



北京大学

本科生毕业论文

题目：面向 LLM 自动定理证明的探针式工具接入
配置优化

Probe-Guided Tool Integration
Configuration for LLM-Based
Automated Theorem Proving

姓 名：	吕杭州
学 号：	2200013126
院 系：	信息科学技术学院
专 业：	软件工程
导师姓名：	熊英飞

二〇二六年五月

北京大学本科毕业论文导师评阅表

学生姓名	吕杭州	本科院系	信息科学技术学院	论文成绩	【请填写成绩】
学生学号	2200013126	本科专业	软件工程	(等级制)	
导师姓名	熊英飞	导师单位/ 所在学院	计算机学院	导师职称	长聘副教授
论文题目	中文	面向 LLM 自动定理证明的探针式工具接入配置优化			
	英文	Probe-Guided Tool Integration Configuration for LLM-Based Automated Theorem Proving			
导师评语 【请填写评语】 导师签名: 斯公子 2026 年 5 月 8 日					

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任。

摘要

随着大语言模型被引入形式化证明，自动定理证明系统逐渐发展为结合检索、符号工具、证明助手反馈和反思验证的多模块工作流。在这类系统中，证明效果不仅取决于后端模型能力，也取决于工具上下文的接入方式。由于不同模型对定义、引理和证明示例的利用方式不同，模型升级后重新穷举评测全部工具配置会带来较高调参成本。

本文研究面向 LLM 自动定理证明的工具接入配置优化问题，提出使用少量探针定理进行低成本筛选。该方法通过小规模预评测构造候选定理的行为向量，衡量定理区分不同模型-配置组合的能力；再结合静态难度特征、定理相似度和 token 成本，将探针选择建模为带预算约束的子集选择问题，并用贪心算法求解。

实验以 ReCENT-PROVER 为基础系统，在 CoqStoq 样本上评估该方法。结果显示，MiniMax M2.5、OpenAI o4-mini、Qwen3-235B-A22B 和 GLM-5 上的探针选择都能在验证集上保持一致，同时不同模型表现出不同工具接入偏好。结果表明，探针定理能够保留配置优劣信号，可用于降低 LLM 证明系统在模型升级后的重新适配成本。

关键词: 大语言模型，自动定理证明，工具接入配置，探针定理，Rocq

ABSTRACT

As large language models (LLMs) are increasingly used in formal proof automation, automated theorem proving systems are evolving into multi-module workflows that combine retrieval, symbolic tools, proof-assistant feedback, and reflection-based validation. Their performance depends not only on the backend model, but also on how definitions, lemmas, and proof examples are integrated. Since different models may prefer different tool-calling patterns, exhaustively retuning all configurations after each model upgrade is costly.

This thesis studies tool integration configuration for LLM-based automated theorem proving and proposes a probe-guided method for low-cost configuration screening. The method builds behavior vectors from small-scale profiling, uses them to estimate how well candidate theorems distinguish model-configuration combinations, and selects probes by combining difficulty features, theorem similarity, and token cost in a budget-constrained greedy procedure.

We evaluate the method on CoqStoq using RECENT-PROVER with MiniMax-M2.5, OpenAI o4-mini, Qwen3-235B-A22B, and GLM-5. The probe-selected configuration is also selected on the validation set for all four models, while the preferred tool integration pattern differs across models. These results suggest that probe theorems preserve useful configuration signals and can reduce the cost of adapting LLM-based proof systems after model upgrades.

KEY WORDS: Large Language Models, Automated Theorem Proving, Tool Integration Configuration, Probe Theorems, Rocq

目 录

1 引言	1
2 背景介绍	3
2.1 前置知识	3
2.1.1 Rocq 证明助手	3
2.1.1.1 简单逻辑系统 Sys_{all} 及其可靠性定理	3
2.1.1.2 证明过程	6
2.1.2 CoqHammer	7
2.1.3 检索增强生成	8
2.2 RECENT-PROVER 简介	9
2.3 问题: RECENT-PROVER 中的最优配置	10
3 方法实现	12
3.1 方法总览	12
3.2 定理特征与信息量	14
3.3 子集选择目标	15
3.4 具体算法	16
3.5 面向模型升级的使用流程	19
3.6 实现细节	20
4 实验评估	21
4.1 实验设置	21
4.2 实验结果	22
4.2.1 MiniMax M2.5 结果	22
4.2.2 OpenAI o4-mini 结果	23
4.2.3 Qwen3-235B-A22B 结果	23
4.2.4 GLM-5 结果	23
4.3 问题 1: 探针能否作为模型能力代理	24
4.4 问题 2: 如何用探针同时衡量能力和成本	24
4.5 问题 3: 不同模型是否表现出不同工具调用偏好	25
4.6 实验局限	25
5 相关工作	27
5.1 基于 LLM 的自动定理证明	27
5.2 LLM 智能体工作流优化	27
5.3 面向模型能力的技能与工具适配	29
5.3.1 工具调用能力优化	29

5.3.2 技能选择与技能优化 30

5.4 本文与相关工作的差异 30

6 总结与展望 32

参考文献 35

致谢 39

北京大学学位论文原创性声明和使用授权说明 40

1 引言

软件系统已经进入航空航天、医疗设备、金融交易和自动驾驶等关键领域。系统规模越大，软件缺陷造成的后果也越难控制。医疗设备软件缺陷曾导致患者伤亡^[1]，金融交易系统中的程序错误可能造成巨额经济损失^[2]，航空航天系统中的软件故障也可能直接威胁生命安全^[3]。这些案例说明，对于高安全性系统，仅依赖测试很难给出足够强的正确性保证。

形式化验证（Formal Verification）通过数学证明来说明程序满足给定规范，是提高软件可靠性的一条重要路径。近年来，Rocq（前身为 Coq）^[4]、Lean^[5]和 Isabelle^[6]等证明助手已经较为成熟，并被用于操作系统^[7]、编译器^[8]和数学定理证明^[9-11]等场景。这类工具的优势在于证明结果可以由小而可信的内核检查；但代价也很明显，用户通常需要手工编写大量证明脚本。

在证明助手中开发形式化证明是一项复杂且耗时的工作^[12]。证明者需要理解程序语义、库中已有引理以及证明助手的策略语言，还要反复处理证明状态变化带来的细节问题。这一成本限制了形式化方法在更大范围内的应用。因此，自动化定理证明（Automated Theorem Proving, ATP）长期以来都是形式化方法中的重要研究方向。

传统自动定理证明技术^[13-16]主要依赖符号推理和搜索。它们在许多局部目标上非常有效，但在复杂定理上容易受到搜索空间规模的限制，也难以直接利用人类证明中常见的高层思路。近年来，大语言模型（Large Language Model, LLM）在代码生成和复杂推理任务中表现出较强能力，为自动定理证明提供了新的思路。LLM 可以根据当前证明状态生成候选策略，也可以用自然语言计划、已有证明片段和错误反馈来调整搜索方向。

不过，LLM 本身并不能替代证明助手的检查机制。模型生成的策略可能语法错误，也可能在局部被接受后把证明带入更困难的分支。为了让 LLM 证明系统稳定工作，通常需把模型生成、证明助手反馈、符号工具、检索模块和失败历史管理结合起来。RECENT-PROVER^[17]就是这类系统的一个例子：它在 Rocq 环境中结合 CoqHammer、LLM 证明生成、自反思验证和规划驱动检索，用证明助手保证最终脚本的可检查性，同时让 LLM 参与证明计划和策略选择。

这类系统带来了一个新的工程问题：工具已经接入之后，应该如何决定它们的调用方式。以环境定义、相关引理和相关证明示例为例，每类信息都可以选择每轮都获取、只获取一次，或按需获取。仅这三类信息源就形成 $3^3 = 27$ 种配置。如果再加入模型选择、检索参数和证明搜索预算，完整评测的成本会迅速增加。直接在完整定理集合上比较所有配置虽然最可靠，但不适合作为早期调参手段。

更重要的是，工具调用方式很难在所有模型之间固定复用。不同模型的上下文利用能力、长文本鲁棒性、工具调用倾向和证明规划能力都不同。同一段相关引理或证明示例，对一个模型可能是必要的提示，对另一个模型可能只是噪声和额外 token 开销；同一种 *on demand* 策略，对能够准确判断信息缺口的模型可能有效，对工具需求判断较弱的模型则可

能导致关键上下文缺失。与此同时，后端模型本身又在持续升级。一个实际的 LLM 证明系统很难在每次模型替换后，都重新对完整配置空间做大规模评测。因此，本文的目标是降低“针对不同模型重新选择工具调用和 agent 设计”的成本，而不只是在当前样本中寻找某一个固定最优配置。

本文关注的问题正是：在完整评测成本较高的情况下，能否用少量有代表性的定理来快速筛选工具接入配置，并进一步观察不同模型是否真的需要不同的 agent 工具接入设计。本文将这类定理称为探针定理（Probe）。一个好的探针集合不能只由困难定理组成，还应尽量包含能够区分不同配置、同时彼此不高度重复的样本。基于这一思路，本文先通过小规模预评测估计候选定理对配置差异的区分能力，再结合定理相似度和 token 成本，在预算约束下选择探针集合。随后，本文在 RECENT-PROVER 的配置选择任务上进行验证，观察探针集上选出的配置能否在独立验证集上保持优势，以及模型之间的最优配置是否出现差异。

本文的主要工作包括以下三个方面。

- 提出面向 LLM 证明系统工具接入配置的探针选择问题，将低成本配置筛选建模为带预算约束的定理子集选择问题。
- 设计基于预评测行为向量的探针信息量度量，并结合静态难度特征、定理相似度和 token 成本，用贪心策略选择探针集合。
- 在 CoqStoq^[18]样本上以 RECENT-PROVER 为实验系统，对 MiniMax M2.5^[19]、OpenAI o4-mini^[20]、Qwen3-235B-A22B^[21]和 GLM-5^[22]四个后端模型进行初步验证。实验结果显示，在本文的两阶段评测准则下，探针集选出的配置在验证集上也取得了最优效率；同时，不同模型的最优配置并不完全一致，说明 agent 工具接入设计需要随模型能力变化而调整。

章节安排如下：第 2 章回顾 Rocq 证明助手^[4]、CoqHammer^[13]、检索增强生成以及 RECENT-PROVER 相关背景；第 3 章介绍探针定理选择方法，包括问题建模、信息量度量和贪心选择算法；第 4 章给出实验设置和结果分析；第 5 章讨论相关工作；第 6 章总结全文并讨论不足与后续工作。

2 背景介绍

2.1 前置知识

2.1.1 Rocq 证明助手

Rocq（前身为 Coq）^[4]是一个交互式定理证明器，它基于依赖类型理论，提供了一个强大的环境来编写形式化证明。Rocq 允许用户定义数据类型、函数和定理，并通过交互式的方式构建证明。Rocq 支持多种证明策略，包括直接证明、归纳证明等，同时还提供了丰富的库和工具来辅助证明开发。下面我们将通过证明一个简单逻辑系统 Sys_{all} 的可靠性定理来详细介绍 Rocq 的基本使用方法。

2.1.1.1 简单逻辑系统 Sys_{all} 及其可靠性定理

定义 2.1.1 (形式语言 L_{all}^S) 给定一个符号集合 S ，考虑一个形式语言 L_{all}^S ，它只包含下面的形式的句子：

$$\text{All } X \text{ are } Y. \quad (2.1)$$

定义 2.1.1 可以在 Rocq 中被形式化为如下的定义：

```
(** * 1. 形式语言 (Formal Language) *)
(** 符号集合 S - 我们用可数类型表示 *)
Parameter Symbol : Type.
(** 假设符号相等是可判定的 *)
Parameter symbol_eq_dec : forall (x y : Symbol), {x = y} + {x <> y}.
(** 语言 L_all 中的句子 (公式) *)
Inductive Formula : Type :=
  | All : Symbol -> Symbol -> Formula. (* All X are Y *)
Notation "'All' X 'are' Y" := (All X Y) (at level 60).
```

代码 2.1 形式语言 L_{all}^S 的 Rocq 定义

定义 2.1.2 (形式系统 Sys_{all}) 给定形式语言 L_{all}^S ，考虑下面两条推理规则：

$$\frac{\text{All } X \text{ are } Y \quad \text{All } Y \text{ are } Z}{\text{All } X \text{ are } Z} \text{Trans}$$

$$\frac{}{\text{All } X \text{ are } X} \text{Id}$$

图 2.1 Sys_{all} 的推理规则

那么我们可以定义一个形式系统 Sys_{all} ，它包含上面两条推理规则。

定义 2.1.3 (推演与证明) 给定一个形式系统 Sys ，给定前提集 Γ ，一个从 Γ 出发、以 φ 结尾的推演是一个有穷公式序列

$$\varphi_1, \varphi_2, \dots, \varphi_n = \varphi, \quad (2.2)$$

满足对每个 $i \leq n$, φ_i 都属于以下三种情况之一：

- φ_i 是形式系统 Sys 中的公理的实例；
- φ_i 是前提集 Γ 中的一个公式；
- φ_i 是通过应用推理规则从前面的公式推导出来的。

如果存在这样的推演，我们就说 Γ 在形式系统 Sys 里推出 φ ，记作 $\Gamma \vdash_{\text{Sys}} \varphi$ (当我们讨论的形式系统是清楚的时候，我们也简写成 $\Gamma \vdash \varphi$)。此时我们也称 φ 是 Γ 的语形后承 (Syntactic Consequence)。特别地，当前提集为空时，推演也叫做证明。如果 $\emptyset \vdash_{\text{Sys}} \varphi$ ，就简写为 $\vdash_{\text{Sys}} \varphi$ ，并说 φ 是可证的。

```
(** * 2. 推理系统 (Deduction System) *)
(** 推理规则:  $\Gamma \vdash \varphi$  表示从前提集合  $\Gamma$  可以推导出  $\varphi$  *)
Reserved Notation "Gamma ' $\vdash$ ' phi" (at level 70).
Inductive Derivable : Ensemble Formula -> Formula -> Prop :=
  | D_Axiom : forall  $\Gamma$   $\varphi$ ,
    In Formula  $\Gamma$   $\varphi$  ->
       $\Gamma \vdash \varphi$ 
  | D_Id : forall  $\Gamma$  X,
     $\Gamma \vdash (\text{All } X \text{ are } X)$ 
  | D_Trans : forall  $\Gamma$  A B C,
     $\Gamma \vdash (\text{All } A \text{ are } B)$  ->
     $\Gamma \vdash (\text{All } B \text{ are } C)$  ->
     $\Gamma \vdash (\text{All } A \text{ are } C)$ 
where "Gamma ' $\vdash$ ' phi" := (Derivable Gamma phi).
```

代码 2.2 Sys_{all} 的 Rocq 定义

形式系统 Sys_{all} (定义 2.1.2) 和在形式系统 Sys_{all} 上的推演 (定义 2.1.3) 可以在 Rocq 中被形式化为代码 2.2。

定义 2.1.4 (L_{all}^S 的模型) 给定形式语言 L_{all}^S ，它的一个模型 $\mathcal{M} = \langle O, I \rangle$ ，其中：

- O 是一个非空集合，叫做对象域；
- $I : S \rightarrow \mathcal{P}(O)$ 是一个解释函数，将每个符号 $X \in S$ 解释成 O 中的一个子集 $I(X) \subseteq O$ 。

$\mathcal{M}$ 满足一个句子 $\text{All } X \text{ are } Y$ 当且仅当 $I(X) \subseteq I(Y)$ 。记作 $\mathcal{M} \models \varphi$ ，这里的 $\varphi = \text{All } X \text{ are } Y$ 。

定义 2.1.5 (满足关系) 对任意模型 $\mathcal{M} = \langle O, I \rangle$, 定义

$$\mathcal{M} \models \text{All } X \text{ are } Y \text{ 当且仅当 } I(X) \subseteq I(Y). \quad (2.3)$$

如果 $\mathcal{M} \models \varphi$, 就说 φ 在 $\mathcal{M}$ 上为真, 或者说 $\mathcal{M}$ 满足 φ 。如果对每个 $\varphi \in \Gamma$ 都有 $\mathcal{M} \models \varphi$, 则记作 $\mathcal{M} \models \Gamma$ 。

定义 2.1.6 (语义后承) 对任意前提集 Γ 和公式 φ , 如果对于每个模型 $\mathcal{M}$, $\mathcal{M} \models \Gamma$ 都蕴含 $\mathcal{M} \models \varphi$, 那么我们说 φ 是 Γ 的语义后承, 记作 $\Gamma \models \varphi$ 。

我们也可以在 Rocq 中形式化这个系统的语义后承 (定义 2.1.6) 为代码 2.3。现在让我们考虑这个系统的可靠性 (Soundness) 定理, 也就是定理 2.1.7。

定理 2.1.7 (可靠性定理) 对于任意的前提集 Γ 和公式 φ , 如果 $\Gamma \vdash \varphi$, 那么 $\Gamma \models \varphi$ 。

```
(** * 3. 形式语义 (Semantics) *)
(** 对象域 (论域): 这里取为符号集合本身 *)
Definition Object : Type := Symbol.
(** 模型 M = (O, I)
    - O 是对象集合 (我们用 Object 类型表示)
    - I 是解释函数: Symbol → Ensemble Object
*)
Record Model : Type := mkModel {
  interpretation : Symbol → Ensemble Object
}.
(** 句子在模型中的真值: M ⊨ φ *)
Definition satisfies (M : Model) (φ : Formula) : Prop :=
  match φ with
  | All X are Y => Included Object (interpretation M X) (interpretation M Y)
  end.
Notation "M '⊨' phi" := (satisfies M phi) (at level 65).
(** 模型满足公式集合 *)
Definition satisfies_set (M : Model) (Γ : Ensemble Formula) : Prop :=
  forall φ, In Formula Γ φ -> satisfies M φ.
(** 语义后承: Γ ⊨ φ *)
Definition entails (Γ : Ensemble Formula) (φ : Formula) : Prop :=
  forall M : Model, satisfies_set M Γ -> M ⊨ φ.
Notation "Gamma '⊨s' phi" := (entails Gamma phi) (at level 80, no associativity).
```

代码 2.3 形式语义的 Rocq 定义

```
(** 可靠性定理 (Soundness): 如果  $\Gamma \vdash \varphi$ , 那么  $\Gamma \models \varphi$  *)
Theorem soundness : forall  $\Gamma \varphi$ ,
   $\Gamma \vdash \varphi \rightarrow \Gamma \models \varphi$ .
```

代码 2.4 可靠性定理

2.1.1.2 证明过程

现在我们已经可以在 Rocq 中定义了形式语言、推理系统和形式语义，并且陈述了我们需要证明的可靠性定理。接下来，我们将进入证明环境来构建这个定理的证明。

一旦我们进入了证明环境，Rocq 就会返回给我们一系列的子目标（subgoals），我们需要逐一证明这些子目标来完成整个证明过程。其中每一个子目标包含一系列的前提和一个需要证明的结论。刚进入证明环境时，我们会得到一个初始的子目标，如图 2.2 所示。这个子目标没有前提，而结论就是我们需要证明的可靠性定理的叙述。

$$\frac{[\text{No premise}]}{\text{forall } (\Gamma : \text{Ensemble Formula}) (\varphi : \text{Formula}), \Gamma \vdash \varphi \rightarrow \Gamma \models \varphi}$$

图 2.2 可靠性定理的初始子目标

为了证明子目标，我们需要 Rocq 提供的各种证明策略（tactics），例如 `intros`、`apply`、`destruct` 等等。一个策略会消耗掉当前的子目标，并且可能产生新的子目标，或者不产生任何新的目标（也就意味着当前的子目标已经被证明）。Rocq 会按顺序应用所有的策略，不断地更新当前的子目标。一般来说，一个策略被默认应用在当前的子目标上^①，并且新产生的子目标会被添加到队列的头部。

同时，Rocq 还允许用户使用自定义的证明策略，这些策略可以通过组合现有的基本策略、通过 `match goal` 对当前证明上下文（假设、目标）进行模式匹配以及使用基本的控制流来实现更复杂的证明步骤。通过合理地选择和组合这些策略，用户可以高效地完成整个证明过程。

现在，我们在图 2.2 的基础上，应用 `intros` 策略来引入所有的量词和假设。这个策略会将当前的子目标中的所有前提引入到证明上下文中，并且将结论中的量词去掉。应用完这个策略后，我们会得到一个新的子目标，如图 2.3 所示。

$$\frac{\begin{array}{l} \Gamma : \text{Ensemble Formula} \\ \varphi : \text{Formula} \\ H_deriv : \Gamma \vdash \varphi \end{array}}{\Gamma \models \varphi}$$
图 2.3 应用 `intros` 后的子目标

下一步，我们应用 `induction H_deriv`，这个策略会对我们引入的假设 `H_deriv` 进行归纳分析。由于 `H_deriv` 是一个关于推导关系的假设，它的结构是由我们之前定义的推导规则

^①不过，`Set Default Goal Selector "all"` 会默认作用于所有目标，`Set Default Goal Selector "!"` 会强制要求每步必须覆盖所有目标。

构成的。因此，应用 `induction` 会根据这些规则来生成相应的子目标。每个子目标对应于一个推导规则的情况，我们需要分别证明每个子目标来完成整个证明过程。这个过程如图 2.4 所示。

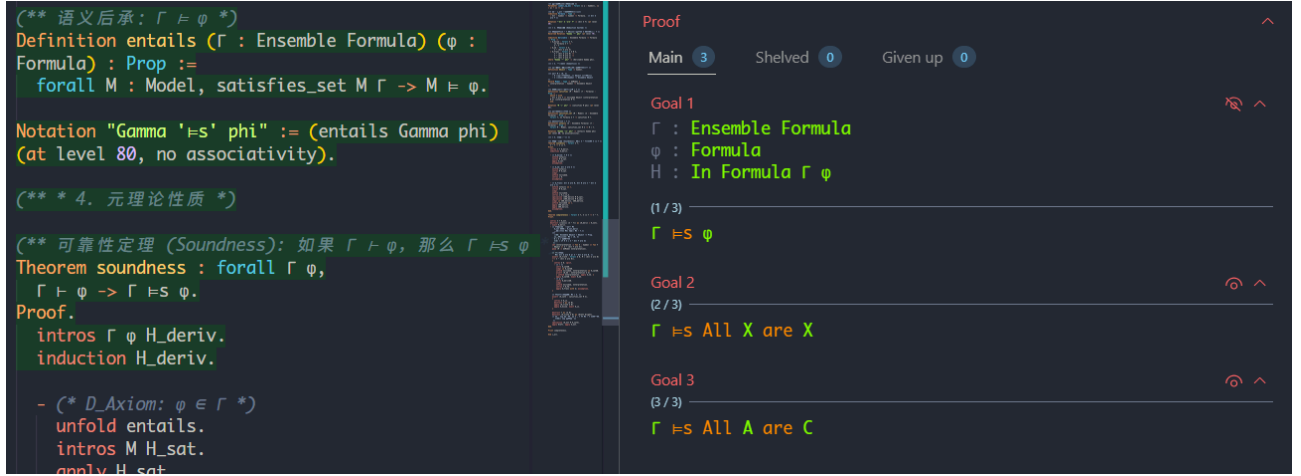


图 2.4 应用 `induction` 后的子目标

后续证明步骤较为直接，本文不再展开。总的来说，定理证明的目标就是找到一个策略序列，使得当我们从初始子目标开始，依次应用这些策略后，最终所有的子目标都得到证明。

2.1.2 CoqHammer

Hammer 系统是基于外部自动定理证明器的证明辅助工具，旨在自动为用户提供给定猜想的证明。常见的 Hammer 系统包括 Isabelle 的 Sledgehammer^[14]。而 CoqHammer^[13]是一个面向依赖类型理论的完整 Hammer 系统，专为 Rocq 证明助手设计。其核心思路是：先用机器学习算法（ k -近邻和朴素贝叶斯算法）从庞大的引理库中筛选出最相关的候选前提，再将目标翻译为无类型一阶逻辑，交由 Eprover^[23]、Vampire^[24]、Z3^[25]、CVC4^[26]等外部自动化定理证明器求解，最后将所得证明在 Rocq 内部重新验证，确保结果完全符合 Rocq 自身的直觉主义逻辑。

下面我们通过继续证明可靠性定理（定理 2.1.7）来介绍 CoqHammer 的使用方法。在第 2.1.1.2 节中，我们已经完成了部分证明过程，接下来使用 CoqHammer 自动完成剩余的证明步骤。在 `induction H_deriv` 之后，需要证明的第一个子目标如图 2.5 所示；该子目标可以由 CoqHammer 直接完成。如代码 2.5 所示，在这个子目标上应用 `hammer` 策略后，CoqHammer 会自动寻找证明，并报告可替换的证明策略，例如 `Replace the hammer tactic with: sfirstorder`。在这个例子中，也可以直接使用 `induction H_deriv; hammer` 一次性完成所有子目标。

```

From Hammer Require Import Hammer.
Set Hammer ATPLimit 50.
Set Hammer ReconstrLimit 50.
(** 为了简洁起见省去 *)
Theorem soundness : forall  $\Gamma$   $\varphi$ ,
   $\Gamma \vdash \varphi \rightarrow \Gamma \models \varphi$ .
Proof.
  intros  $\Gamma$   $\varphi$  H_deriv.
  induction H_deriv.
  - (* D_Axiom:  $\varphi \in \Gamma$  *)
    hammer.
    (* 经过 hammer 之后，这个子目标就得到证明 *)
    (** 下面是原先的手动证明过程：
    unfold entails.
    intros M H_sat.
    apply H_sat.
    assumption.
    *)

```

代码 2.5 使用 CoqHammer 证明可靠性定理

$$\begin{array}{l}
 \Gamma : \text{Ensemble Formula} \\
 \varphi : \text{Formula} \\
 H : \text{In Formula } \Gamma \varphi \\
 \hline
 \Gamma \models \varphi
 \end{array}$$

图 2.5 需要证明的第一个子目标

2.1.3 检索增强生成

检索增强生成（Retrieval-Augmented Generation, RAG）^[27]是指在语言模型生成阶段，引入检索机制，让模型先从外部知识库检索相关文档，再将这些文档作为上下文输入至语言模型，从而生成回答或文本，如图 2.6 所示。

其中的检索器负责根据查询向量从大规模文本库中检索相关文档或段落，其策略通常分为两类：稀疏检索与密集检索。稀疏检索基于倒排索引及 TF-IDF、BM25 等词频统计方法，具有计算高效、可解释性强的优势；密集检索则通过预训练语言模型将文本编码为稠密向量，利用向量空间的语义相似度进行匹配，以 DPR^[28]为代表，能够捕捉词汇层面之外的语义关联。

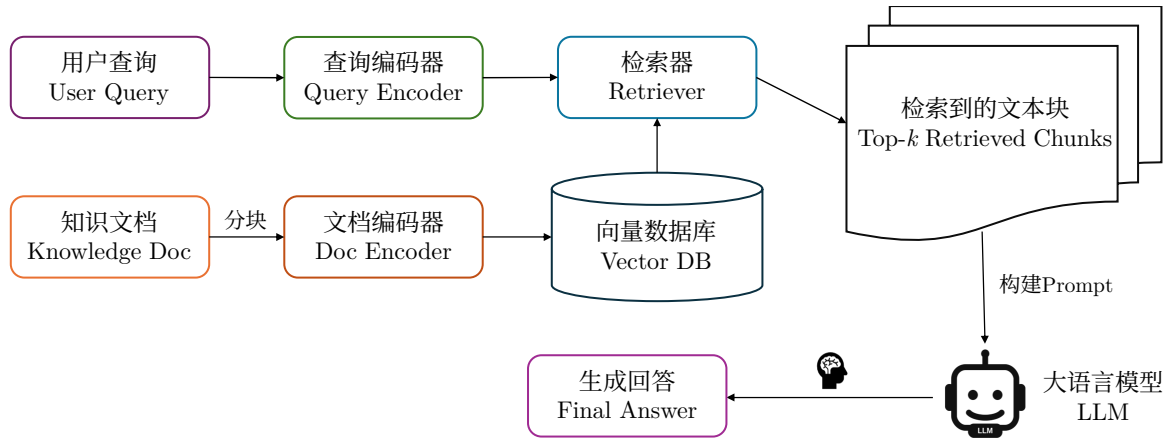


图 2.6 RAG 系统的架构示意图

2.2 RECENT-PROVER 简介

RECENT-PROVER^[17]是一个结合大型语言模型（LLM）、检索增强技术（RAG）、Rocq 证明助手和 CoqHammer 的自动定理证明系统。它主要引入两项机制：自反思验证（validation with reflection）让 LLM 审查自身生成的策略，并在发现潜在误用时记录失败原因；规划驱动检索（retrieval with planning）则根据 LLM 生成的证明方案检索相关引理和证明，而非仅依赖子目标的文本相似性。在 CoqStoq^[18]上，RECENT-PROVER 证明的定理数量较现有最先进方法 COBBLESTONE^[29]提升了 22.58%。

如图 2.7 所示，该系统首先检查是否存在剩余子目标：若所有子目标均已解决，则报告证明成功并返回完整的形式化证明；若已达到迭代上限 T ，则报告证明失败；若两者均不满足，则继续进行证明。在继续证明时，该系统尝试证明剩余子目标列表中的第一个子目标 S 。对于每个待证子目标 S ，系统首先调用符号求解器 CoqHammer 尝试直接求解；若失败，则转入大语言模型驱动的策略生成流程，依次执行规划驱动检索、提示词构造、策略生成与自反思验证，将通过验证的策略应用于证明状态，并将失败记录写入历史以指导下一轮迭代，如此循环直至证明完成或达到迭代上限。

下面我们分别介绍该系统的几项关键机制：

1. 自反思验证机制。现有证明智能代理通常会保留所有通过语法验证的策略，即使这些策略在语义上存在误用，例如对错误变量归纳或在归纳前未充分泛化假设。为此，RECENT-PROVER 对可能产生多个子目标或导致不可证状态的高风险策略（如 `apply`、`induction` 等）触发自反思模块，由 LLM 判断策略是否误用，并在确认失败时记录相关子目标、策略序列和失败原因。这一机制可以更早截断无效搜索路径，提升证明搜索的健壮性。
2. 规划驱动检索机制。仅依赖子目标与引理文本相似度（如 BM25）的检索方法，容易在包含领域特定符号的目标上返回语法相似但证明无关的引理。RECENT-PROVER 改以证明规划为检索中心：如图 2.8 所示，系统先让 LLM 为当前子目标生成自然语言

言证明计划，再用计划步骤的嵌入向量检索预先索引的引理描述；证明示例的检索也基于证明计划嵌入进行匹配。该方案将检索依据从语法表层转向策略层语义，更容易找到与预期证明思路对齐的引理与示例。

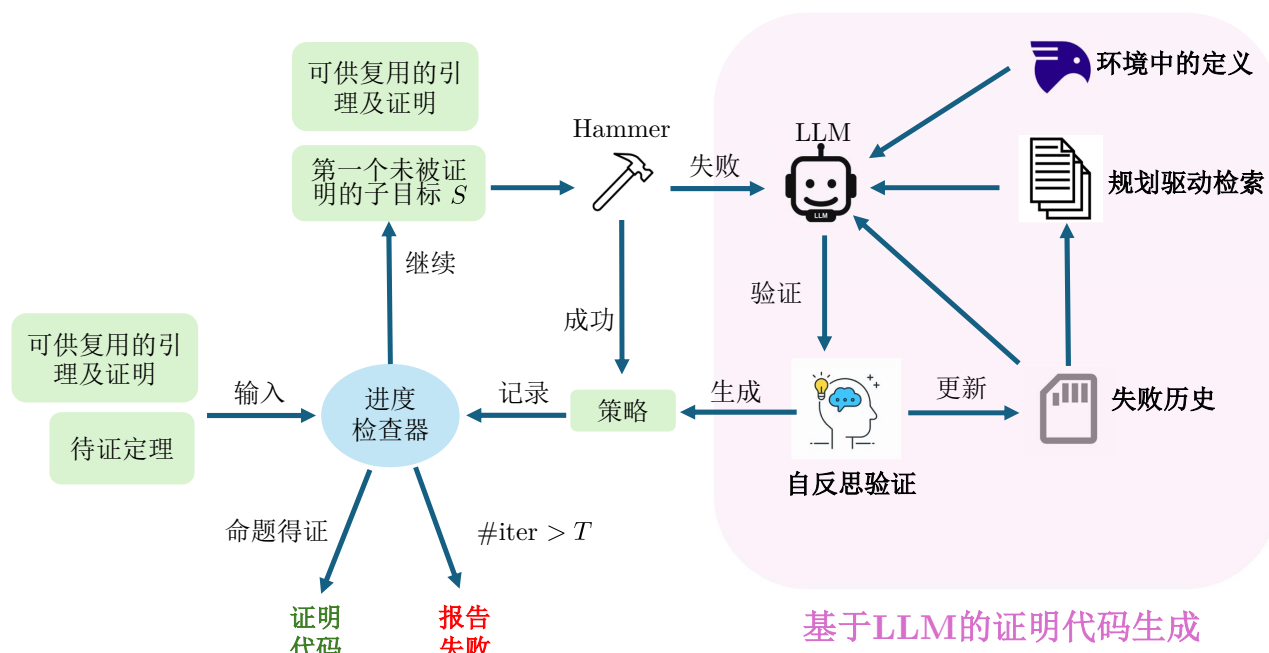


图 2.7 RECENT-PROVER 系统的工作流

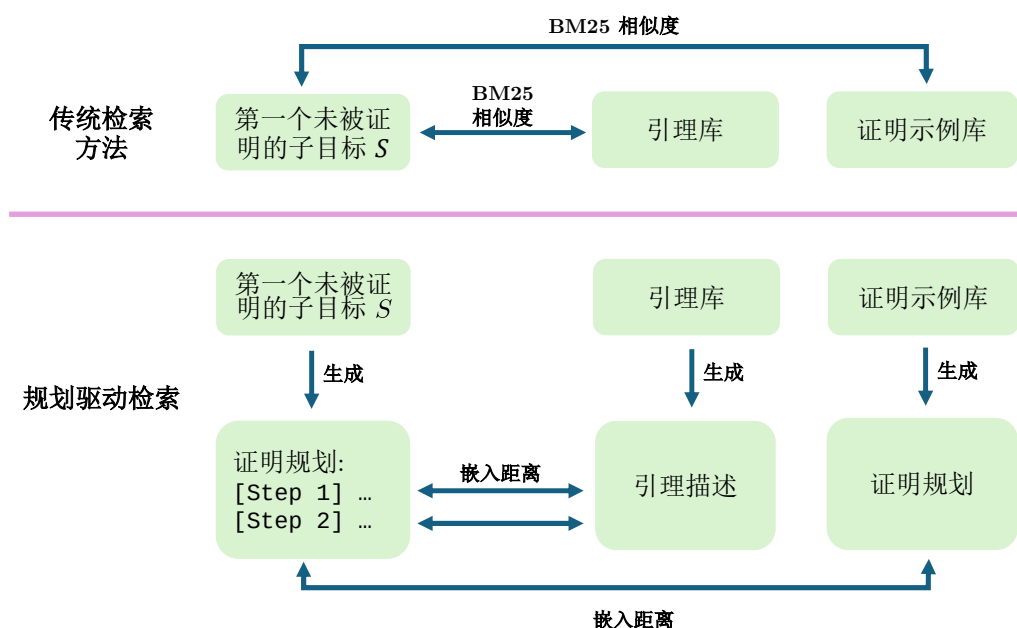


图 2.8 规划驱动检索机制

2.3 问题：RECENT-PROVER 中的最优配置

RECENT-PROVER 已经展示了将大语言模型与证明助手、符号求解器和检索模块结合起来的潜力。然而，这并不意味着当前的工作流就是最优的。自动证明系统不能简单地把所有可用工具都接入进去；不同工具应当在什么时机被调用、每次调用应当提供多少信息、信息

是否需要随着证明状态变化而更新，都会直接影响系统的证明能力和运行成本。过少的信息可能使模型无法生成有效策略，过多或过频繁的信息供给又可能带来上下文冗余、检索噪声和额外的 LLM 调用开销。

这一问题在多模型场景下会进一步放大。LLM 证明系统通常需要在不同后端模型之间切换：有时是因为更强模型发布，有时是因为价格、延迟、上下文长度或接口稳定性发生变化。模型变化之后，原先的工具接入策略不一定仍然合适。较强的模型可能能够从较少的定义和引理中推断出证明方向，因而更适合减少重复检索；另一些模型可能更依赖显式证明示例或每轮更新的相关引理，过早固定上下文会降低成功率。也就是说，agent 工作流中的工具调用设计需要同时考虑任务、工具和后端模型三者的匹配关系。

因此，ReCENT-PROVER 仍然面临一个更基础的问题：这样的 workflow 设计是否已经接近最优？更具体地说，系统中的各类工具和信息源应当如何接入，才能在证明成功率、搜索效率和调用成本之间取得更好的平衡？进一步地，当后端模型变化时，是否需要重新设计这一套 agent 工具调用方式？在本工作中，我们关注三类对证明生成影响较大的信息：当前环境中的定义、与目标相关的引理，以及可供参考的相关证明。这三类信息都能帮助模型理解证明上下文，但它们的最佳使用方式并不显然，也不一定能在不同模型之间直接迁移。

对于上述每一类信息，都至少存在三种自然的接入策略。第一种是 *always*，即在每一轮策略生成时都重新获取并提供该信息；这种方式信息最充分，也最能适应证明状态的变化，但代价最高，并且可能引入重复内容。第二种是 *once*，即只在证明开始或首次需要时获取一次，后续复用已有结果；这种方式开销较低，但当证明状态发生较大变化时，早期获取的信息可能不再足够相关。第三种是 *on demand*，即由模型或系统根据当前证明状态判断是否需要调用相应工具；这种方式更灵活，但也依赖模型能否准确判断自己缺少什么信息。

由此可见，获取环境定义、相关引理和相关证明并非固定的工程细节，它会影响整个证明搜索过程。我们的目标是系统地比较这些接入策略，探究在不同工具组合下，哪些设计能够更稳定地提升 ReCENT-PROVER 的证明效果，并进一步理解自动证明系统中工具调用时机、后端模型能力与证明性能之间的关系。

从工程角度看，真正需要降低的是模型升级后的重新适配成本。如果一个新模型发布后，研究者必须重新运行所有定理、所有配置和所有检索参数的组合，才能判断 agent 工作流是否仍然合适，那么系统迭代速度会受到严重限制。本文采用探针定理的思路，把完整评测之前的配置选择转化为一个小规模筛选问题：先用少量对配置差异敏感的定理观察模型与工具接入方式的匹配程度，再把更有希望的配置放入较大验证集。

3 方法实现

在第 2.3 节中已经提到，ReCENT-PROVER 的难点除了把外部工具接入 LLM 证明流程，还包括决定这些工具在什么时候、以什么频率被调用。以环境定义、相关引理和相关证明三类信息为例，每一类都可以采用 *always*、*once* 或 *on demand* 三种策略。仅这三类信息源就对应 $3^3 = 27$ 种配置；如果再加入模型选择、检索参数、提示词模板和证明搜索预算，完整评测空间很快就会变得难以承受。

本文特别关注这一问题中的模型适配维度。一个配置并不存在脱离后端模型的绝对优劣：同样的工具上下文可能帮助一个模型补足信息，也可能让另一个模型受到噪声干扰；同样的按需调用机制可能被一个模型正确触发，也可能被另一个模型过度或不足触发。因此，本文把候选对象看作“模型-配置”组合，而非孤立的工具配置。探针定理的目标也相应扩展为：用较低成本识别哪些配置适合当前模型，并观察不同模型之间的配置偏好是否一致。

最直接的做法是在完整定理集合上跑完所有模型与配置，但这通常不适合作为早期实验方案。一次证明尝试可能包含多轮 LLM 生成、检索和 Rocq 检查；失败样例还常常更贵，因为系统会反复尝试直到达到迭代上限。因此，本章采用一个更实际的折中：先从完整定理集合中挑出一小组有代表性的定理，用它们快速观察不同工具接入设计的差异。下文把这样的定理称为探针定理（Probe Theorems）。一个有用的探针集合不一定由最难的定理组成，还应覆盖不同难度、不同证明结构以及不同配置敏感性的样本。

为此，我们先用少量模型和少量代表性配置对候选定理做预评测（Profiling），粗略估计每个定理对配置差异的区分能力；随后再把定理之间的相似度和评测成本纳入考虑，在预算限制下选出一组信息量较高、重复度较低的探针。这个过程可以自然地写成一个带预算约束的子集选择问题：目标函数奖励区分能力，同时惩罚集合内部的冗余；求解时则使用贪心策略作为近似。

3.1 方法总览

方法的输入包括完整定理集合、候选模型以及待比较的工具接入配置，输出是一组规模较小的探针定理。之后的配置比较先在这组探针上进行，用较低成本筛掉明显不合适的设计，再把少数有希望的配置放到更大规模实验中验证。由于模型本身可能持续更新，探针集还可以作为模型替换后的快速体检集合：当一个新模型接入证明系统时，先在探针集上比较少量配置，就能初步判断原有 agent 工具接入策略是否仍然适用。

设完整定理集合为 $\mathcal{T} = \{t_1, t_2, \dots, t_N\}$ ，其中每个 t_i 是一个待证明的 Rocq 定理。设模型集合为 $\mathcal{M} = \{m_1, m_2, \dots, m_K\}$ ，其中每个 m_i 表示一个可用于生成证明策略的大语言模型。设配置集合为 $\mathcal{C} = \{c_1, c_2, \dots, c_L\}$ 。在本文关注的工具接入问题中，一个配置 $c \in \mathcal{C}$ 可以被看作一个三元组： $c = (a_{\text{def}}, a_{\text{lemma}}, a_{\text{proof}})$ 。其中 a_{def} 表示环境定义的获取策略， a_{lemma} 表示相关引理的获取策略， a_{proof} 表示相关证明的获取策略，并且

$$a_{\text{def}}, a_{\text{lemma}}, a_{\text{proof}} \in \{\text{always}, \text{once}, \text{on demand}\}. \quad (3.1)$$

因此，在只考虑这三类信息源时，共有 $3^3 = 27$ 种候选配置。

为了记录一次证明尝试的结果，定义评测函数

$$f(t, m, c) \in [0, 1]. \quad (3.2)$$

最简单的做法是把 $f(t, m, c)$ 看作二值函数：模型 m 在配置 c 下证明定理 t 成功则为 1，失败则为 0。如果需要更细的度量，也可以把证明长度、调用次数和运行时间等因素合并成归一化得分。后续选择过程并不依赖 f 的具体形式，只要求它能反映某个模型和配置在某个定理上的表现。

给定一个定理子集 $S \subseteq \mathcal{T}$ ，模型 m 和配置 c 在该子集上的平均表现定义为

$$F(m, c; S) = \frac{1}{|S|} \sum_{t \in S} f(t, m, c). \quad (3.3)$$

当 $S = \mathcal{T}$ 时， $F(m, c; \mathcal{T})$ 就是完整评测结果；当 S 是探针集合时， $F(m, c; S)$ 则是一个低成本估计。我们希望这个估计至少能保留模型-配置组合之间的相对差异。因此，探针集合不宜只包含一批对所有模型和配置都同样容易或同样困难的定理，还需要尽量保留那些会改变配置排序、甚至会改变不同模型最优 agent 设计的关键信号。

定理 3.1.1 (探针数量的信息论下界) 假设每个探针定理的评测结果只有成功和失败两种可能。若需要仅根据 n 个探针定理的结果区分 L 种候选配置，则必须满足

$$2^n \geq L. \quad (3.4)$$

特别地，当仅考虑环境定义、相关引理和相关证明三类信息源，且每类信息源都有 *always*、*once* 和 *on demand* 三种接入策略时，候选配置数为 $L = 3^3 = 27$ 。此时至少需要的探针定理的个数为：

$$n \geq \lceil \log_2 27 \rceil = 5. \quad (3.5)$$

证明. 对于任意一个探针定理，由于评测结果只有成功和失败两种可能，它最多提供 2 种不同的观测结果。因此， n 个探针定理共同形成的结果向量的取值个数最多有

$$\underbrace{2 \times 2 \times \dots \times 2}_{n \text{ 个}} = 2^n. \quad (3.6)$$

若希望这些结果向量能够区分 L 种候选配置，那么不同配置至少需要被映射到足够多的结果模式中。因此，结果模式数量不能少于配置数量，即

$$2^n \geq L. \quad (3.7)$$

在本文讨论的配置空间中，三类信息源各有三种接入策略，所以 $L = 3^3 = 27$ 。代入上式可得

$$n \geq \log_2 27 \approx 4.75. \quad (3.8)$$

由于探针定理数量必须是整数，因此 $n \geq 5$ 。 □

当然并不意味着 5 个定理就能区分配置。真实实验中，单个定理的结果会受到模型随机性、检索噪声和证明搜索路径的影响。因此实际选择时会取更大的 k ，例如 7 个左右的探针定理，用冗余数量换取更稳定的排序结果。

3.2 定理特征与信息量

选择探针时，定理难度只是标准之一。我们更关心三个问题：它能不能体现不同配置的差距；它和已经选出的定理是不是在重复同一种模型能力；以及评测它要花多少 token。这三点分别对应区分能力、冗余度和预算。

区分能力通过小规模预评测来估计。设 $\mathcal{M}' \subseteq \mathcal{M}, \mathcal{C}' \subseteq \mathcal{C}$ 分别表示用于预评测的模型子集和配置子集。它们不需要覆盖全部模型和配置，只需要包含足够有代表性的组合。例如，可以选取一个能力较强的模型和一个能力较弱的模型，再选取若干差异明显的配置，如三类信息都采用 *always* 的配置，以及三类信息都采用 *once* 的配置。对每个定理 t ，定义它的行为向量为

$$\text{behavior}(t) = \{f(t, m, c)\}_{(m, c) \in \mathcal{M}' \times \mathcal{C}'} \quad (3.9)$$

这个向量记录了定理 t 在不同模型和配置下的响应模式。若一个定理在所有组合下都成功，或在所有组合下都失败，它可以反映绝对难度，却很难帮助比较工具接入策略。更有价值的是那些在不同配置下表现差异较大的定理，因为它们会直接暴露配置之间的差异。

基于行为向量，我们定义定理 t 的区分能力为

$$d_t = \text{Var}(\text{behavior}(t)). \quad (3.10)$$

这里用方差作为度量。方差越大，说明该定理在不同组合下的表现差异越明显。如果一个定理在某些组合下表现良好，在另一些组合下表现较差，那么它就更有可能是区分不同配置的探针。相反，如果一个定理在所有组合下的表现都很接近，那么它就不太适合作为探针，因为它无法提供足够的信息来区分配置。

仅有区分能力还不够。若多个定理在证明结构、依赖引理和难度上都很接近，那么即使它们各自的方差较高，放在同一个探针集合中也可能只是重复确认同一种现象。为此，为每个定理构造特征向量

$$\varphi(t) = [\delta(t), \text{behavior}(t)]. \quad (3.11)$$

其中 $\delta(t)$ 表示静态难度特征，可以由证明步数、证明深度、依赖引理数量、目标中符号数量、上下文大小等指标构成； $\text{behavior}(t)$ 则来自预评测结果。基于这个特征向量，定义两个定理之间的相似度为

$$\text{sim}(t_i, t_j) = \text{similarity}(\varphi(t_i), \varphi(t_j)). \quad (3.12)$$

实现时可以使用余弦相似度，也可以使用基于欧氏距离的核函数。相似度越高，说明两个定理越可能提供重复信号，选择时就应降低它们同时进入探针集合的倾向。

还需要考虑评测成本。设

$$c_t = \text{token cost of theorem } t. \quad (3.13)$$

这个成本可以包括输入提示词中的上下文 token、模型生成证明策略的 token、检索到的定义和引理 token，以及失败重试带来的额外开销。实际系统中， c_t 可以直接从预评测运行中估计，也可以由定理文本长度、上下文规模和历史平均调用次数近似得到。预算约束写作

$$\sum_{t \in S} c_t \leq B, \quad (3.14)$$

其中 B 是允许用于探针评测的总成本。

3.3 子集选择目标

把区分能力、冗余度和成本合在一起，探针选择可以写成如下优化问题：

$$\max_{S \subseteq \mathcal{T}} f(S) \quad \text{s.t.} \quad \sum_{t \in S} c_t \leq B, \quad |S| = k. \quad (3.15)$$

其中 k 是希望选出的探针数量， B 是 token 预算。目标函数取为

$$f(S) = \sum_{t \in S} d_t - \lambda \sum_{(t_i, t_j) \subseteq S, t_i \neq t_j} \text{sim}(t_i, t_j). \quad (3.16)$$

第一项奖励区分能力高的定理，第二项惩罚集合内部的相似样例。参数 $\lambda \geq 0$ 控制两者的权衡： λ 较小时，选择结果更偏向高方差定理； λ 较大时，算法会更主动地避开相似定理，探针集合的覆盖面也会更宽。

这个目标函数背后的直觉很简单：探针要能区分配置，但不能只是同一种困难模式的重复堆叠。在 ReCENT-PROVER 的工具接入问题中，不同配置可能擅长处理不同类型的证明。例如，频繁获取环境定义可能更有利于涉及复杂局部定义的目标；相关引理检索更适合依赖库定理的目标；相关证明检索则可能帮助那些具有可迁移证明模式的目标。探针集合需要覆盖这些不同情形，才更可能稳定反映配置之间的真实差异。

从优化角度看，这是一个带基数约束和预算约束的子集选择问题，与带背包约束的子模函数（Submodular Function）最大化问题密切相关。对于非负、单调非减的子模函数，已有工作证明，在一个背包约束下结合常数规模枚举和按单位成本边际收益选择的贪心过程，可以得到 $(1 - \frac{1}{e})$ 近似算法^[30]。本文不复现该算法的全部枚举步骤，而是借用其中最适合本场景的部分：利用边际收益递减的结构，用密度贪心快速得到一个可解释的候选集合。

定理 3.3.1 (探针选择目标的子模性) 假设对任意两个不同定理 t_i, t_j ，相似度满足 $\text{sim}(t_i, t_j) \geq 0$ 。定义

$$f(S) = \sum_{t \in S} d_t - \lambda \sum_{(t_i, t_j) \subseteq S, t_i \neq t_j} \text{sim}(t_i, t_j), \quad \lambda \geq 0. \quad (3.17)$$

则 f 是子模函数。也就是说，对任意 $A \subseteq B \subseteq \mathcal{T}$ 和 $x \in \mathcal{T} \setminus B$ ，都有

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B). \quad (3.18)$$

证明. 对集合 S 加入新定理 x 时，边际收益为

$$\Delta(x \mid S) = f(S \cup \{x\}) - f(S) = d_x - \lambda \sum_{s \in S} \text{sim}(x, s). \quad (3.19)$$

若 $A \subseteq B$ ，则

$$\sum_{s \in A} \text{sim}(x, s) \leq \sum_{s \in B} \text{sim}(x, s), \quad (3.20)$$

因为相似度非负。因此

$$d_x - \lambda \sum_{s \in A} \text{sim}(x, s) \geq d_x - \lambda \sum_{s \in B} \text{sim}(x, s). \quad (3.21)$$

这正是子模函数的边际收益递减性质。 $\square$

由定理 3.3.1 可知，已选集合越大，新定理带来的额外收益越不会增加；如果它与已有探针相似，冗余惩罚会直接降低它的边际价值。当 λ 的取值使得所有可行选择中的边际收益保持非负时，目标函数还满足单调性，此时问题就落在单调子模背包最大化框架内。实际求解时，我们采用最直接的密度贪心策略：每一步选择当前单位成本边际收益最高的定理，

$$\frac{\Delta(t \mid S)}{c_t} = \frac{d_t - \lambda \sum_{s \in S} \text{sim}(t, s)}{c_t}, \quad (3.22)$$

这将优先加入当前最有信息、且与已选集合最不重复的定理。在本文场景中，候选定理数量远大于最终探针数量，完整枚举所有满足 $|S| = k$ 且 $\sum_{t \in S} c_t \leq B$ 的子集代价过高。密度贪心虽然不承诺得到全局最优解，但每一步的选择依据明确，只依赖当前边际收益和成本，适合在预评测结果更新后快速重新运行。

3.4 具体算法

探针选择算法的整体流程如图 3.2 所示。

算法 3.1 Greedy Selection with Budget

- 1: **procedure** GREEDY-SELECT(A, S, R, h, λ)
 - 2: **Input:** candidates A ; selected probes S ; remaining budget R ; target size h ;
 redundancy penalty λ
 - 3: **Output:** updated selected probes S and remaining budget R
 - 4:
 - 5: **while** $|S| < h$ **do**
-

```

6:    $A_R \leftarrow \{t \in A \setminus S : c_t \leq R\}$ 
7:   if  $A_R = \emptyset$  then
8:     BREAK
9:   end
10:   $t^* \leftarrow \arg \max_{t \in A_R} \frac{d_t - \lambda \sum_{s \in S} \text{sim}(t, s)}{c_t}$ 
11:   $\Delta^* \leftarrow d_{t^*} - \lambda \sum_{s \in S} \text{sim}(t^*, s)$ 
12:  if  $\Delta^* \leq 0$  then
13:    BREAK
14:  end
15:   $S \leftarrow S \cup \{t^*\}$ 
16:   $R \leftarrow R - c_{t^*}$ 
17: end
18: return  $S, R$ 
19: end
    
```

图 3.1 带预算约束的辅助贪心选择算法

算法 3.2 Probe Theorem Selection

```

1: procedure PROBE-SELECT( $\mathcal{T}, \mathcal{M}', \mathcal{C}', k, B, \lambda$ )
2:   Input: theorem set  $\mathcal{T}$ ; profiling models  $\mathcal{M}'$ ; profiling configs  $\mathcal{C}'$ ; probe size  $k$ ;
   budget  $B$ ; redundancy penalty  $\lambda$ 
3:   Output: selected probe set  $S$ 
4:
5:    $\mathcal{T}_{\text{cand}} \leftarrow \text{StratifiedSample}(\mathcal{T})$ 
6:   for  $t \in \mathcal{T}_{\text{cand}}$  do
7:      $\delta(t) \leftarrow \text{StaticFeatures}(t)$ 
8:   end
9:
10:  for  $t \in \mathcal{T}_{\text{cand}}$  do
11:    for  $(m, c) \in \mathcal{M}' \times \mathcal{C}'$  do
12:       $f(t, m, c) \leftarrow \text{Evaluate}(t, m, c)$ 
13:    end
14:     $\text{behavior}(t) \leftarrow \{f(t, m, c)\}_{(m, c) \in \mathcal{M}' \times \mathcal{C}'}$ 
15:     $d_t \leftarrow \text{Var}(\text{behavior}(t))$ 
16:     $\varphi(t) \leftarrow [\delta(t), \text{behavior}(t)]$ 
17:     $c_t \leftarrow \text{EstimateCost}(t)$ 
18:  end
19:  for  $t_i, t_j \in \mathcal{T}_{\text{cand}}$  do
20:     $\text{sim}(t_i, t_j) \leftarrow \text{Similarity}(\varphi(t_i), \varphi(t_j))$ 
21:  end
22:
23:   $\mathcal{L} \leftarrow \text{PartitionByDifficulty}(\mathcal{T}_{\text{cand}})$ 
24:  for  $L \in \mathcal{L}$  do
25:     $q_L \leftarrow \text{LayerQuota}(L, k)$ 
26:  end
27:   $S \leftarrow \emptyset$ 
28:   $R \leftarrow B$ 
29:
30:  for  $L \in \mathcal{L}$  do
31:     $h_L \leftarrow |S| + q_L - |S \cap L|$ 
32:     $S, R \leftarrow \text{Greedy-Select}(L, S, R, h_L, \lambda)$ 
    
```

```

33:   end
34:
35:    $S, R \leftarrow \text{Greedy-Select}(\mathcal{T}_{\text{cand}}, S, R, k, \lambda)$ 
36:   return  $S$ 
37: end

```

图 3.2 探针选择算法

在图 3.1 中，Greedy-Select 的参数 A 表示当前可选定理集合， S 表示已经选出的探针集合， R 表示剩余预算， h 表示本轮选择结束时希望达到的集合大小， λ 表示冗余惩罚系数；该过程返回更新后的 S 和 R 。在图 3.2 中，Probe-Select 的参数 $\mathcal{T}$ 表示完整定理集合， $\mathcal{M}'$ 和 $\mathcal{C}'$ 分别表示预评测使用的模型子集和配置子集， k 表示希望选出的探针数量， B 表示总 token 预算， λ 与辅助过程含义相同；该过程返回最终的探针定理集合 S 。

其他辅助函数含义如下。StratifiedSample 从完整定理集合中分层抽取候选定理；StaticFeatures 提取定理长度、上下文规模、证明结构等静态难度特征；Evaluate 运行一次自动证明评测并返回结果；EstimateCost 估计评测某个定理所需的 token 成本；Similarity 计算两个定理特征向量之间的相似度；PartitionByDifficulty 按照难度把候选定理划分为若干层；LayerQuota 根据总探针数量 k 为每个难度层分配最低选择数量。

算法可以按下面几个阶段理解。

先构造候选定理集合。完整的 CoqStoq 定理集合规模较大，在全部定理上计算相似度和预评测结果成本较高。实际实现中可以先采样一个中等规模的候选集合 $\mathcal{T}_{\text{cand}}$ ，例如 100 个定理。采样时要避免只抽到某一种难度或某一个库中的样例，因此可以根据文件来源、定理长度或初步难度分数进行分层采样。

然后计算静态难度特征 $\delta(t)$ 。这一步不依赖模型运行结果，可以直接从定理本身和已有证明中提取。若数据集中包含人工证明或参考证明，可以统计证明步数、证明深度、使用引理数量，以及证明中是否出现归纳、改写、自动化策略等结构；若没有参考证明，也可以使用目标长度、上下文假设数量、涉及的常量数量等近似特征。这些特征经过归一化后组成 $\delta(t)$ 。

接下来进行小规模预评测。对每个候选定理 $t \in \mathcal{T}_{\text{cand}}$ ，只在较小的模型集合 $\mathcal{M}'$ 和配置集合 $\mathcal{C}'$ 上运行证明系统，得到 $f(t, m, c)$ 。例如，可以取

$$\mathcal{M}' = \{\text{OpenAI o4-mini, MiniMax M2.5}\}, \quad (3.23)$$

并选择两个极端配置：

$$\mathcal{C}' = \{(\text{always}, \text{always}, \text{always}), (\text{once}, \text{once}, \text{once})\}. \quad (3.24)$$

这里并不追求完整评估所有配置，只希望用较低成本观察一个定理是否对模型能力和工具接入方式敏感。

预评测完成后，计算每个定理的信息量、相似度和成本。根据预评测结果得到 $\text{behavior}(t)$ ，再计算方差 d_t 。同时，将 $\delta(t)$ 与 $\text{behavior}(t)$ 拼接为 $\varphi(t)$ ，并计算候选定理两

两之间的相似度 $\text{sim}(t_i, t_j)$ 。成本 c_t 可以直接使用预评测中的实际 token 消耗，也可以用平均 token 消耗估计。

如果候选集合的难度分布很不均衡，还可以加入分层处理。具体来说，算法先按照难度将候选定理划分为 *easy*、*medium* 和 *hard* 三层，并为每一层设置最低选择数量 q_L 。贪心选择分两步进行：第一步在各个难度层内部按单位成本边际收益选择定理，尽量满足每层的最低名额；第二步再回到全局候选集合中继续贪心选择，直到达到探针数量 k 或预算耗尽。这样做的目的，是避免探针集合被某一类定理主导，同时仍然保留按信息量和成本排序的选择原则。

最后进行贪心选择。初始化 $S = \emptyset$ ，剩余预算为 B 。每一轮只考虑尚未选择且成本不超过剩余预算的定理 t ，并计算其边际收益：

$$\Delta(t \mid S) = d_t - \lambda \sum_{s \in S} \text{sim}(t, s). \quad (3.25)$$

为了同时考虑成本，排序时使用单位成本收益 $\frac{\Delta(t \mid S)}{c_t}$ 。算法把单位成本收益最高的定理加入 S ，并扣除相应成本。这个过程持续到 $|S| = k$ ，或者没有任何候选定理能在剩余预算内带来正收益为止。最终得到的 S 就是后续配置评测使用的探针集合。

3.5 面向模型升级的使用流程

前面的形式化描述把模型集合 $\mathcal{M}$ 和配置集合 $\mathcal{C}$ 同时作为输入，这是为了强调本文方法并不局限于在一个固定模型上做配置调参。在实际系统中，更常见的情况是：已有证明系统已经在某个后端模型上调试过，但随后出现了新的模型版本、不同供应商模型或价格更合适的模型。此时研究者真正关心的是快速判断原有 agent 工具接入策略是否仍然合适，以及哪些配置值得优先进入完整评测，而非重新设计整个证明系统。

因此，本文方法可以按照两个阶段使用。第一阶段是离线构建相对稳定的探针集合。系统先从目标数据集中抽取候选定理，利用已有模型和少量代表性配置完成预评测，计算定理的行为向量、区分能力和相似度。这个阶段的成本高于一次普通配置比较，但它不需要在每次模型更新时完整重做。只要证明任务分布没有发生根本变化，探针集合就可以作为后续模型适配的公共评测入口。换言之，探针集合本身更接近一个小型诊断集，不只是服务于某一次实验的临时样本。

第二阶段是模型接入后的快速筛选。当一个新模型 m_{new} 被接入证明系统时，系统只需要在探针集合 S 上运行若干候选配置 $\mathcal{C}_{\text{cand}}$ ，得到 $F(m_{\text{new}}, c; S)$ 和相应 token 成本。随后使用与完整实验一致的两阶段准则进行排序：先排除证明成功数明显不足的配置，再在成功数接近最优的配置中选择 token 效率更高者。这样做的好处是，新模型的适配实验规模由 $|\mathcal{T}| \times |\mathcal{C}|$ 降为 $|S| \times |\mathcal{C}_{\text{cand}}|$ 。当 $|S|$ 远小于完整数据集规模，并且 $\mathcal{C}_{\text{cand}}$ 只包含少量代表性配置时，适配成本会显著下降。

这一流程还允许逐步扩大评测范围。若探针集结果显示某个配置明显优于其他配置，可以直接把它和少数竞争配置放入验证集；若探针集结果不稳定，例如多个配置成功数相同但效率差异受单个定理影响较大，则可以临时扩大探针集合或增加若干边界样本。本文并不要求探针集一次性给出最终答案，而是把它作为完整评测之前的低成本过滤器。对于快速演进的 LLM 证明系统，这种分阶段流程比直接穷举所有配置更适合工程迭代。

探针集合也可以帮助解释模型差异。假设两个模型在同一探针集合上选择出不同配置，那么可以进一步检查这些探针定理的失败原因和上下文使用情况：一个模型是否在缺少环境定义时更容易生成未定义符号；另一个模型是否在引理过多时更容易选择错误引理；按需调用是否被频繁触发但没有带来证明状态推进。这样的分析虽然不一定能直接导出新的算法，但可以帮助研究者理解模型能力与 agent 工作流之间的匹配关系。

需要注意的是，探针集合的复用有前提条件。如果目标数据集从软件验证转向纯数学库，定理结构、可用引理和证明风格都可能发生明显变化，旧探针集合就未必能代表新任务。如果证明系统本身加入了新的工具，例如反例搜索器、归纳模式建议器或更强的 Hammer 后端，原有行为向量也可能不足以刻画新的配置差异。因此，在任务分布或工具集合发生较大变化时，仍然需要重新进行一次预评测，或者至少补充一批覆盖新工具行为的探针定理。本文方法降低的是常见模型替换和配置微调场景下的评测开销，不能完全消除配置评测需求。

3.6 实现细节

实现上，系统分成预评测、特征计算和探针选择三个模块。预评测模块负责调用自动证明系统，并记录每个 (t, m, c) 组合的证明结果、token 消耗、运行时间和失败原因。特征计算模块把预评测结果和定理静态信息整理成 d_t 、 $\delta(t)$ 、 $\varphi(t)$ 、 c_t 以及相似度矩阵。探针选择模块读取这些中间结果，根据目标函数和预算约束运行贪心算法，输出最终的定理编号列表。

为了保证结果可复现，所有随机采样都固定随机种子，并保存候选集合、预评测配置和最终探针集合。对于失败证明，系统不仅记录二值失败结果，还保留失败发生时的最后子目标、已尝试策略和工具调用历史。这些信息虽然不直接进入当前目标函数，但有助于后续解释某些探针定理为什么能够区分配置。

配置编码采用显式三元组。例如，(always, on demand, once) 表示环境定义在每轮都获取，相关引理由系统按需获取，相关证明只获取一次。这样的编码方式便于追踪配置差异，也方便在后续实验中分析单个工具策略变化对证明成功率的影响。

需要注意的是，探针集合上的最优配置不一定就是完整数据集上的最优配置。它的作用是降低早期探索成本，帮助我们更快筛掉明显较差的设计，并识别值得进入完整评测的候选配置。

4 实验评估

本章验证第 3 章中提出的探针选择方法是否能够作为模型能力的低成本代理 (Proxy)，用于判断一个后端模型更适合哪一类工具接入配置。本文关注的核心不是重新给出 RECENT-PROVER^[17] 的完整性能排名，也不只是单纯寻找某个全局最优配置；真正的问题是：模型能力本身很难直接定量描述，我们无法只根据模型名称或通用基准分数判断它在证明任务中应该如何调用环境定义、相关引理和证明示例。因此，本文使用一小组探针定理来观察模型在不同配置下的行为，并用这种行为近似刻画“该模型适合哪种 agent 工具调用方式”。

换言之，探针集在本章中承担的是模型能力测量工具的角色。若一个模型在探针集上只有在频繁更新相关引理时才表现良好，那么这说明它可能更依赖动态检索；若另一个模型在探针集上更适合稳定的定义上下文和较少示例刷新，那么它的 agent 设计也应相应调整。后续验证集的作用，是检查这种由探针集得到的模型能力信号，能否迁移到未参与探针选择的定理上。

本章围绕以下三个问题展开：

- **问题 1：** 探针定理能否作为模型能力的有效代理？也就是说，探针集上观察到的模型-配置匹配关系，能在多大程度上迁移到独立验证集。
- **问题 2：** 用探针估计模型适配配置时，如何同时考虑证明能力和调用成本？本文不把低消耗但几乎不能证明定理的配置视为有效配置，因此采用先比较成功数、再比较效率的两阶段准则。
- **问题 3：** 不同后端模型是否需要不同的 agent 设计？如果不同模型在同一候选空间中选出不同配置，就说明 agent 设计需要围绕模型能力重新适配。

4.1 实验设置

实验在 Rocq 8.18 环境下进行，数据集采用 CoqStoq^[18]。CoqStoq 由真实 Rocq 项目中的定理组成，覆盖编译器验证、形式化数学等不同类型的证明任务，因此适合用来观察配置变化对自动证明过程的影响。证明系统同时使用 CoqHammer 和 LLM 生成证明策略：每轮证明先调用 CoqHammer，超时时间设为 25 秒，最多使用 64 个线程；如果 CoqHammer 未能直接完成当前子目标，再调用后端语言模型生成证明策略。每个定理的证明搜索最多进行 25 轮。

检索数据库由 CoqStoq 中的定理和已有证明离线构建。构建数据库时，系统为引理和证明示例生成自然语言描述或证明计划，并使用 OpenAI 的 text-embedding-3-large 模型^[31] 计算向量表示。在线证明时，系统不会从整个数据库中任意取用信息，而是先把候选范围限制在当前证明库中已经可见的引理和证明上，再从中检索 8 个相关引理和 8 个相关证明示

例。这样可以避免把当前目标本来无法使用的信息带入提示词。若最终提示词超过后端模型的长度限制，则从左侧裁剪较早的上下文，保留更靠近当前证明状态的信息。

本文关注三类工具接入策略：环境定义（definitions）、相关引理（lemmas）和相关证明示例（examples）。在配置名称中， d 、 l 、 e 分别表示这三类信息源；后缀 A 、 O 、 D 分别表示 *always*、*once* 和 *on demand*。例如，配置 $d0_lA_eA$ 表示环境定义只在开始时获取一次，相关引理和相关证明示例则在每轮证明中都获取。

为了检验探针集合的预测作用，我们对每个后端模型随机选择 10 个配置，并分别在两个集合上评测：

- 探针集（Probe）：包含 7 个由第 3.4 节方法选出的探针定理；
- 验证集（Validation）：包含 15 个未参与探针选择的定理。

实验分别在 MiniMax M2.5^[19]、OpenAI o4-mini^[20]、Qwen3-235B-A22B^[21]和 GLM-5^[22]四个后端模型上进行。对每个配置，本文记录成功证明的定理数量和总 token 消耗。只看 token 效率是不够的：一个配置如果几乎证明不了定理，即使消耗很低，也不能算作好的证明配置。因此，本文采用两阶段准则。首先，在同一个评测集合上找出最高成功数 BestSolved，只保留成功数不少于 BestSolved − 1 的配置；也就是说，候选配置的成功数至少要当前最优配置只相差一个定理。然后，才在这些候选配置中比较每百万 token 成功数：

$$\text{Efficiency} = \frac{\#Solved}{\#Tokens} \times 10^6. \quad (4.1)$$

4.2 实验结果

4.2.1 MiniMax M2.5 结果

配置	Probe	Probe Tok.	Probe Eff.	Val.	Val. Tok.	Val. Eff.
dA_lA_eA	2/7	1.335M	1.50	7/15	2.723M	2.57
dA_lA_eD	2/7	4.922M	0.41	7/15	6.628M	1.06
dA_lO_eO	3/7	1.187M	2.53	8/15	3.229M	2.48
dA_lO_eD	3/7	3.855M	0.78	7/15	8.664M	0.81
dA_lD_eO	2/7	6.668M	0.30	7/15	9.848M	0.71
d0_lA_eA	3/7	0.912M	3.29	7/15	2.647M	2.64
d0_lA_eD	4/7	3.457M	1.16	7/15	6.971M	1.00
d0_lO_eD	2/7	4.012M	0.50	7/15	7.520M	0.93
dD_lA_eA	2/7	5.879M	0.34	7/15	8.558M	0.82
dD_lA_eO	2/7	4.612M	0.43	7/15	7.356M	0.95

表 4.1 MiniMax M2.5 后端下随机 10 个配置的探针集与验证集结果

如表 4.1 所示，在 MiniMax M2.5 后端下，探针集上的最高成功数为 4/7，验证集上的最高成功数为 8/15。按照两阶段准则，探针集和验证集最终都选择 $d0_lA_eA$ 。

4.2.2 OpenAI o4-mini 结果

配置	Probe	Probe Tok.	Probe Eff.	Val.	Val. Tok.	Val. Eff.
dA_1A_e0	3/7	0.680M	4.41	7/15	1.902M	3.68
dA_1A_eD	3/7	0.608M	4.94	6/15	1.620M	3.70
dA_10_e0	4/7	0.827M	4.84	8/15	2.301M	3.48
dA_10_eD	3/7	0.640M	4.69	7/15	1.913M	3.66
dA_1D_eA	2/7	1.277M	1.57	6/15	2.679M	2.24
dA_1D_eD	3/7	1.288M	2.33	7/15	2.294M	3.05
d0_1A_eA	5/7	0.464M	10.78	9/15	1.684M	5.34
d0_1A_eD	3/7	0.521M	5.76	6/15	1.906M	3.15
d0_10_e0	4/7	1.255M	3.19	7/15	3.255M	2.15
d0_1D_eA	2/7	1.460M	1.37	5/15	2.755M	1.82

表 4.2 OpenAI o4-mini 后端下随机 10 个配置的探针集与验证集结果

如表 4.2 所示，在 OpenAI o4-mini 后端下，d0_1A_eA 在探针集上达到 5/7 的最高成功数，并以 10.78 的效率排名第一；在验证集上也达到 9/15 的最高成功数，并以 5.34 的效率排名第一。

4.2.3 Qwen3-235B-A22B 结果

配置	Probe	Probe Tok.	Probe Eff.	Val.	Val. Tok.	Val. Eff.
dA_1A_eA	4/7	0.397M	10.08	6/15	0.623M	9.63
dA_10_eA	3/7	0.401M	7.48	6/15	0.719M	8.34
dA_10_e0	4/7	0.425M	9.41	6/15	1.506M	3.99
dA_1D_e0	2/7	0.461M	4.34	7/15	0.994M	7.04
d0_10_e0	3/7	0.221M	13.57	6/15	1.247M	4.81
d0_1D_eA	2/7	0.306M	6.54	6/15	0.704M	8.52
d0_1D_eD	3/7	0.191M	15.67	6/15	0.470M	12.75
dD_10_eA	1/7	0.552M	1.81	6/15	0.909M	6.60
dD_10_eD	3/7	0.334M	8.98	6/15	0.540M	11.11
dD_1D_e0	2/7	0.743M	2.69	6/15	1.125M	5.34

表 4.3 Qwen3-235B-A22B 后端下随机 10 个配置的探针集与验证集结果

Qwen3-235B-A22B 上的最优配置不同于 MiniMax M2.5 和 OpenAI o4-mini。由表 4.3 可见，探针集上的最高成功数为 4/7；在满足 BestSolved - 1 的候选配置中，d0_1D_eD 以 15.67 的效率被选出。验证集上的最高成功数为 7/15，由 dA_1D_e0 达到；按两阶段准则保留成功数不低于 6/15 的配置后，d0_1D_eD 以 12.75 的效率排名第一。因此，探针集和验证集在 Qwen3-235B-A22B 上也选择了相同配置。

4.2.4 GLM-5 结果

配置	Probe	Probe Tok.	Probe Eff.	Val.	Val. Tok.	Val. Eff.
dA_1A_eA	5/7	0.268M	18.64	6/15	0.347M	17.31
dA_10_e0	4/7	0.297M	13.48	7/15	0.462M	15.14
dA_1D_e0	3/7	1.198M	2.50	6/15	1.122M	5.35
dA_1D_eD	3/7	1.535M	1.95	5/15	1.076M	4.65
d0_1A_e0	4/7	0.263M	15.23	6/15	0.371M	16.18
d0_1D_eD	4/7	0.683M	5.85	5/15	1.022M	4.89
dD_1A_eD	3/7	0.769M	3.90	7/15	0.539M	13.00
dD_10_eA	4/7	0.529M	7.57	6/15	0.731M	8.21
dD_10_eD	3/7	0.879M	3.41	7/15	0.899M	7.78
dD_1D_e0	2/7	3.642M	0.55	6/15	1.685M	3.56

表 4.4 GLM-5 后端下随机 10 个配置的探针集与验证集结果

如表 4.4 所示，在 GLM-5 后端下，探针集上的最高成功数为 5/7，由 **dA_1A_eA** 达到；验证集上的最高成功数为 7/15，由 **dA_10_e0**、**dD_1A_eD** 和 **dD_10_eD** 达到。按照两阶段准则，探针集和验证集最终都选择 **dA_1A_eA**。该配置虽然在验证集上没有达到最高成功数，但仍满足成功数不低于 6/15 的候选条件，并以 17.31 的效率在候选配置中排名第一。

4.3 问题 1：探针能否作为模型能力代理

根据上面的实验结果，探针集上观察到的模型-配置匹配关系能够迁移到验证集。

MiniMax M2.5、OpenAI o4-mini、Qwen3-235B-A22B 和 GLM-5 的探针选择都能在验证集上保持一致。探针集因此可以看作一种低成本近似：它用少量定理测量模型在当前证明任务中适合哪种工具调用方式。

具体来说，MiniMax-M2.5 上，探针集显示该模型更适合 **d0_1A_eA**，该配置在验证集上也以 2.64 的效率排名第一；OpenAI o4-mini 上，探针集显示 **d0_1A_eA** 最适合该模型，验证集上也以 5.34 的效率排名第一。Qwen3-235B-A22B 上，探针集和验证集都选择 **d0_1D_eD**；GLM-5 上，二者都选择 **dA_1A_eA**。这说明探针集在当前实验中不仅保留了高效候选信号，也能较好预测两阶段准则下的最终配置。

这个结果支持本文的核心假设：模型能力很难直接定量描述，但可以通过探针定理上的行为间接观察。探针不能替代完整评测，但可以作为模型能力与工具配置匹配关系的低成本代理。

4.4 问题 2：如何用探针同时衡量能力和成本

第一阶段用成功数判断模型在某个配置下是否具有足够证明能力；第二阶段再用 token 效率比较这些可行配置的调用成本。这样可以避免仅因某个配置消耗较低，就忽略它证明能力不足的问题。

MiniMax M2.5 的结果最能说明这一点。探针集上 d0_1A_eD 成功 4/7，高于 d0_1A_eA 的 3/7，但它消耗 3.457M token，效率只有 1.16；d0_1A_eA 虽然少证明一个探针定理，但仍满足 BestSolved - 1 的成功数要求，并以 3.29 的效率成为探针集选择。验证集上也类似：成功数最高的配置达到 8/15，d0_1A_eA 达到 7/15，仍在候选范围内，并以 2.64 的效率排名第一。

OpenAI o4-mini 上，最终选择的 d0_1A_eA 同时达到最高成功数和最高效率，说明两阶段准则没有牺牲成功数。Qwen3-235B-A22B 上，两阶段准则的作用更明显：验证集上 dA_1D_eD 达到最高成功数 7/15，但 token 效率只有 7.04；d0_1D_eD 虽然成功数为 6/15，仍满足 BestSolved - 1 的候选条件，并以 12.75 的效率成为最终选择。综合来看，本文采用的两阶段准则可以同时反映模型的证明能力和工具调用成本。

4.5 问题 3：不同模型是否表现出不同工具调用偏好

当前实验显示，不同模型确实表现出不同工具调用偏好。MiniMax M2.5 和 OpenAI o4-mini 都选择 d0_1A_eA，即只在开始阶段获取环境定义，并在每轮证明中更新相关引理和证明示例。Qwen3-235B-A22B 选择 d0_1D_eD，即只在开始阶段获取环境定义，并按需获取相关引理和证明示例。GLM-5 则选择 dA_1A_eA，更倾向于在每轮证明中持续注入三类上下文。

这说明工具调用策略不能简单看作模型无关的固定参数。MiniMax M2.5 和 OpenAI o4-mini 可能更能从随证明状态变化的引理和证明示例中获益，因此频繁更新检索内容能提高单位 token 收益；Qwen3-235B-A22B 在当前样本上则更倾向于减少无条件注入的证明示例，通过按需调用控制上下文规模；GLM-5 在当前候选集合中更适合持续提供完整上下文。本文不能从这些结果直接推断模型内部机制，也不把单次实验解释为稳定偏好；但可以得到一个工程结论：当后端模型升级或替换时，应当重新用探针检测模型能力与配置的匹配关系，不能默认沿用旧模型上的 agent 设计。

4.6 实验局限

本章实验仍然是初步验证。首先，每个模型只随机采样 10 个配置，尚未覆盖完整的 $3^3 = 27$ 种工具接入方式，因此当前结果只能说明探针集在这批候选配置中保留了有用的排序信息，还不能视为已经找到全局最优配置。如果未采样的配置在完整空间中表现更好，当前实验无法发现这一点。其次，探针集和验证集分别只有 7 个和 15 个定理，规模较小，容易受到单个定理和模型随机性的影响；更稳健的结论需要更多定理、更多随机种子和重复运行。尤其是在探针集较小时，一个定理的成功或失败就可能改变两阶段准则下的候选范围。

再次，本文主要用成功数和 token 消耗衡量配置优劣，尚未纳入运行时间、失败类型、证明脚本质量等指标。实际系统中，一个配置即使 token 效率较高，也可能因为运行时间长、失败难以诊断或生成脚本不稳定而不适合使用。最后，实验说明了模型间配置偏好可能不同，但这种差异是否稳定，还需要在更大的 CoqStoq 样本和更多后端模型上继续验证。

因此，本章结果更适合作为探针式筛选方法的可行性证据，不能视为对具体模型或具体配置的最终排序结论。

5 相关工作

5.1 基于 LLM 的自动定理证明

大语言模型被用于自动定理证明后，形式化证明生成逐渐从单步策略生成转向更复杂的证明智能体。早期神经证明方法主要关注如何从当前目标直接生成证明脚本或证明策略；随着模型能力增强，后续工作开始引入检索、搜索、证明助手反馈和失败历史等机制，使 LLM 参与到多轮交互组成的证明过程之中。

在 Rocq 和 Lean 等证明助手中，LLM 生成的证明步骤必须经过内核检查才能被接受。因此，LLM 证明系统通常会先由模型提出候选策略，再由证明助手检查其语法和语义效果，并根据检查结果继续搜索。Draft, Sketch, and Prove^[32]利用非形式化证明草稿引导形式化证明器，说明自然语言层面的证明计划可以为形式化搜索提供方向。LeanDojo^[33]构建了面向 Lean 的检索增强证明环境，使模型能够在交互过程中获取相关引理和上下文。Rango^[18]将检索增强生成用于自动软件验证，通过检索相关上下文来帮助模型生成证明。Cobblestone^[29]则采用分治式思路，把较复杂的验证任务拆分为更适合自动化处理的子任务。

这些系统的共同点是，它们都不再把 LLM 看作孤立的文本生成器，而是把它放入一个有外部反馈的证明循环中。证明助手提供强约束的验证反馈，检索模块提供局部可用的知识，已有证明示例提供可迁移的策略模式，符号自动化工具则负责处理一部分局部目标。RECENT-PROVER^[17]进一步把 CoqHammer、规划驱动检索、LLM 证明生成和自反思验证组织在同一证明流程中，使模型既可以生成策略，也可以参与证明计划和错误分析。

这些工作表明，LLM 自动定理证明的关键已经扩展到系统组织层面：除了模型能否生成正确策略，还要看模型、检索器、符号工具和证明助手之间如何交互。本文的研究对象也位于这一背景之下。不同于上述工作主要关注提升证明系统本身的成功率，本文关注一个更具体的配置问题：当证明系统已经包含环境定义、相关引理、相关证明示例和 Hammer 等工具时，如何以较低成本选择这些工具的接入方式和调用频率。

这一问题在形式化证明场景中尤其突出。首先，证明上下文通常较长，完整地提供所有定义、引理和证明示例会迅速消耗上下文窗口。其次，不同证明目标对信息源的依赖并不相同：有的目标需要展开局部定义，有的目标更依赖库中引理，有的目标则更适合参考已有证明结构。最后，后端模型能力不同，同一上下文策略在不同模型上可能产生不同效果。因此，工具接入配置本身成为 LLM 证明系统的重要设计变量。

5.2 LLM 智能体工作流优化

LLM 智能体通常由多个计算组件组成，而不只是一次模型调用。近期关于 LLM 智能体工作流优化的综述将这类系统抽象为智能体计算图（Agentic Computation Graph），其中节点可以是 LLM 调用、信息检索、工具使用、代码执行、记忆更新和验证模块，边则描述信

息在这些组件之间的流动^[34]。该综述按照 workflow 结构在何时被确定来组织已有方法：有些系统在运行前使用固定模板，有些系统在运行时动态选择下一步工具或子任务，还有一些系统会根据反馈不断修改执行图。

LLM 智能体 workflow 优化的一个基本动机是，固定提示词模板很难适应复杂任务。ReAct^[35]通过交替生成推理轨迹和行动，把语言推理与外部工具调用结合起来；Reflexion^[36]则让智能体根据失败经验生成语言反馈，并在后续尝试中利用这些反馈改进行为。这类工作虽然不直接面向形式化证明，但它们强调了同一点：当任务需要多步推理和外部环境反馈时，系统性能会受到单次模型调用质量、调用顺序、反馈利用方式和工具选择策略的共同影响。

从这个角度看，LLM 自动定理证明系统本身就是一种领域特定的智能体 workflow。一次证明尝试可能包含 CoqHammer 调用、检索、LLM 策略生成、Rocq 检查、反思验证和失败记录更新。不同模块的顺序、触发条件和上下文传递方式都会影响最终证明能力和 token 成本。例如，相关引理可以在每一轮都检索，也可以只在开始时检索；证明示例可以总是加入提示词，也可以仅在模型需要时加入；环境定义可以在证明开始时统一获取，也可以随着子目标变化重新查询。这些设计并不会改变证明助手的逻辑内核，却会显著改变 LLM 看到的上下文和系统消耗的资源。

本文可以看作 LLM 智能体 workflow 优化在自动定理证明场景中的一个具体实例。与通用 workflow 优化不同，本文并不尝试自动生成完整的执行图，而是聚焦于证明 workflow 中几类关键工具的接入策略。本文将 *always*、*once* 和 *on demand* 视为不同的 workflow 配置，并研究如何在完整评测代价较高时，用少量探针定理筛选出更有希望的配置。

与现有通用智能体 workflow 研究相比，本文的约束更强，也更具体。形式化证明系统的每一步都必须由证明助手检查，因此 workflow 优化不能只追求模型输出的自然语言质量，还要考虑证明状态是否真正推进、上下文中提供的信息是否在当前证明库中可用，以及失败重试带来的成本。本文提出的探针选择方法并不替代 workflow 优化算法，而是为这类优化提供一个低成本评测入口：先用少量能够区分配置的定理估计不同 workflow 配置的相对优劣，再决定哪些配置值得进入完整实验。

另一个区别在于反馈粒度。通用智能体 workflow 常常使用任务是否完成、最终回答是否正确或人工偏好作为反馈；而自动定理证明可以获得更细的执行反馈。每个候选策略都会被证明助手检查，失败时可以得到语法错误、未解决子目标、类型不匹配或策略无效等信息。这些反馈使 workflow 优化不必完全依赖黑盒成功率，也可以分析失败发生在证明搜索的哪个环节。本文当前只把这些信息作为辅助解释，没有把失败类型直接纳入目标函数；但从相关工作角度看，形式化证明为智能体 workflow 优化提供了一个更可验证、更可诊断的应用场景。

此外，通用 workflow 优化通常默认模型固定，然后优化执行图或工具选择；本文则把模型变化本身纳入问题定义。随着模型持续升级，同一个 workflow 配置可能在旧模型上有效、在新模型上低效。这个问题在实际部署中非常常见：模型供应商更新版本、上下文窗口扩大、价

格下降或工具调用接口变化，都会促使系统替换后端模型。本文的探针方法可以看作一种面向模型迁移的工作流再评估机制，重点是快速判断已有工作流参数是否需要调整，而非自动合成全新工作流。

5.3 面向模型能力的技能与工具适配

在 LLM 智能体中，“工具”和“技能”经常同时出现，但二者关注点并不相同。工具通常指外部 API、检索器、解释器、证明助手或符号求解器等可调用接口，核心问题是模型能否在合适的时机选择正确工具，并生成正确的调用参数。技能则更像可复用的任务解决单元，可能包含自然语言说明、示例、代码片段、工具调用模板和执行经验，核心问题是如何选择、压缩、组合或改写这些能力单元，使它们适配不同模型和任务。本文讨论的环境定义获取、相关引理检索、证明示例检索和 Hammer 调用同时具有这两种属性：它们是证明系统中的可调用工具，也可以被看作证明智能体在特定任务中使用的领域技能。

5.3.1 工具调用能力优化

工具调用优化主要关注模型如何学会使用外部接口。Toolformer^[37]研究模型如何通过自监督方式学习在文本生成过程中调用外部工具，使工具使用不完全依赖人工标注。Gorilla^[38]面向大规模 API 调用场景，使模型能够根据自然语言需求选择合适的 API 并生成调用参数。ToolLLM^[39]进一步构建大规模工具使用数据和评测环境，训练模型掌握大量真实世界 API。近期关于工具学习的综述^[40]也将相关研究归纳为工具感知、工具选择、工具调用、执行反馈和工具能力泛化等环节。这些工作已经形成较清晰的研究脉络：工具使用能力不是固定写在提示词中的规则，它可以通过数据构造、模型训练、API 检索和执行反馈持续优化。

除训练模型掌握工具接口外，近期工作也开始直接优化工具选择和工具上下文。TECTON^[41]通过元推理机制改进模型的工具使用决策，使模型在调用工具前显式考虑工具是否适合当前任务。Tool Utilization Joint Optimization^[42]同时优化工具文档选择和上下文组织，以提高 LLM 智能体使用工具的效率。SMART^[43]进一步关注工具过度使用问题，让智能体先判断当前任务是否真的需要工具，从而减少不必要的调用。RAG-MCP^[44]从检索增强角度缓解工具选择中的提示词膨胀问题，只把与当前任务相关的工具说明放入上下文。AutoTool^[45]则进一步把大规模工具集合中的工具选择作为效率问题处理，目标是在大量可用工具中快速筛选少量候选工具。相比早期侧重训练模型调用 API 的研究，这些工作更接近本文关注的场景，即在工具集合已经存在时，减少无效工具上下文，并让模型在更合适的时机使用更合适的工具。

这些工作与本文的关系在于，它们说明工具本身可用还不够。模型是否知道何时使用工具、如何使用工具，以及系统是否应该把某个工具暴露给模型，同样会影响任务表现。对于自动定理证明而言，CoqHammer、引理检索器和证明助手接口都可以看作工具；如果模型不能在合适的证明状态下使用它们，工具本身的能力就无法转化为证明成功率。本文不直接

训练模型的工具调用能力，也不优化自然语言 API 文档，而是假设证明系统中的工具接口已经可用，进一步研究这些工具在多轮证明 workflows 中应当以何种频率和时机接入。换句话说，上述工具优化工作更多关注工具选择、工具调用和工具上下文本身，本文更关注工具接入策略对整个证明流程的系统级影响。

5.3.2 技能选择与技能优化

技能优化则更关注可复用能力单元本身如何被管理。新近的 SkVM^[46]将技能视为可编译的程序单元，并把 LLM 看作具有不同能力画像的异构处理器。该工作先分析技能所需的基本能力，再根据模型和执行环境的能力画像进行编译、环境绑定、并发提取和运行时适配。其核心思想是，技能不应被静态地塞入上下文，而应根据模型能力和执行环境进行变换，使其在不同模型和平台上更稳定、更高效地运行。由于该工作较新，本文引用其预印本版本。

技能优化与工具优化的差别在于，前者通常把一段可复用的任务经验作为优化对象，后者则更多围绕外部接口的选择、说明和调用展开。比如，SkVM 关注的是技能如何被编译到不同模型和执行环境；而 RAG-MCP 和 AutoTool 关注的是在大量工具说明中如何选择当前任务需要的工具。本文同时受到这两类工作的启发，但更接近工具配置优化：环境定义、相关引理、证明示例和 Hammer 都是证明 workflow 中的外部能力来源，关键不在于重新生成这些能力，而在于决定它们以何种策略进入证明上下文。

本文与这些技能优化工作有相似出发点：同一套技能或上下文并不一定适合所有模型。对于 LLM 自动定理证明而言，能力较强的模型可能更善于从少量定义中推断证明方向，而能力较弱的模型可能更依赖频繁提供相关引理或证明示例；同一模型在不同证明任务上也可能需要不同的信息源。如果把“获取定义”“检索引理”“检索证明示例”“调用 Hammer”等操作理解为证明智能体的领域技能，那么本文研究的就是这些技能在证明 workflow 中的接入方式。*always* 策略相当于默认激活某项技能，*once* 策略相当于只在证明开始时提供一次信息，*on demand* 策略则把技能调用推迟到系统判断需要时。不同策略对应不同的上下文成本和信息时效性，也对应不同模型能力假设。

与上述通用技能优化工作不同，本文不直接生成、压缩或编译技能，也不尝试构建通用的模型能力画像。本文关注的是形式化证明系统中已经存在的工具和技能如何接入证明流程，并通过探针定理评测不同接入策略在当前模型和证明任务上的效果。换言之，本文的方法更接近一种面向证明任务的经验适配：用少量具有区分能力的样本快速观察模型与工具配置的匹配程度，从而为后续完整评测选择更合适的配置。

5.4 本文与相关工作的差异

综合来看，本文与相关工作之间存在三个层面的差异。

第一，本文研究工具接入策略，而非单个模型或单个工具的能力。基于 LLM 的自动定理证明工作通常关注如何提高证明成功率，例如改进检索、生成更好的证明计划、引入自反

思或结合符号自动化工具。本文默认这些工具已经存在，并进一步追问：这些工具应该以什么频率进入提示词，哪些信息应当随证明状态更新，哪些信息只需在开始时提供。这个问题更接近系统配置和工作流设计，而不是模型训练或证明策略生成本身。

第二，本文显式考虑模型差异。许多工具调用和工作流优化方法会在一个固定模型上评估优化效果，然后把得到的工作流视为系统设计的一部分。本文的出发点不同：由于模型会持续升级，固定工作流很可能很快过时。因此，本文把“不同模型是否需要不同 agent 设计”作为研究问题之一，并通过 MiniMax M2.5、OpenAI o4-mini、Qwen3-235B-A22B 和 GLM-5 四个后端模型的实验展示这种差异。这样做的意义在于，工具配置不再被看作一次性调参结果，而是被看作需要随模型变化重新评估的对象。

第三，本文强调低成本筛选，而非完整自动优化。已有工作中有不少方法尝试直接学习工具选择策略、生成工作流或训练模型掌握工具调用。相比之下，本文提出的探针方法更保守：它不承诺自动找到全局最优配置，也不替代完整评测，而是在完整评测之前提供一个可解释的小规模筛选阶段。这种定位更适合当前 LLM 自动定理证明系统的现实条件，因为一次完整实验往往需要大量模型调用、证明助手运行和检索开销。通过先筛掉明显不合适的配置，研究者可以把有限预算集中在更有希望的候选设计上。

从方法论上说，本文可以被理解为把软件工程中的选择性回归测试思想引入 LLM 证明系统配置选择。选择性回归测试不会覆盖所有输入，但会选择一组对系统行为敏感的测试用例，用于快速发现修改带来的行为变化。探针定理也承担类似作用：它们不代表完整数据集上的最终性能，却能帮助系统在模型或配置变化后快速发现工具接入行为是否发生显著变化。这一点与形式化证明场景尤其契合，因为每个探针定理都有明确的成功标准，评测结果比开放式自然语言任务更容易解释。

6 总结与展望

本文围绕 LLM 自动定理证明系统中的模型能力评估与工具接入适配问题展开研究。随着 LLM 证明系统逐渐从单次策略生成发展为包含检索、符号工具、证明助手反馈和反思验证的多模块工作流，系统性能不再只取决于后端模型的通用能力，也取决于该模型在具体证明任务中如何利用外部工具。以环境定义、相关引理和相关证明示例三类信息源为例，仅 *always*、*once* 和 *on demand* 三种接入方式就可以形成 $3^3 = 27$ 种配置。问题在于，我们很难直接定量描述一个模型适合哪种工具调用方式；通用模型排名或模型名称并不能告诉我们它在 Rocq 证明任务中更依赖定义、引理还是证明示例。

因此，本文进一步强调了模型能力适配这一动机。工具调用方式不是一组可以脱离模型固定下来的参数。不同后端模型对上下文长度、检索噪声、证明示例和按需工具调用的敏感程度不同；随着模型持续升级，原先适合某个模型的 agent 工作流未必仍然适合新模型。本文关注如何用少量探针定理作为模型能力的低成本代理，判断一个模型在当前证明任务中更适合哪种工具接入方式，而不只是在当前样本中寻找某个最优配置。

针对这一问题，本文提出使用少量探针定理近似测量模型-配置匹配关系。本文首先将探针选择建模为定理子集选择问题：探针集合应当能够放大不同模型在不同工具接入方式下的行为差异，同时避免多个定理重复表达同一种证明模式。为此，本文通过小规模预评测构造定理的行为向量，用不同模型和不同配置下的表现方差衡量定理对模型能力差异的区分能力；同时结合静态难度特征、定理相似度和 token 成本，将探针选择建模为带预算约束的子集选择问题，并使用基于单位成本边际收益的贪心算法进行近似求解。

在实验部分，本文以 RECENT-PROVER^[17]为基础系统，在 CoqStoq^[18]样本上评估探针是否能够作为模型能力代理。实验分别使用 MiniMax M2.5^[19]、OpenAI o4-mini^[20]、Qwen3-235B-A22B^[21]和 GLM-5^[22]作为后端模型，并在随机采样的 10 个工具接入配置上比较探针集和验证集结果。实验采用两阶段准则：先用成功证明数量判断模型在某配置下是否具有足够证明能力，再在候选配置中比较每百万 token 成功数。结果显示，四个后端模型上的探针选择都能够在独立验证集上保持一致。这说明少量探针定理能够在一定程度上保留模型能力信号，可以作为判断模型适合哪类 agent 工具调用方式的低成本代理。

实验还回答了模型能力差异问题。MiniMax M2.5 和 OpenAI o4-mini 在当前样本上表现出对动态引理和证明示例更新的偏好：它们更适合只在开始阶段获取环境定义、并在每轮证明中更新相关引理和证明示例。Qwen3-235B-A22B 更适合按需获取相关引理和证明示例，以控制上下文规模；GLM-5 则在当前候选集合中更适合持续提供完整上下文。这一差异说明，不同模型在证明任务中利用工具上下文的方式不同。对于持续演进的 LLM 证明系统而言，这一点很重要：模型升级以后，系统不能默认原有工作流仍然最优，而应当用探针重新检测新模型的能力与工具接入配置之间的匹配关系。

本文的工作也存在若干局限。首先，实验规模仍然较小。当前实验中每个模型只随机采样 10 个配置，尚未覆盖全部 27 种工具接入方式；探针集和验证集规模也分别只有 7 个和 15 个定理，因此实验结论更适合作为初步验证，不能视为最终性能结论。其次，当前实验主要用于说明模型能力差异的存在，仍需要在更多随机种子、更大验证集和更多模型上确认这种能力画像是否稳定。再次，本文主要使用成功数和 token 消耗刻画模型在某配置下的能力与成本，还没有充分考虑证明时间、失败类型、证明脚本长度和交互轮数等指标。对于实际证明系统而言，这些因素也会影响我们对模型能力的判断。最后，本文的探针选择依赖预评测结果，而预评测本身也需要一定成本；当模型或工具集合频繁变化时，如何复用已有预评测信息仍有待进一步研究。

未来工作可以从三个方向继续推进。第一，可以扩大实验规模，在完整配置空间和更大规模 CoqStoq 样本上验证探针作为模型能力代理的稳定性，并进一步考察不同随机种子、不同候选集合和不同预算下的结果变化。第二，可以引入更细粒度的模型能力指标，将证明成功数、token 成本、运行时间、失败原因和证明脚本质量共同纳入探针评估目标，使得到的能力画像更符合实际系统使用需求。第三，可以探索动态工具接入策略。目前本文主要比较静态配置，即某类工具采用固定的 *always*、*once* 或 *on demand* 策略。后续可以根据探针暴露出的模型能力特征，在证明过程中动态决定是否调用某个工具。

此外，后续还可以把探针结果进一步转化为更明确的模型能力画像。例如，如果某个模型在多数探针上都更依赖每轮更新的引理，那么说明它在证明过程中可能需要持续外部知识补充；如果某个模型在提供过多证明示例时效率下降，那么说明它可能对示例噪声或上下文膨胀更敏感。这样，探针评测可以从一次配置筛选工具扩展为模型升级后的 agent workflow 诊断机制。

从落地角度看，一个可行的使用方式是把探针评测纳入证明系统的持续集成流程。每当后端模型版本、提示词模板、检索器或工具调用逻辑发生变化时，系统先在固定探针集合上运行一轮小规模评测，比较新旧模型在成功数、token 成本和失败类型上的变化。如果新模型在原配置下表现稳定，就可以暂时沿用原有 agent 设计；如果探针结果显示模型能力画像发生变化，则再启动更大规模验证。这种流程类似软件工程中的回归测试，不能证明系统在所有定理上都保持最优，但可以在较低成本下暴露明显的模型行为变化。

当然，探针评测也需要治理。探针集合如果长期不更新，可能逐渐不能代表新的证明任务；如果频繁根据实验结果调整探针，又可能过拟合当前模型和配置。因此，后续系统可以维护两类样本：一类是长期固定的核心探针，用于比较不同模型版本之间的变化；另一类是随任务分布更新的扩展探针，用于覆盖新项目、新库和新工具行为。核心探针保证横向可比性，扩展探针保证任务覆盖面。这样的设计能够使本文方法从一次论文实验进一步发展为可持续的系统评测机制。

本文的结果还说明，LLM 证明系统很难依靠一套固定提示词长期适配不同模型。在自动定理证明中，证明状态会随着每一步策略改变，工具返回的信息也具有时效性；同时，不

同模型对上下文、检索结果和证明示例的利用方式并不相同。因此，系统需要持续评估模型在具体证明任务中的能力，并根据评估结果调整证明工作流。探针方法提供的正是这样一个低成本入口：每次系统变化后，先运行一组有区分能力的定理，观察模型在不同工具调用方式下的行为是否发生变化，再决定是否需要更大规模的重新评测。

这种思路也可以推广到本文没有覆盖的工具类型。例如，未来系统可能加入自动生成辅助引理的模块、调用外部 SMT 求解器的模块、根据失败历史自动选择归纳变量的模块，或者使用另一个模型专门负责反思验证。每加入一种工具，配置空间都会进一步扩大。如果仍然依赖完整数据集穷举评测，系统会越来越难维护。探针方法提供了一种可扩展的评测框架：无论工具种类如何增加，都可以先寻找能暴露配置差异的定理，再用这些定理压缩早期搜索空间。

因此，本文工作的价值不只在于当前三类信息源的实验结果，也在于提出了一种面向 LLM 证明系统演化的模型能力诊断方式。它把工具配置选择从一次性实验转化为可重复执行的模型能力测量过程，使系统能够在模型和工具不断变化时保持可控的调参成本。

更进一步看，LLM 自动定理证明系统可以被视为一种领域特定的 LLM 智能体工作流。本文的方法只是对其中少数工具接入策略进行了初步优化。随着后端模型、检索方法和证明助手接口不断发展，证明系统中的工具种类和配置空间还会继续扩大。如何在可控评测成本下为不同模型、不同证明任务自动选择合适的工具组合，将是提升 LLM 自动定理证明实用性的重要问题。

参考文献

- [1] LEVESON N, TURNER C. An investigation of the therac-25 accidents[J/OL]. Computer, 1993, 26(7): 18-41. DOI: 10.1109/MC.1993.274940.
- [2] U.S. SECURITIES AND EXCHANGE COMMISSION. Order instituting administrative and cease-and-desist proceedings: in the matter of knight capital americas llc: Release No. 34-70694[R/OL]. U.S. Securities and Exchange Commission, 2013. <https://www.sec.gov/files/litigation/admin/2013/34-70694.pdf>.
- [3] EUROPEAN SPACE AGENCY. Ariane 5 flight 501 failure: report by the inquiry board[R/OL]. ESA and CNES, 1996. <https://sci.esa.int/documents/34817/36255/1567255205913-esa-x-1819eng.pdf>.
- [4] BERTOT Y, CASTERAN P. Interactive theorem proving and program development[M]. SpringerVerlag, 2004.
- [5] de MOURA L, KONG S, AVIGAD J, et al. The Lean theorem prover (system description)[C]//Automated Deduction - CADE-25. Cham: Springer International Publishing, 2015: 378–388.
- [6] PAULSON L C. Isabelle: the next 700 theorem provers[A]. 2000.
- [7] KLEIN G, ANDRONICK J, ELPHINSTONE K, et al. Sel4: formal verification of an operating-system kernel[J/OL]. Commun. ACM, 2010, 53(6): 107–115. <https://doi.org/10.1145/1743546.1743574>. DOI: 10.1145/1743546.1743574
- [8] LEROY X. Formal verification of a realistic compiler[J/OL]. Commun. ACM, 2009, 52(7): 107–115. <https://doi.org/10.1145/1538788.1538814>. DOI: 10.1145/1538788.1538814
- [9] GONTHIER G. Formal proof—the four-color theorem[J/OL]. Notices of the American Mathematical Society, 2008, 55(11): 1382–1393. <https://www.ams.org/journals/notices/200811/tx081101382p.pdf>.
- [10] COHEN C, MÖRTBERG A. A coq formalization of finitely presented modules[C]//Interactive Theorem Proving. Cham: Springer International Publishing, 2014: 193–208.
- [11] DOCZKAL C, SMOLKA G. Regular language representations in the constructive type theory of coq[J/OL]. J. Autom. Reason., 2018, 61(1–4): 521–553. <https://doi.org/10.1007/s10817-018-9460-x>. DOI: 10.1007/s10817-018-9460-x
- [12] FISHER K, LAUNCHBURY J, RICHARDS R. The hacms program: using formal methods to eliminate exploitable bugs[J/OL]. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2017, 375(2104). <https://doi.org/10.1098/rsta.2015.0401>. DOI: 10.1098/rsta.2015.0401

- [13] CZAJKA Z, KALISZYK C. Hammer for coq: automation for dependent type theory[J/OL]. *J. Autom. Reason.*, 2018, 61(1–4): 423–453. <https://doi.org/10.1007/s10817-018-9458-4>. DOI: 10.1007/s10817-018-9458-4
- [14] BLANCHETTE J C, BÖHME S, PAULSON L C. Extending sledgehammer with smt solvers[C]//*Automated Deduction – CADE-23*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011: 116–130.
- [15] QIAN Y, CLUNE J, BARRETT C, et al. Lean-auto: an interface between lean 4 and automated theorem provers[C]//*Computer Aided Verification*. Cham: Springer Nature Switzerland, 2025: 175–196.
- [16] NORMAN C, AVIGAD J. Canonical for automated theorem proving in lean[C/OL]//*16th International Conference on Interactive Theorem Proving (ITP 2025)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025: 14:1–14:20. <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2025.14>. DOI: 10.4230/LIPIcs.ITP.2025.14
- [17] SUN Y, SHI C, LYU H, et al. On reasoning-centric llm-based automated theorem proving[A/OL]. 2026. <https://arxiv.org/abs/2604.19558>.
- [18] THOMPSON K, SAAVEDRA N, CARROTT P, et al. Rango: adaptive retrieval-augmented proving for automated software verification[C/OL]//*Proceedings of the 47th International Conference on Software Engineering (ICSE)*. Ottawa, Canada, 2025. <https://arxiv.org/abs/2412.14063>. DOI: 10.48550/arXiv.2412.14063
- [19] MINIMAX. Minimax-m2.5: built for real-world productivity[Z]. <https://www.minimax.io/news/minimax-m25>. 2026. Accessed 2026.
- [20] OPENAI. O4-mini[Z]. <https://platform.openai.com/docs/models/o4-mini>. 2025. OpenAI API model documentation.
- [21] TEAM Q. Qwen3 technical report[A/OL]. 2025. <https://arxiv.org/abs/2505.09388>.
- [22] GLM-5-TEAM, :, ZENG A, et al. Glm-5: from vibe coding to agentic engineering[A/OL]. 2026. <https://arxiv.org/abs/2602.15763>.
- [23] SCHULZ S. System description: e 1.8[C]//*Logic for Programming, Artificial Intelligence, and Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013: 735–743.
- [24] KOVÁCS L, VORONKOV A. First-order theorem proving and vampire[C]//*Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013: 1–35.
- [25] de MOURA L, BJØRNER N. Z3: an efficient smt solver[C]//*Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008: 337–340.

- [26] BARRETT C, CONWAY C L, DETERS M, et al. Cvc4[C]//Computer Aided Verification. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011: 171–177.
- [27] LEWIS P, PEREZ E, PIKTUS A, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks[C/OL]//Advances in Neural Information Processing Systems. Curran Associates, Inc., 2020: 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.
- [28] KARPUKHIN V, OGUZ B, MIN S, et al. Dense passage retrieval for open-domain question answering[C/OL]//Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Online: Association for Computational Linguistics, 2020: 6769–6781. <https://aclanthology.org/2020.emnlp-main.550/>. DOI: 10.18653/v1/2020.emnlp-main.550
- [29] KASIBATLA S R, AGARWAL A, BRUN Y, et al. Cobblestone: a divide-and-conquer approach for automating formal verification[J/OL]. 2025. <https://arxiv.org/abs/2410.19940>. DOI: 10.48550/arXiv.2410.19940
- [30] SVIRIDENKO M. A note on maximizing a submodular set function subject to a knapsack constraint[J/OL]. Operations Research Letters, 2004, 32(1): 41–43. DOI: 10.1016/S0167-6377(03)00062-2.
- [31] OPENAI. Text-embedding-3-large[Z]. <https://platform.openai.com/docs/models/text-embedding-3-large>. 2024. OpenAI API model documentation; model released 2024-01-25.
- [32] JIANG A Q, WELLECK S, ZHOU J P, et al. Draft, sketch, and prove: guiding formal theorem provers with informal proofs[C/OL]//International Conference on Learning Representations. 2023. <https://openreview.net/forum?id=SMA9EAovKMC>.
- [33] YANG K, SWOPE A M, GU A, et al. Leandojo: theorem proving with retrieval-augmented language models[C/OL]//Advances in Neural Information Processing Systems. 2023. https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.
- [34] YUE L, BHANDARI K R, KO C Y, et al. From static templates to dynamic runtime graphs: a survey of workflow optimization for llm agents[A/OL]. 2026. <https://arxiv.org/abs/2603.22386>.
- [35] YAO S, ZHAO J, YU D, et al. React: synergizing reasoning and acting in language models[C/OL]//International Conference on Learning Representations. 2023. https://openreview.net/forum?id=WE_vluYUL-X.
- [36] SHINN N, CASSANO F, GOPINATH A, et al. Reflexion: language agents with verbal reinforcement learning[C]//Advances in Neural Information Processing Systems. 2023.

- [37] SCHICK T, DWIVEDI-YU J, DESSÌ R, et al. Toolformer: language models can teach themselves to use tools[C/OL]//Advances in Neural Information Processing Systems. 2023. <https://arxiv.org/abs/2302.04761>.
- [38] PATIL S G, ZHANG T, WANG X, et al. Gorilla: large language model connected with massive apis[C/OL]//Advances in Neural Information Processing Systems. 2024: 126544–126565. https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html. DOI: 10.52202/079017-4020
- [39] QIN Y, LIANG S, YE Y, et al. Toolllm: facilitating large language models to master 16000+ real-world apis[C/OL]//International Conference on Learning Representations. 2024. <https://openreview.net/forum?id=dHng2O0Jjr>.
- [40] XU W, HUANG C, GAO S, et al. Llm-based agents for tool learning: a survey[J/OL]. Data Science and Engineering, 2025, 10: 533–563. <https://link.springer.com/article/10.1007/s41019-025-00296-9>. DOI: 10.1007/s41019-025-00296-9
- [41] ALAZRAKI L, REI M. Meta-reasoning improves tool use in large language models[C/OL]//Findings of the Association for Computational Linguistics: NAACL 2025. Albuquerque, New Mexico: Association for Computational Linguistics, 2025: 7900–7912. <https://aclanthology.org/2025.findings-naacl.440/>. DOI: 10.18653/v1/2025.findings-naacl.440
- [42] WU B, MEIJ E, YILMAZ E. A joint optimization framework for enhancing efficiency of tool utilization in llm agents[C/OL]//Findings of the Association for Computational Linguistics: ACL 2025. Vienna, Austria: Association for Computational Linguistics, 2025: 22361–22373. <https://aclanthology.org/2025.findings-acl.1149/>.
- [43] QIAN C, ACIKGOZ E C, WANG H, et al. SMART: self-aware agent for tool overuse mitigation[C/OL]//Findings of the Association for Computational Linguistics: ACL 2025. Vienna, Austria: Association for Computational Linguistics, 2025: 4604–4621. <https://aclanthology.org/2025.findings-acl.239/>. DOI: 10.18653/v1/2025.findings-acl.239
- [44] GAN T, SUN Q. Rag-mcp: mitigating prompt bloat in llm tool selection via retrieval-augmented generation[A/OL]. 2025. <https://arxiv.org/abs/2505.03275>.
- [45] JIA J, LI Q. Autotool: efficient tool selection for large language model agents[A/OL]. 2025. <https://arxiv.org/abs/2511.14650>.
- [46] CHEN L, FENG E, XIA Y, et al. Skvm: compiling skills for efficient execution everywhere[A/OL]. 2026. <https://arxiv.org/abs/2604.03088>.

致谢

首先感谢我的导师熊英飞老师。我仍然记得第一次听《软件科学基础》时的兴奋，那像是推开了一扇通往新世界的门，那时的我消沉且低落，而这门课中的 Rocq 代码居然能给我带来极大的快乐，又让我找回了当初选择计算机科学的热情。后来的《软件分析技术》课程，又让我进一步领略到编程语言研究的魅力。熊老师为人谦和，治学严谨，教会我如何在研究中保持好奇心和批判性思维，也让我明白学术道路上的坚持、耐心与自我要求。

感谢实验室的孙奕灿、房建、黄柘铨、邢文恺学长和石城炜同学。无论是选题讨论、实验推进，还是论文写作中的具体问题，你们都给了我许多细致而真诚的帮助。

感谢我的父母、姐姐、爷爷奶奶、叔叔婶婶和哥哥弟弟。是你们长期以来的支持和鼓励，让我有勇气追求自己的兴趣，也能在遇到困难时继续往前走。无论在学业上还是生活中，你们始终是我坚实的后盾。我仍然记得我第一次开车剐蹭时，父母没有责怪我，而是第一时间安慰我，并嘱咐我注意安全，妈妈还等在门口，直到我深夜处理完事故回家；我仍然记得我说我喜欢吃臭豆腐，爷爷奶奶总是张罗着在我回家时一定要让我吃到家乡的臭豆腐；我仍然记得我小时家庭条件比较困难时，叔叔说的就算是借钱也要让我上学。我小时是一个很内向的人，但是我的每一次尝试你们都在背后默默支持，鼓励我去做新的事情，去认识新的朋友。正是有了你们的支持，我才能够上大学里不断尝试新的事物，结交新的朋友，逐渐变得更加自信和开朗。

当然，也要感谢我自己。感谢过去的自己在面对挑战时没有放弃，感谢现在的自己在不断学习和成长。未来的路还很长，希望我能继续保持对知识的热爱和对未知的好奇，不断前行。

最后，以一首我从高中就铭记于心的诗来结束我的致谢：

野歌

【唐】李贺

鸦翎羽箭山桑弓，仰天射落衔芦鸿。
麻衣黑肥冲北风，带酒日晚歌田中。
男儿屈穷心不穷，枯荣不等嗔天公。
寒风又变为春柳，条条看即烟蒙蒙。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：

吕杭州

日期：2026年5月8日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保留学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；

论文作者签名：

吕杭州

导师签名：

斯公子

日期：2026年5月8日