achieving 23.87% overall improvement. For the complete evaluation on all 6,199 theorems, proving with tactics extracted by OpenAI GPT-5.2 and Deepseek-V3.2 took approximately 98 hours and 178 hours, respectively. The corresponding fractions of theorems that hit the 600s timeout were 7.99% and 15.66%, respectively. The time gap arises because the tactics mined by Deepseek-V3.2 are more likely to produce subgoals, which deepens the search, whereas the tactics mined by GPT-5.2 often fail to yield effective subgoals, so the search algorithm does not go deeper.

While both models demonstrate substantial improvements, the performance of tactics mined from different LLMs is different. This difference indicates that the reasoning strategies mined from different LLMs have an impact on the performance of LLM2LTAC. Notably, as shown in Table 2, GPT-5.2 produces nearly three times more generalizable tactics than Deepseek-V3.2 (5,766 vs. 2,025), yet its proving count is slightly lower (1,950 vs. 1,972). This suggests that a larger number of tactics does not necessarily lead to better proving performance. A larger tactic pool introduces more noise during retrieval, causing the prover to spend time on less effective tactics before finding the right one.

The mined tactics contribute to 35.04% of the successfully proved theorems, and a total of 691 distinct tactics are applied across these theorems. These tactics can be grouped into six categories: definition unfolding and normalization (34.01%), structural induction followed by automation (28.08%), context-aware lemma application and rewriting (11.43%), decomposition by structure/constructors (10.85%), computation-based goal closing (1.74%), and other (13.89%).

We observe the extracted tactics often use similar patterns to human-written tactics. The `inv_all` tactic is extracted from Rocq standard library, which matches a function application with different numbers of parameters. Since Rocq does not directly support this, a set of patterns with different numbers of parameters is used. Tactic `decompose_algebraic_structure` is a tactic extracted by our approach, which uses a similar form to match a function application with different numbers of parameters. This shows that the new tactics extracted by our approach are already very similar to those manually written by humans, exhibiting good readability and writing style.

(\* Tactic written by human in Coq standard library \*)

```
Ltac inv_all f :=
  match goal with
  | H: f _ |- _ => inversion_clear H; inv f
  | H: f _ _ |- _ => inversion_clear H; inv f
```

**Table 3: Performance of CoqHammer with and Tactics Mined By LLM2LTAC**

|           | #Lemma | OpenAI GPT-5.2 |                        | Deepseek-V3.2 |                        |
|-----------|--------|----------------|------------------------|---------------|------------------------|
|           |        | CoqHammer      | CoqHammer +LLM2LTAC    | CoqHammer     | CoqHammer +LLM2LTAC    |
| COMP CERT | 4840   | 1022           | <b>1269</b> (↑ 24.17%) | 1022          | <b>1276</b> (↑ 24.85%) |
| EXT-LIB   | 187    | 79             | <b>100</b> (↑ 26.58%)  | 79            | <b>99</b> (↑ 25.32%)   |
| COQ-ART   | 842    | 411            | <b>482</b> (↑ 17.27%)  | 411           | <b>498</b> (↑ 21.17%)  |
| VFA       | 330    | 80             | <b>99</b> (↑ 23.75%)   | 80            | <b>99</b> (↑ 23.75%)   |
| Total     | 6199   | 1592           | <b>1950</b> (↑ 22.49%) | 1592          | <b>1972</b> (↑ 23.87%) |

```

| H: f _ _ _ |- _ => inversion_clear H; inv f
| H: f _ _ _ _ |- _ => inversion_clear H; inv f
| _ => idtac
end.
(* Extracted tactics by our approach *)
Ltac decompose_algebraic_structure :=
  match goal with
  | |- ?T _ _ _ _ _ => constructor
  | |- ?T _ _ _ _ _ => constructor
  | |- ?T _ _ _ _ _ => constructor
  | |- ?T _ _ _ _ _ => constructor
  | |- ?T _ _ _ _ _ => constructor
  | |- ?T _ _ _ _ _ => constructor
  | |- ?T _ _ _ _ _ => constructor
  end.

```

**Answer to RQ2:** The tactics mined by LLM2LTAC significantly enhance the capability of CoqHammer by 17.27% to 26.58%, and the improvements are related to the reasoning capability of the underlying LLMs, suggesting further improvements as LLMs advance.

### 5.3 RQ3: Do the mined tactics benefit LLM agents?

*Procedure.* Due to the high cost of LLM-based agents, we sample 50 theorems from the test set of each of the same four benchmarks used in RQ2, 200 theorems in total, to evaluate whether LLM2LTAC can benefit the LLM-based agent. We select Claude Code [1], the state-of-the-art LLM-based agent, powered by Claude Sonnet 4.6 [2] as the base model. [30] shows that Claude Code also performs well on Rocq theorem proving.

We write three categories of skills to instruct Claude Code to write theorem proofs. The first category covers the base proving pipeline, guiding the agent to locate the target theorem within a project and verify whether a constructed proof is correct. This category is common to all three settings in the following evaluation. The second category is CoqHammer-specific: it instructs the agent on how to import the required header files into the proof script and which tactics to apply. The third category is LLM2LTAC-specific: it explains to the agent how to use the LLM2LTAC-enhanced prover. Both categories direct the agent to first attempt to prove the theorem with the respective tool, and only if that attempt fails, to synthesize a proof on its own. For each task, we grant Claude Code three tool permissions: `bash`, `read`, and `edit`, allowing the agent to execute

shell commands, read files, and edit files. We set a timeout of 600 seconds for each proof task.

We evaluate three settings: i) Claude Code proves theorems without access to CoqHammer; ii) Claude Code proves theorems with access to CoqHammer via the corresponding skills; iii) Claude Code proves theorems with the LLM2LTAC-specific skills. For each setting, we record the number of theorems successfully proved and the total token consumption of Claude Code.

**Table 4: Performance of the LLM Agents**

|        | Claude Code | Claude Code + CoqHammer | Claude Code + LLM2LTAC |
|--------|-------------|-------------------------|------------------------|
| #Thm   | 71          | 101                     | 111 (↑ 9.90%)          |
| Tokens | 59,899,928  | 38,346,691              | 34,317,809 (↓ 10.51%)  |
| Cost   | \$67.21     | \$44.30                 | \$39.77 (↓ 10.23%)     |

*Results.* As reported in Table 4, the LLM2LTAC-enhanced agent proved 111 out of 200 randomly sampled theorems, compared to 101 by the CoqHammer-enhanced agent (a 9.90% improvement). This result demonstrates that LLM2LTAC is more effective than CoqHammer when integrated into a LLM-based proving agent. Meanwhile, the token consumption decreased from 38,346,691 to 34,317,809, a 10.51% reduction. Accordingly, the monetary cost decreased from \$44.30 to \$39.77, a 10.23% reduction. This demonstrates that LLM2LTAC not only enhances the proving capability of LLM agents but also reduces their token cost.

**Answer to RQ3:** The mined tactics from LLM2LTAC can benefit LLM agents by improving their theorem-proving success rate and reducing token consumption.

### 5.4 RQ4: How does LLM2LTAC perform compared to existing extraction methods?

*Procedure.* As will be discussed in Section 6, a recently posted work, STRAT2ROCQ [17], mines lemmas from the LLM reasoning process to enhance symbolic provers.

For STRAT2ROCQ, we conduct two evaluations. Setting I: we mine lemmas from the RQ1 dataset and test on the RQ2 benchmarks. Setting II: we use 90% of the theorems in RQ1 for learning and the remaining 10% for testing. These two settings respectively simulate the scenarios where the projects are less closely related and more closely related. In both settings, the extracted lemmas are

integrated into CoqHammer to compare with our approach. We also compare LLM2LTAC against TacMiner [45], the state-of-the-art tactic mining approach. To compare our approach with TacMiner, we implemented TacMiner to extract tactics over the same dataset used in RQ1. We then integrate the extracted tactics into CoqHammer and evaluate the performance improvement on the same four benchmarks used in RQ2.

**Results.** The results are shown in Table 5. By reproducing STRAT2ROCQ in Setting I, we extracted 2,977 new lemmas and integrated them into CoqHammer’s context. In our experiments, we observe that, under the current experimental setup, STRAT2ROCQ does not effectively enhance the symbolic prover. This discrepancy from the existing evaluation results reported in the STRAT2ROCQ paper may be attributed to the differences in dataset selection. In their paper, the dataset for mining lemmas and the dataset for testing are more related, and may come from the same project, while in our evaluation the two sets are from strictly different projects, better matching real-world scenarios. This result suggests that the tactics mined by our approach are more generalizable across different projects compared to the lemmas mined by STRAT2ROCQ. In the setting II of STRAT2ROCQ, where we use 90% of the theorems in RQ1 for learning and the remaining 10% for testing, we mined 1367 new lemmas and evaluate on 966 theorems. CoqHammer proved 521 theorems, CoqHammer with STRAT2ROCQ proved 546, and CoqHammer with LLM2LTAC proved 722. These results show that even across closely related projects, our approach still achieves better performance.

As for TacMiner, we mined 1,217 usable tactics in total. TacMiner extracts less than 10% as many tactics as our approach, and these tactics provide only a marginal improvement of 1.76% compared to our approach’s 23.87% improvement. This indicates that, compared with the state-of-the-art approach, the tactics mined by our approach are more broadly applicable.

#### Answer to RQ4:

Compared with existing state-of-the-art extraction methods, LLM2LTAC is better than them in both applicability and capability.

### 5.5 RQ5: How do mining methods affect the mined tactics?

**Procedure.** To evaluate the contribution of different components in the mining process, we conduct an ablation study using Deepseek-V3.2. Specifically, we compare three configurations: (1) the original LLM2LTAC with all components, (2) LLM2LTAC without the natural language analysis step (the first step in Section 4.1), and (3) LLM2LTAC without the natural language generalization step (the second step in Section 4.1). We apply these configurations to mine tactics from the same dataset and measure the number of mined tactics, valid tactics, and generalizable tactics.

**Results.** As shown in Table 6, the ablation study reveals the importance of both natural language analysis and generalization steps in LLM2LTAC. Removing the natural language analysis step leads to a 34.31% reduction in mined tactics, with more drops in valid tactics (51.64% decrease) and generalizable tactics (71.56% decrease)

test pass rates. This indicates that the analysis step is helpful for guiding LLMs to generate higher-quality tactic candidates.

Without the natural language generalization step, valid tactics drop by 66.49%. This indicates that directly formalizing proof analysis is more challenging for LLMs than describing tactics in natural language. Generalizable tactics decreased by 88.59%. This demonstrates that the tactic generalization step plays a crucial role in producing the final usable generalizable tactics.

**Answer to RQ5:** Both the proof analysis and tactic generalization steps are essential for LLM2LTAC to mine tactics from LLMs.

### 5.6 RQ6: Do tactics with different generalization abilities impact the improvement of symbolic provers?

**Procedure.** Since there are infinitely many possible ways to choose the interval, we divide the overall generalizability into small intervals and evaluate the impact of each interval on symbolic proving. Specifically, we divide the generalization passing rate into intervals of 10%, and evaluate on the same four benchmarks used in RQ2, counting the number of theorems proved within each interval.

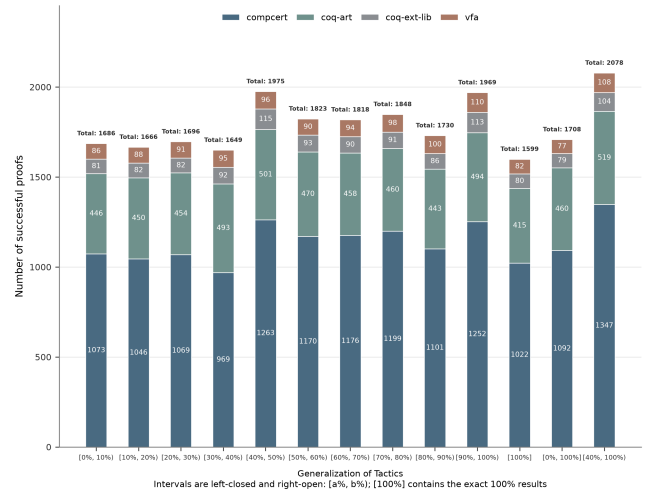


Figure 9: The statistics of the generalization intervals

**Results.** As shown in Figure 9, directly using all mined tactics without generalization testing (the 0%–100% configuration) proves substantially fewer theorems than our current filtered configuration ([10%, 100%]), which confirms that the generalization filtering is indeed beneficial. Moreover, both the intervals with generalizability no larger than 40% and the [100%, 100%] interval yield relatively limited improvements, which shows that discarding tactics with low generalizability as well as fully general (100%) tactics is a feasible design choice. Finally, further restricting the range to [40%, 100%) significantly outperforms our current result, indicating that, by selecting appropriate interval parameters, the effectiveness of LLM2LTAC can be further improved significantly.

**Table 5: Comparison with the state-of-the-art extraction approaches STRAT2ROCQ and TacMiner**

|           | CoqHammer | CoqHammer<br>+STRAT2ROCQ | CoqHammer<br>+TacMiner | CoqHammer<br>+LLM2LTAC |
|-----------|-----------|--------------------------|------------------------|------------------------|
| COMP CERT | 1022      | 1072                     | 1031                   | 1276                   |
| EXT-LIB   | 79        | 73                       | 79                     | 99                     |
| COQ-ART   | 411       | 379                      | 427                    | 498                    |
| VFA       | 80        | 71                       | 83                     | 99                     |
| Total     | 1592      | 1595                     | 1620 (↑ 1.76%)         | 1972 (↑ 23.87%)        |

**Table 6: The ablation study of the mining methods**

|                   | Original<br>Tactics | -NL<br>Analysis |         | -NL<br>Generalization |         |
|-------------------|---------------------|-----------------|---------|-----------------------|---------|
|                   |                     | #               | Change  | #                     | Change  |
| Mined             | 35408               | 23256           | ↓34.31% | 38003                 | ↑7.33%  |
| Valid             | 13174               | 6371            | ↓51.64% | 4415                  | ↓66.49% |
| Gen. ([10%,100%]) | 2025                | 576             | ↓71.56% | 231                   | ↓88.59% |

**Answer to RQ6:** Tactics with different generalization abilities have different effects on the proving performance of LLM2LTAC.

## 6 Related Work

Recent work uses LLMs to extract structured knowledge such as logic rules [27], mathematical equations [36], knowledge graphs [47], and lemmas [17]. These methods target specific rule types and data sources, and none of them mine reusable tactics from formal proofs. LLM2LTAC fills this gap by mining reasoning strategies from existing proofs and using the mined tactics to enhance symbolic provers.

The most related work is STRAT2ROCQ [17], which also enhances automatic theorem proving but extracts lemmas rather than tactics. LLM2LTAC differs in three ways: (1) we mine reusable tactics from existing formal proofs (written by humans or LLMs), rather than extracting strategies solely from LLM internals; (2) tactics are more generally applicable than lemmas, which only match proof goals of specific types; and (3) we introduce an indirect retrieval method that addresses the lack of textual information in Ltac tactics. Our evaluation shows that LLM2LTAC significantly outperforms STRAT2ROCQ.

Different from our approach, both TacMiner [45] and Peano [32] apply pattern mining to discover common patterns in existing proofs, and the mined new tactics are merely sequences of existing tactics. Our approach is able to infer the application condition of tactics (the “match goal” part in Ltac) with the help of LLMs. Furthermore, in our approach, generalization testing ensures that the extracted tactics have generalization capability, and we further recombine existing proof strategies into new proof strategies and attach application conditions to existing tactics. Finally, previous approaches extract tactics to enhance the capability of AI models, whereas our approach is the first to apply the extracted strategies to symbolic provers.

*Machine learning-based methods.* Many machine learning methods have been applied to theorem proving [6, 7, 18, 20, 21, 26, 35, 40, 41, 46]. These approaches can be divided into two categories: i)

training new models for theorem proving through large amounts of corpus, and ii) implementing agents to interact with LLMs to generate proof scripts. In other way, a non-trivial neural model is involved in the proving process. On the other hand, LLM2LTAC focuses on enhancing symbolic provers, and as our evaluation shows, the enhancement complements neural models, and can further improve the performance of proving agents as well as reducing their costs.

*Symbolic provers.* Many existing symbolic provers are proposed to reduce the burden of manually writing proof scripts, such as CoqHammer [12], Lean-auto [33], and Sledgehammer [8]. These provers typically focus on integrating existing automated theorem provers (ATPs) such as SMT solver to prove theorems in the target proof language, and none of them attempt to mine new tactics from existing proofs as far as we are aware.

## 7 Conclusion and Future Work

In this paper, we studied how to enhance provers to mine generalizable tactics from large language models. We proposed LLM2LTAC, the first framework for mining tactics from LLMs with support for generalization validation. Our evaluation experiments demonstrate that by using LLM2LTAC, the success rate of CoqHammer improved by 23.87%.

Currently, because automated theorem provers in interactive theorem assistants do not support user-defined tactics, the mined tactics can only be used externally to the symbolic prover. But this way cannot be deeply integrated into the automated proving processes of the symbolic solvers. How to integrate mined tactics into the automated proving processes remains as future work.

## 8 Acknowledgments

This work is supported by the National Cyber Security-National Science and Technology Major Project (No. 2025ZD1503300) and National Natural Science Foundation of China (No. 92582202).

## 9 Data Availability

The implementation of LLM2LTAC and all data of this paper are available at <https://doi.org/10.5281/zenodo.21407479>.

## References

- [1] Anthropic. 2025. Claude Code: An agentic coding tool. <https://code.claude.com/>.
- [2] Anthropic. 2026. Introducing Claude Sonnet 4.6. <https://www.anthropic.com/news/claude-sonnet-4-6>.
- [3] Andrew W. Appel. 2024. *Verified Functional Algorithms*. Software Foundations, Vol. 3. Electronic textbook. <https://softwarefoundations.cis.upenn.edu/vfa-current/index.html>

- [4] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13243)*, Dana Fisman and Grigore Rosu (Eds.). Springer, 415–442. doi:10.1007/978-3-030-99524-9\_24
- [5] Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Springer. doi:10.1007/978-3-662-07964-5
- [6] Lasse Blaauwbroek, Mirek Olsák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. 2024. Graph2Tac: Online Representation Learning of Formal Math Concepts. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. <https://openreview.net/forum?id=A7CtiozznN>
- [7] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. 2020. Tactic Learning and Proving for the Coq Proof Assistant. In *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020 (EPIC Series in Computing, Vol. 73)*, Elvira Albert and Laura Kovács (Eds.). EasyChair, 138–150. doi:10.29007/WG1Q
- [8] Sascha Böhme and Tobias Nipkow. 2010. Sledgehammer: Judgement Day. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6173)*, Jürgen Giesl and Reiner Hähnle (Eds.). Springer, 107–121. doi:10.1007/978-3-642-14203-1\_9
- [9] Brigitte Chauvin, Philippe Flajolet, Danièle Gardy, and Bernhard Gittenberger. 2004. And/Or Trees Revisited. *Comb. Probab. Comput.* 13, 4-5 (2004), 475–497. doi:10.1017/S0963548304006273
- [10] Yizhou Chen, Zeyu Sun, Guoqing Wang, and Dan Hao. 2025. Gpass: A Goal-Adaptive Neural Theorem Prover Based on Coq for Automated Formal Verification. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 29–41. doi:10.1109/ICSE55347.2025.00116
- [11] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. 2004. C-CoRN, the Constructive Coq Repository at Nijmegen. In *Mathematical Knowledge Management, Third International Conference, MKM 2004, Białowieża, Poland, September 19-21, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3119)*, Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec (Eds.). Springer, 88–103. doi:10.1007/978-3-540-27818-4\_7
- [12] Łukasz Czajka and Cezary Kaliszyk. 2018. Hammer for Coq: Automation for Dependent Type Theory. *J. Autom. Reason.* 61, 1-4 (2018), 423–453. doi:10.1007/S10817-018-9458-4
- [13] Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 625–635. doi:10.1007/978-3-030-79876-5\_37
- [14] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. doi:10.1007/978-3-540-78800-3\_24
- [15] DeepSeek-AI. 2025. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. arXiv:2512.02556 [cs.CL] <https://arxiv.org/abs/2512.02556>
- [16] David Delahaye. 2000. A Tactic Language for the System Coq. In *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1955)*, Michel Parigot and Andrei Voronkov (Eds.). Springer, 85–95. doi:10.1007/3-540-44404-1\_7
- [17] Jian Fang, Yican Sun, and Yingfei Xiong. 2025. Proof Strategy Extraction from LLMs for Enhancing Symbolic Provers. arXiv:2510.10131 [cs.LO] <https://arxiv.org/abs/2510.10131>
- [18] Emily First, Yuriy Brun, and Arjun Guha. 2020. TacTok: semantics-aware proof synthesis. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 231:1–231:31. doi:10.1145/3428299
- [19] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. 2016. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, Kimberly Keeton and Timothy Roscoe (Eds.). USENIX Association, 653–669. <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gu>
- [20] Albert Q. Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. 2022. Thor: Wielding Hammers to Integrate Language Models and Automated Theorem Provers. *CoRR* abs/2205.10893 (2022). doi:10.48550/arXiv.2205.10893
- [21] Saketh Ram Kasibatla, Arpan Agarwal, Yuriy Brun, Sorin Lerner, Talia Ringer, and Emily First. 2024. Cobblestone: Iterative Automation for Formal Verification. *CoRR* abs/2410.19940 (2024). arXiv:2410.19940 doi:10.48550/ARXIV.2410.19940
- [22] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David A. Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. seL4: formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, Jeanna Neefe Matthews and Thomas E. Anderson (Eds.). ACM, 207–220. doi:10.1145/1629575.1629596
- [23] Andrei Kozyrev, Nikita Khranov, Gleb Solovov, and Anton Podkopaev. 2026. RocqStar: Leveraging Similarity-driven Retrieval and Agentic Systems for Rocq generation. arXiv:2505.22846 [cs.LG] <https://arxiv.org/abs/2505.22846>
- [24] Robbert Krebbers, Xavier Leroy, and Freek Wiedijk. 2014. Formal C Semantics: CompCert and the C Standard. In *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8558)*, Gerwin Klein and Ruben Gamboa (Eds.). Springer, 543–548. doi:10.1007/978-3-319-08970-6\_36
- [25] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. 2016. CompCert – A Formally Verified Optimizing Compiler. In *ERTS 2016: Embedded Real Time Software and Systems*. SEE. [http://xavierleroy.org/publi/erts2016\\_compcert.pdf](http://xavierleroy.org/publi/erts2016_compcert.pdf)
- [26] Minghai Lu, Benjamin Delaware, and Tianyi Zhang. 2024. Proof Automation with Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, Vladimir Filkov, Baishakhi Ray, and Minghui Zhou (Eds.). ACM, 1509–1520. doi:10.1145/3691620.3695521
- [27] Linhao Luo, Jiaxin Ju, Bo Xiong, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2025. ChatRule: Mining Logical Rules with Large Language Models for Knowledge Graph Reasoning. In *Advances in Knowledge Discovery and Data Mining - 29th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2025, Sydney, NSW, Australia, June 10-13, 2025, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 15871)*, Xintao Wu, Myra Spiliopoulou, Can Wang, Vipin Kumar, Longbing Cao, Yanqiu Wu, Yu Yao, and Zhangkai Wu (Eds.). Springer, 314–325. doi:10.1007/978-981-96-8173-0\_25
- [28] Gregory Malecha, Gregory Malecha, and Yishuai Li. 2024. coq-ext-lib. <https://github.com/rocq-community/coq-ext-lib>
- [29] OpenAI. 2025. Introducing GPT-5.2. <https://openai.com/index/introducing-gpt-5-2/>
- [30] Zoe Paraskevopoulou. 2026. Machine-Generated, Machine-Checked Proofs for a Verified Compiler (Experience Report). arXiv:2602.20082 [cs.PL] <https://arxiv.org/abs/2602.20082>
- [31] Christine Paulin-Mohring. 2015. Introduction to the calculus of inductive constructions. *All about Proofs, Proofs for All* 55 (2015). <https://inria.hal.science/hal-01094195v1/file/CIC.pdf>
- [32] Gabriel Poesia and Noah D. Goodman. 2022. Peano: Learning Formal Mathematical Reasoning. *CoRR* abs/2211.15864 (2022). arXiv:2211.15864 doi:10.48550/ARXIV.2211.15864
- [33] Yicheng Qian, Joshua Clune, Clark Barrett, and Jeremy Avigad. 2025. Lean-auto: An Interface between Lean 4 and Automated Theorem Provers. arXiv:2505.14929 [cs.LO] <https://arxiv.org/abs/2505.14929>
- [34] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389. doi:10.1561/15000000019
- [35] Alex Sanchez-Stern, Yousef Alhessi, Lawrence K. Saul, and Sorin Lerner. 2020. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2020, London, UK, June 15, 2020*, Koushik Sen and Mayur Naik (Eds.). ACM, 1–10. doi:10.1145/3394450.3397466
- [36] Parshin Shojae, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. 2025. LLM-SR: Scientific Equation Discovery via Programming with Large Language Models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. <https://openreview.net/forum?id=m2nmp8P5in>
- [37] Yuheng Su, Qiusong Yang, Yiwei Ci, Tianjun Bu, and Ziyu Huang. 2025. The rIC3 Hardware Model Checker. In *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 15931)*, Ruzica Piskac and Zvonimir Rakamaric (Eds.). Springer, 185–199. doi:10.1007/978-3-031-98668-0\_9
- [38] Yuheng Su, Qiusong Yang, Yiwei Ci, Yingcheng Li, Tianjun Bu, and Ziyu Huang. 2025. Deeply Optimizing the SAT Solver for the IC3 Algorithm. In *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 15931)*, Ruzica Piskac and Zvonimir Rakamaric (Eds.). Springer, 185–199. doi:10.1007/978-3-031-98668-0\_9

- Ruzica Piskac and Zvonimir Rakamaric (Eds.). Springer, 237–257. doi:10.1007/978-3-031-98668-0\_12
- [39] The Coq Development Team. 2024. The Coq Reference Manual – Release 8.20.0. <https://rocq-prover.org/doc/V8.20.0/refman/index.html>
- [40] Kyle Thompson, Nuno Saavedra, Pedro Carrott, Kevin Fisher, Alex Sanchez-Stern, Yuriy Brun, João F. Ferreira, Sorin Lerner, and Emily First. 2025. Rango: Adaptive Retrieval-Augmented Proving for Automated Software Verification. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 347–359. doi:10.1109/ICSE55347.2025.00161
- [41] Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2024. LEGO-Prover: Neural Theorem Proving with Growing Libraries. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. <https://openreview.net/forum?id=3f5PALef5B>
- [42] Yile Wang, Sijie Cheng, Zixin Sun, Peng Li, and Yang Liu. 2024. Speak It Out: Solving Symbol-Related Problems with Symbol-to-Language Conversion for Language Models. arXiv:2401.11725 [cs.CL] <https://arxiv.org/abs/2401.11725>
- [43] Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. 2008. The Isabelle Framework. In *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5170)*, Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar (Eds.). Springer, 33–38. doi:10.1007/978-3-540-71067-7\_7
- [44] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with Large Language Models. arXiv:2205.12615 [cs.LG] <https://arxiv.org/abs/2205.12615>
- [45] Yutong Xin, Jimmy Xin, Gabriel Poesia, Noah D. Goodman, Qiaochu Chen, and Isil Dillig. 2025. Automated Discovery of Tactic Libraries for Interactive Theorem Proving. *Proc. ACM Program. Lang.* 9, OOPSLA2 (2025), 1974–2001. doi:10.1145/3763121
- [46] Kaiyu Yang and Jia Deng. 2019. Learning to Prove Theorems via Interacting with Proof Assistants. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6984–6994. <http://proceedings.mlr.press/v97/yang19a.html>
- [47] Bowen Zhang and Harold Soh. 2024. Extract, Define, Canonicalize: An LLM-based Framework for Knowledge Graph Construction. arXiv:2404.03868 [cs.CL] <https://arxiv.org/abs/2404.03868>

Received 2026-03-26; accepted 2026-06-18