arXiv:2602.01291v1 [cs.LO] 1 Feb 2026

构造–验证：Lean 4 中应用数学形式化的基准测试

Bowen Yang [1] **Yi Yuan** [1] **Chenyi Li** [1] **Ziyu Wang** [1] **Liangqi Li** [2] **Bo Zhang** [2] **Zhe Li** [2] **Zaiwen Wen** [1]

摘要

大语言模型近期的进展展现了其在数学形式化方面令人瞩目的能力。然而，现有基准测试侧重于对声明式命题进行逻辑验证，往往忽视了明确合成解这一任务。这一局限在应用数学领域尤为突出，因为这些领域的目标往往是推导出具体数值或可执行算法，而非仅仅证明定理。为此，我们引入了一个 Lean 4 框架，强制采用构造—验证工作流程，要求智能体先定义明确的解，再证明其正确性。我们精心整理了一个全面的基准测试 **AMBER**（**应用数学基准测试**，用于**推理**），涵盖应用数学的核心领域，包括凸分析、优化、数值代数和高维概率论。除定理证明外，我们的基准测试还包含评估、算法设计和表示转换等复杂任务。实验表明，当前模型在这些构造性任务上面临显著困难。值得注意的是，我们发现通用推理模型的表现始终优于专用定理证明器。我们认为，这是因为专用模型的指令遵循能力有所退化。在证明语料库上进行微调似乎会导致“策略过拟合”，削弱模型遵循复杂构造性要求的能力；而通用模型则保留了多任务形式化推理所需的灵活性。

[1]中国北京大学 [2]蚂蚁集团旗下蚂蚁数科。通讯作者：Zaiwen Wen<wenzw@pku.edu.cn>。

预印本。*2026*年*2*月*3*日。

1. 引言

现代数学日益复杂，加之数学证明中的错误可能造成严重后果，这使得快速验证和错误检测成为迫切需求。这一需求凸显了形式化数学至关重要。交互式定理证明器（交互式定理证明器）使用形式化数学语言，例如 Harrison，1996年），Coq（Huet 等人，1997年），Isabelle（Paulson，1994），以及 Lean（de Moura 与 Ullrich，2015年；Moura 与 Ullrich，2021），通过实现对证明的自动验证并确保绝对可靠性来应对这一挑战，显著降低了传统人工审查中固有的人为错误风险。

然而，这种正确性保障是以显著的易用性门槛为代价的：构造形式化证明需要掌握大量库的专业知识，而且由于存在重复性任务，往往是一个耗时费力的手动过程。这一挑战催生了一个前景广阔的研究方向：利用大语言模型（LLM）在这些形式化系统中实现自动推理。值得注意的是，通用大语言模型（例如 DeepSeek-V3.2-Thinking（Liu等人，2025a）、Kimi K2（Team等人，2025年）、Gemini-3 Pro，以及经过微调的模型，例如 Seed-Prover（Chen等人，2025年）、DeepSeek-Prover-V2（Ren等人，2025），在自动形式化方面均取得了显著成功，尤其是在证明高中竞赛题方面。

为了评估并提升大语言模型在自动形式化方面的能力，研究人员开发了多种专门的基准测试，例如 MiniF2F（Zheng等人，2021）、ProofNet（Azerbayev等人，2023）。然而，现有基于 Lean 的基准测试主要聚焦于初等数学或纯数学，对本科层次及更高层次应用数学的覆盖明显不足。更关键的是，这些基准测试对应用数学核心概念的表征存在系统性不足，例如数值求值、算法设计和表示变换，这最终阻碍了形式化数学知识边界的拓展。

为弥合这一差距，我们提出，要有效评估应用数学推理，需要进行范式转变

从验证存在命题转向综合构造性解法。与纯粹的定理证明不同，在后者中，证明解的存在往往就已足够；应用数学则要求推导出显式值和可执行算法。这一区别要求采用一种严谨的形式化方法，在结构上强制执行“构造—验证”工作流程，防止智能体利用非构造性捷径。通过使形式化任务与凸优化和数值代数等领域的计算直觉相契合，我们旨在为高阶数学中的自动推理确立一种更忠实的标准。

此外，在这一构造性框架中评估当前模型，揭示了一项根本性权衡：对形式逻辑的高强度专门化往往会削弱复杂数学建模所需的通用推理能力与指令遵循能力。我们观察到，在证明语料库上进行微调这一主流策略虽对标准演绎有效，却可能在无意中损害多阶段数学建模所需的指令遵循灵活性。这一现象凸显了当前神经符号系统中的一项重大权衡：局部策略熟练度与应用数学所必需的高层算法综合之间的张力。

综上所述，我们的贡献如下。

- 我们针对不同类别的应用数学开发了专门的形式化范式。具体而言，我们为求值问题、算法设计与表示变换设计了一种构造—验证模式。

- 我们构建了**AMBER** （**A**应用 **M**数学**BE**基准测试，用于**R**推理）基准测试，覆盖凸分析、优化、数值代数、概率论等核心应用领域，难度等级不一，从标准本科习题到博士级研究问题。

- 我们进行了全面评估，结果表明当前模型在构造性应用数学方面表现不佳。我们的分析揭示了证明搜索专门化与一般构造能力之间的权衡，为未来的神经符号研究提供了新的见解。

2. 相关工作

形式化基准测试。 形式化基准测试对于严谨、可复现地评估自动定理证明能力至关重要。在大学预科数学领域，MiniF2F基准测试（Zheng等人，2021）聚焦于数学竞赛中的问题，这一方向也由 FIMO（Liu et al.，2023）和 PutnamBench（Tsoukalas et al.，2024年）所探索。在大学层面，ProofNet（Azerbayev等人，2023）提供了一个用于形式化本科层次数学的关键数据集。后续工作进一步丰富了这一领域，包括针对组合学的专项基准测试（Xionget al.，2025年；Liu et al.，2025b），以及抽象代数基准测试 FATE（Jiang et al.，2025年）。

基于大语言模型的自动形式化。Polu & Sutskever（2020）的早期工作率先将基于搜索的方法用于证明生成。后续方法探索了多种搜索策略，包括最佳优先搜索的变体（Yang et al., 2023；Lin et al., 2024；Wu et al., 2025；Li et al., 2024；Xin et al., 2025）以及蒙特卡洛树搜索（Lample et al., 2022；Gloeckle et al., 2024；Xin et al., 2024）。近年来，该领域逐渐转向单遍生成，最新的证明器便是其中的代表（Ren et al., 2025；Lin et al., 2025；Wang et al., 2025；Zhou et al., 2025；Chen et al., 2025）。这些模型通常通过大规模强化学习进行训练，并生成较长的思维链推理，以分解和修正 Lean 代码，在 miniF2F 基准测试上取得接近 100% 的准确率。

不同类型内容的形式化。 认为所有数学知识都可以形式化的观点，推动了对各种领域进行形式化的大量工作。研究人员针对不同类型的内容开发了框架和基准，但定理证明领域的形式化工作相对而言仍未得到充分探索。例如，(Miranda et al.，2025年) 在 VeriBench 中对 Python 算法进行了形式化，(Liu et al.，2025c) 提出了一个用于形式化问题求解过程的框架，而 (Bentkamp etal.，2023) 则为优化问题的形式化提供了见解。

3. 应用数学的形式化

尽管形式化验证环境能够确保严格的正确性，标准基准测试主要鼓励逻辑有效性（证明某个命题为真），而非构造性合成（寻找解）。这导致评估任务与应用数学中的常见需求不匹配。这些问题的首要目标并非验证解是否存在，而是推导出具体的符号或数值结果。为此，我们提出一种灵活的框架，利用 Lean 丰富的构造性特性（包括定义、归纳类型和类型类实例），以适应面向计算的任务。这种方法能够显式建模解结构和算法，从而确保在兼容标准形式化验证逻辑的同时，评估智能体推导未知量取值的能力。

3.1. 求值问题

在应用数学中，一类常见问题涉及显式解合成，例如求解线性方程、优化目标函数或确定概率分布，而非仅仅验证此类解是否存在。这一区别至关重要：推导解需要构造性搜索过程和数值推理，其范围超出了检查候选解或以非构造性方式证明解存在。因此，这些评估问题代表了有别于传统定理证明的独特范式，需要专门设计的形式化框架。

考虑以下二次优化问题作为一个典型示例：

$$\min_{x\in\mathbb{R}^n} \quad f(x) = \frac{1}{2}x^T Ax - b^T x, \tag{1}$$

其中 $A \in \mathbb{R}^{n\times n}$ 是一个对称正定矩阵，而 $b \in \mathbb{R}^n$ 是一个常向量。该优化问题的解为 $x^* = A^{-1}b$。以往的方法通常将此类求值问题建模为存在命题（即证明存在 x，使得 x 是问题（1）的最优点）（Liuet al., 2025c）。然而，逻辑上的存在性远弱于构造性推导。虽然识别出显式解$x^* = A^{-1}b$ 自动意味着该解存在，但通过非构造性方法确立存在性并不能提供任何计算层面的洞见。例如，仅凭定义域紧致性和函数连续性，就可以通过极值定理证明最优解存在，而无需计算 x^* 的具体值。我们发现，大语言模型经常利用这类抽象策略证明存在性，却不构造显式解，如附录B.2中的案例研究所示。这种不一致会使大语言模型在没有构造任何解的情况下也被认为解决了求值问题，从而削弱基准测试评估应用计算能力的目标。为确保稳健性，我们要求采用更严格的表述，以强制进行显式解的构造。

在上述示例中，目标是推导出一个构造性的目标变量 x^*，用给定的参数 A 和 b 表示，使得 x^* 满足最优性条件。基于这一思路，我们提出以下形式化定义。

定义 3.1（求值问题）。给定一个性质 $Q(x, V)$ 及参数 V，求值问题是构造一个表达式 $x(V)$，使得 $Q(x(V)\ V)$ 成立。

我们通过以下两阶段形式化模式来实现这一点：

1. 构造。需要显式定义一个函数（def），以计算解项 $x(V)$。

2. 验证。随后需要一个定理来证明这个特定项满足性质 $Q(x(V)\ V)$。在优化示例（1）中，$V = \{A, b\}$，构造项为 $x(V) = A^{-1}b$，而性质 $Q(x,\ V)$ 是如下命题：x 是一个最优点。一般的形式化模式结构如下。

```
变量 {V： 类型_V} {Q： 类型_X →
    类型_V →命题} {hV： 命题}
    -- hV: V 的假设

定义 x (V： 类型_V)： 类型_X： =sorry

定理 验证 (V： 类型_V):
    Q （x V） V： = 由 sorry
```

关键在于，这种架构设置了严格的语法屏障，防止使用非构造性捷径。尽管模型能够读取完整的问题上下文，但 x 的定义受到其函数签名的严格约束：它仅接受原始参数 V 作为输入，而不接受目标性质 Q 或其证明。这种结构设计从根本上阻止模型利用解存在这一事实来构造解本身。因此，智能体必须进行显式构造，根据可用参数逐步构建解项。更多示例请参见附录B.2。

3.2. 算法设计问题

应用数学中很大一部分内容都与定义算法有关。这包括用于优化的迭代算法，例如梯度下降法；用于优化或求根的牛顿法；以及用于大规模学习的交替方向乘子法（ADMM）。在这些问题中，解不是一个闭式表达式，而是一条转移规则，它在离散时间步中演化状态，以实现特定行为（例如收敛到一个驻点）。这里的目标是评估智能体综合算法逻辑本身的能力，而非仅仅验证抽象序列的性质。

考虑用于最小化可微函数 $f(x)$ 的经典梯度下降算法。该算法通过数学方式指定初始化 x_0 和迭代更新规则来描述：

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{2}$$

其中 α 为步长。重点在于以构造性定义说明如何从 x_k 推导出 x_{k+1}。

为了有效地对算法设计进行基准测试，我们强制实施一项结构性约束：智能体必须显式提供循环体。我们将算法视为定义在自然数上的递归函数。智能体的任务是在给定第 n 步的情况下，编写第 $(n + 1)$ 步的代码。为了形式

化这一数学直觉，我们提出如下算法定义。

定义3.2（算法）。给定状态空间 S 和参数 V，构造算法需要定义一个序列生成器 $A : \mathbb{N} \to S$，使得：

$A(0) = \text{init}(V)$（基本情形），$A(n+1) = \Phi(A(n), V)$（归纳步骤），其中 Φ 是待构造的转移函数。

我们利用 Lean 的递归定义能力（对归纳类型自然数进行匹配）对算法设计问题中的迭代结构进行建模。这一过程分为两个阶段。

设计。智能体必须定义一个函数，将迭代次数和当前状态映射到下一个状态。我们将其形式化为寻找一个转移函数 $\Phi : S \times V \to S$，其中 S 是状态空间，V 是问题参数。

分析。序列 A 得到有效定义后，智能体必须证明某个性质 $P(A)$，例如单调性（$f(A(n+1)) \leq f(A(n))$）或收敛性。

在我们的基准测试中，这是通过下面的模板实现的。智能体必须严格用可计算逻辑（更新规则）填入第一个 sorry，并用正确性证明填入第二个 sorry。抽象形式化模板如下所示。

```
变量 {状态: 类型} (V:类型_v)
变量 (P : (自然数 → 状态) →命题)

定义 算法 (x0 : 状态) (V : 类型_v):自然数
 –> 状态| 0    :=x0| (n+1):= sorry


定理 验证
   (x0 :状态)(V: 类型_v) :
   P (算法 x0 V) : = bysorry
```

该框架将评估重点从求出一个数值转向编写循环体，确保智能体真正扮演算法设计者的角色。更多示例请参见附录 B.2。

3.3. 表示变换

在实际应用中，数学问题很少以标准形式出现。应用数学家的关键能力之一是建模：将原始、非标准的问题转化为结构化表示的能力，例如整数线性规划、半正定规划或特定的矩阵分解，从而确保计算可处理性。这一过程既不是简单地推导数值，也不是迭代式算法设计，而是对问题陈述本身进行结构变换。

为了评估这项能力，我们定义了一类表示变换问题。在这里，目标是在源问题实例与目标标准形式之间建立严格的映射。我们将数学问题一般地定义为一个元组 $\mathcal{P} = (X, C, f)$，其中包括定义在领域 X 中的决策变量 x、一组约束 $C(x)$（其中 $C : X \to \text{Prop}$），以及目标函数 $f : X \to V$；其中 V 表示值空间，例如优化问题中的 $\mathbb{R}$ 或特征值问题中的复数集合 $\mathbb{C}^n$。这项任务的目标是将源问题 $\mathcal{P}_{src} = (X, C, f)$ 转换为目标问题 $\mathcal{P}_{tgt} = (Y, C', f')$，其中 $\mathcal{P}_{tgt}$ 通常属于某一类易于处理的问题，例如凸问题。我们通过松弛这一概念来形式化这些问题之间的关系。

定义 3.3（问题松弛与等价）。我们在优化问题空间上定义二元关系 $\preceq$，以形式化可归约性的结构层级。若源问题中的每个可行解都能映射为目标问题中的一个可行解，且目标值相同，则称 $\mathcal{P}_{tgt}$ 是 $\mathcal{P}_{src}$ 的松弛，记作 $\mathcal{P}_{src} \preceq \mathcal{P}_{tgt}$：

$$\forall x \in X, C(x) \implies \exists y \in Y, C'(y) \wedge f(x) = f'(y). \quad (3)$$

如果两个问题在这一关系下彼此可比较，并满足反对称性质：$\mathcal{P}_{src} \preceq \mathcal{P}_{tgt} \wedge \mathcal{P}_{tgt} \preceq \mathcal{P}_{src}$，则称它们等价。

这种形式化体现了数学建模的本质。例如，要证明某个特定的图割问题与谱聚类表述等价，就需要证明割的集合能够映射到拉普拉斯矩阵的谱（其中 f 将图配置映射到特征值）。在 Lean 中，我们通过要求智能体定义目标问题的结构，并证明松弛关系或等价关系，使这一过程得以具体实现。该框架区分了源问题（作为参数给定）与目标形式（由智能体构造）。

建模（构造）。智能体必须定义目标问题的组成部分：新领域 Y、约束条件 C' 和目标函数 f'。

验证。智能体必须证明逻辑命题 $\mathcal{P}_{src} \preceq \mathcal{P}_{tgt}$（或等价命题）。

为了使这一过程严谨化，我们首先定义一个通用的 MathProb- lem 结构，其中封装了决策域 D、约束谓词和目标函数。该框架的核心是 relaxes 定义，它将问题归约的概念形式化：它断言，对于源问题 P_1 中的每个可行解，都存在一个

映射到目标问题 P_2 中的解，且其目标值相同。在此任务中，代理会收到一个源问题 P src，并须履行三项具体义务以解决 sorry 占位符：构造目标域 D tgt（通常需要引入辅助松弛变量）、明确定义目标问题 P tgt，并证明有效的变换定理；这要求证明源问题与目标问题之间存在相互松弛。表示变换的通用模式如下所示。代理充当翻译器，将用户定义的问题转换为求解器所需的标准形式。

```
结构 数学问题 (DV： 类型)： =
  (约束: D → 命题)
  (目标函数： D → V)

定义 放宽 {D1 D2 V: 类型}(P1: 数学问题
D1 V) (P2: 数学问题 D2 V): 命题: =∀ x,
P1.约束 x →∃ y, P2.约束 y ∧ P1.目标函数 x=
P2.目标函数 y

变量 {D_源 V: 类型}（P_源： 数学问题 D_源
V）定义 D_ 目标： 类型: =sorry 定义 P_ 目
标： 数学问题 D_目标 V: =sorry

定理 有效变换: ¯放宽 P_源 P_目标 ∧放宽
P_目标 P_源： =通过sorry
```

通过以这种方式构建基准测试，我们确保评估指标与问题归约的数学直觉相符。我们要求智能体做的不只是解决问题，还要证明它对数学结构有足够深入的理解，能够在不同领域间转换这些结构，同时保留其基本性质。案例研究见附录B.2。

3.4. 定理证明

虽然我们的基准测试侧重于构造性任务，但定理证明仍是形式化验证的基石。我们纳入了一部分传统定理证明任务，作为基线，用于评估模型的逻辑推理能力，并将其与构造能力区分开来。与前几类任务中解答是数据结构或算法不同，这里的目标纯粹是演绎性的。给定上下文 Γ 中的命题 P，任务是构造一个证明项 t，使得 $\Gamma \vdash t : P$。在 Lean 4 中，这遵循标准的声明模式。

```
定理 命题_名称 (V： 参数)(h： 假设):
目标_命题： = 通过 sorry
```

智能体必须用有效的策略序列或证明项替换 sorry，从而完成目标。这些问题涵盖从确立凸集的性质到证明高维概率论中的不等式，可作为对照组，用于衡量模型是否会以牺牲纯粹的演绎能力为代价，换取构造上的灵活性。

4. 基准整理与特征

4.1. 基准整理

问题收集。内容选择优先考虑基准测试的代表性和领域特有特征，样本主要取自凸分析、优化、数值线性代数和高维概率论领域的经典且广受认可的教材（Boyd& Vandenberghe，2004年；Golub & Van Loan，2013年；Vershynin，2018年）。内容由应用数学专业的博士生和高年级本科生遴选，他们负责确定能够代表核心概念的主题。根据问题来源进行的初步难度分类作为基线：简单问题对应标准课程习题，困难问题则与博士资格考试题相当。我们将样本分为定理证明和构造—验证两类。后一类要求先综合出解答，再对评估、算法设计和表示变换任务进行验证。详细定义见第3节。附录C提供了各领域任务分布的完整统计数据。通过纳入这两种任务类型，我们的基准测试旨在评估模型在不同数学推理模式下的能力——从概率论中所需的严谨推导，到优化和数值代数中不可或缺的算法综合。

Lean形式化。形式化由一支经验丰富的 Lean 用户团队执行。团队采用标准化模板，确保严格遵循 Mathlib 规范，并将每个子领域中精选的 50 个问题转换为可执行的 Lean 代码。这一严谨流程确保生成的代码与原始数学陈述保持高度的逻辑一致性，并能在标准 Lean 4 环境中出色地复现。

审查。验证采用严格的两阶段流程，确保形式化 Lean 代码与原始数学问题之间严格的语义一致性，包括基于大语言模型的初步检查和人工

深入验证。在大语言模型初步检查中，我们使用最先进的大语言模型自动筛查整个语料库，逐一核对每个形式化问题与其原始表述。大语言模型将形式化代码与原始问题表述进行比较，以检测潜在的语义差异、逻辑遗漏或不一致之处；所有标记出的案例都会优先进行人工审查和修订。有关具体模型、配置和提示策略的详细信息见附录C.4。人工深入验证采用多轮团队交叉验证，每位专家都是经验丰富的 Mathlib 贡献者。每一轮中，每个形式化问题至少由两位 Lean 专家审查，他们会严格验证所有构造——包括自定义定义/引理和调用的 Mathlib 概念——是否符合原始问题的意图。

4.2. 基准测试特征

专家驱动的难度评估。传统基准测试可能依赖语法长度或启发式代理指标来衡量难度，而我们的分类则以专家的数学共识为依据。为衡量假设与结论之间内在的逻辑鸿沟，我们邀请来自知名研究机构的十名博士生进行评估。他们评估构建证明蓝图所需的推理深度，以确保被归类为“困难”的问题确实体现了数学上的复杂性（例如需要新颖的辅助构造），而非仅仅是记号冗长。这种以人为中心的方法验证了我们的 AMBER 基准测试能够体现研究级应用数学的严谨性。

量化形式复杂度。在专家定性评估的基础上，我们利用 Lean 的元编程框架量化基准测试内在的信息密度。与通常以词元长度作为复杂度代理指标的基于文本的数据集不同，形式化环境使我们能够衡量问题陈述的概念依赖负荷。具体而言，我们将每个基准测试实例 p 表示为相应的 Lean 声明，并将底层形式化库建模为全局语义依赖图 $\mathcal{G}$，其顶点集合为 $\mathcal{V}$，有向边集合为 $\mathcal{E}$，其中 $\mathcal{V}$ 由定义、定理等形式化声明组成，而 $\mathcal{E}$ 对使用依赖进行编码。对于问题 p，我们将依赖密度 $\rho_k(p)$定义为其非平凡概念邻域的大小：

$$\rho_k(p) = \left| \{ v \in \mathcal{V} \mid 1 \leq \mathrm{dist}_{\mathcal{G}}(p, v) \leq k \wedge v \notin \mathcal{S}_{\mathrm{trivial}} \} \right|,$$

其中，$\mathrm{dist}_{\mathcal{G}}(p, v)$ 表示问题 p 到概念 v 在依赖图中的最短路径距离。集合$\mathcal{S}_{\mathrm{trivial}}$ 充当高通语义滤波器，用于去除逻辑连接词和基本类型等纯句法构造，从而提取领域专门知识。我们将局部半径 $k = 3$设为该值，以涵盖理解陈述所需的直接概念前提，同时避免更深层级中的基础公理稀释指标。较高的 $\rho_k(p)$ 表明模型必须成功检索数学概念的稠密子图并进行推理，才能正确解析问题，这对模型的检索增强生成（RAG）和长上下文理解能力构成了重大挑战。为量化数据集 $\mathcal{D}$的整体复杂度，我们计算各项密度的算术平均值，作为平均依赖密度：$\bar{\rho}_k(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{p \in \mathcal{D}} \rho_k(p)$。为使这些指标更具背景意义，我们将 AMBER 与三个成熟的基线进行比较：MiniF2F（高中竞赛）、ProofNet（本科数学）和 FATE。图1 展示了结果。我们的 AMBER 基准测试具有最高的依赖密度，以定量方式证实了 AMBER 更深的语义深度。

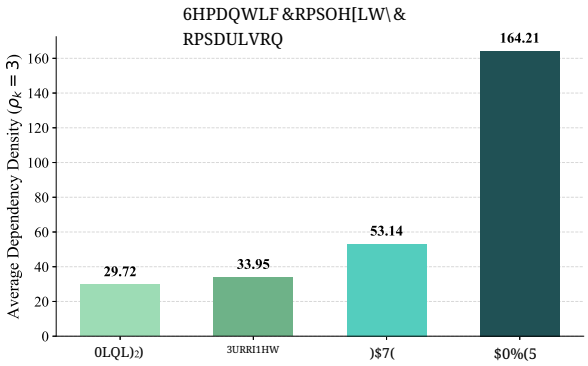

图 *1.* 语义复杂度的定量比较。

5. 实验

为了评估当前模型在应用数学形式化方面的能力，我们在 AMBER 基准测试上进行了全面评估。我们的实验不仅关注定理证明，还关注智能体遵循结构要求的能力，具体来说，即完成各种形式的 Lean 内容，包括第 3 节所述的评估、算法设计和表示变换。

5.1. 实验设置

模型。我们评估了一系列代表两种不同范式的模型：通用推理智能体和专用定理证明器。在通用推理方面，我们采用 DeepSeek-V3.2-Thinking （Liu et al.，2025a ）和 GPT-5.1，它们代表了自然语言理解和思维链推理领域的最先进能力。在专用形式化方面，我们评估了 Goedel Prover-32B（Lin et al.，2025年）和 Kimina Prover-72B （Wangetal.，2025年），这些模型经过了大量微调

表*1.* 各模型在不同子集上的 Pass@16 结果。每个数据集被划分为简单子集和困难子集。

	优化		凸分析		数值代数		概率论	
模型	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
DeepSeek-V3.2-Thinking	3/24	0/26	5/26	0/24	3/25	0/25	1/25	0/25
GPT-5.1	2/24	0/26	3/26	0/24	3/25	0/25	1/25	0/25
Gemini-3.0 Pro	3/24	0/26	4/26	0/24	4/25	0/25	1/25	0/25
Goedel Prover-32B	3/24	0/26	1/26	0/24	3/25	0/25	1/25	0/25
Kimina Prover-72B	2/24	0/26	2/26	0/24	3/25	0/25	1/25	0/25

所用语料库包括 Mathlib 和 Lean。

评估协议。 对于每个问题，模型都会获得一个上下文头（导入和变量声明）以及一条基于与问题类型相对应的结构模板的 Lean语句，涵盖评估、算法设计、表示变换和传统定理证明。关键在于，模型必须通过填充 `sorry` 占位符来完成代码，分别补全定义（构造）和定理（验证）。我们报告 Pass@k 指标（$k = 16$）。仅当在 k 个样本中，至少有一个生成的解使 Lean 能够成功编译构造性定义和验证证明且不报错时，才认为问题已解决。不同 k 取值下 Pass@ k 的无偏估计量详见附录 D.3。其他超参数和提示工程细节见附录D。

5.2. 基准测试性能

我们在表1中总结了四个应用数学子领域的表现。结果揭示了当前自动形式化的若干重要见解，其中最值得注意的是，与 MiniF2F 等标准纯数学基准测试相比，总体通过率极低。这一性能差距在困难子集中尤为明显，几乎所有受评模型的成功率都接近于零。这些发现验证了我们的假设：应用数学带来了独特的结构性挑战。智能体必须克服两方面的困难：一是理解领域特定证明所需的内在逻辑深度，二是构造准确数学对象时所涉及的显著结构复杂性。这一难度与我们的定量分析结果一致：AMBER 基准测试的语义依赖密度显著高于现有数据集，因此要求模型在更加密集的数学概念子图上进行推理。我们在表 k 中提供了 $k \leq 16$ 的 Pass@2无偏估计结果，完整结果详见附录 D.3。

通用模型优于专用证明器。 一个引人注目的发现是，通用推理

表 *2.* 在完整的 AMBER 基准测试上，各模型的 Pass@k ($k = 1$, 16) 统计数据。数值表示在给定 k 预算下解决问题的百分比。

模型	Pass@1	Pass@16
通用推理模型		
DeepSeek-V3.2-Thinking	1.68%	6.00%
GPT-5.1	1.40%	4.50%
Gemini-3.0 Pro	2.62%	6.00%
专用定理证明器		
Goedel Prover-32B	0.97%	4.00%
Kimina Prover-72B	1.03%	4.00%

模型始终优于专用数学证明器，这一趋势与通常的性能层级相矛盾；在该层级中，专用证明器在标准基准测试中占据主导地位。我们将此归因于一种称为“策略过拟合”的现象：尽管对状态-策略对进行大规模微调能够最大限度地提升模型解决逻辑目标的能力，但这似乎会诱发一种“锤子找钉子”的偏见，使模型将每个结构性挑战都视为纯粹的定理证明任务。这种专门化会导致通用对齐能力发生灾难性遗忘，使模型忽略应用数学的多阶段特性。因此，它们难以遵循严格的“构造—验证”模板，常常跳过所需的定义（定义 块），或错误地尝试使用非构造性证明策略来解决构造目标。相比之下，通用模型保留了稳健的指令遵循能力，因此能够弥合非形式化意图与形式语法之间的差距，在进入验证阶段之前正确地构造数学对象。附录E对此现象进行了详细分析。

5.3. 提示消融研究。

为验证我们提出的框架的有效性以及结构约束的必要性，我们针对优化和凸分析领域的全部 100 个问题进行了广泛的消融研究。我们将构造—验证提示词与两个基线进行了比较：

标准代码补全（提供文件但不提供结构指示）和传统纯证明（仅要求给出证明）。附录 D.4中的详细结果表明，构造—验证策略的成功率最高，为 8%。相比之下，标准补全的成功率降至 6%，而仅要求证明的提示词则进一步降至 5%。

表 *3.* 100 题子集上的消融结果。

提示类型	成功率
构造—验证	**8/100**
标准补全	6/100
传统纯证明	5/100

定性分析表明，传统的纯证明提示经常会误导模型使用非构造性的捷径。例如，模型常常会采用 Classical.choice 等策略，在不提供计算解的情况下，通过逻辑推理证明存在性。此外，标准补全提示往往会导致模型忽略填充定义块的要求，转而尝试仅使用证明搜索策略来解决整个问题。这些发现证实，明确的结构化提示对于防止模型绕过基准测试的构造性要求至关重要。

5.4 错误分析

为探究应用数学形式化过程中的具体障碍，我们对 DeepSeek-R1 模型生成的 3,189 条随机抽样错误消息进行了详细的人工分类。分析表明，这些挑战的分布并不均匀，某些特定的失败模式在结果中占主导地位。

大语言模型幻觉。 最常见的错误来源是调用不存在的定理或定义，占错误总数的 47.3%（1,512 个案例）。在这些情况下，模型会臆造听起来合理、但实际上并不存在于 Mathlib 库中的标识符（例如 c onvexHull eq sInter）。这表明，尽管模型通常能够正确识别必要的数学步骤，却缺乏实现这些步骤所需的 Lean 4 库签名的精确记忆。

形式化能力。 第二大类别占其余错误的 32.7%，源于形式化技能不足。这类失败发生在模型使用有效的定理和策略，却错误地应用它们、误判某个策略的效果，或未能完成证明时
重写所需的必要侧条件。这反映了模型的高层推理与其低层策略执行能力之间存在差距。

不完整输出。 尽管有明确指示要求提供完整可执行的代码，仍有 15.2%（487 个案例）的输出包含以 sorry 或 submit 标记的不完整片段。这一问题在复杂的算法设计任务中尤为突出：模型成功勾勒出算法结构，却在归纳步骤的严格证明上半途而废；当推导变得过于复杂时，便退回到使用占位符。

格式与语法。最后，还有一小部分错误占 4.8%（154 个案例），归因于基本的输出格式错误，例如未能将代码封装在标准 Markdown 代码块中，或生成语法格式错误的标题，导致编译器无法解析文件。

总而言之，近一半的失败源于大语言模型的幻觉，这表明未来的改进应优先考虑检索增强生成或专门微调，以使模型的内部知识与 Mathlib 库的确切状态保持一致。详情和案例研究见附录 F。

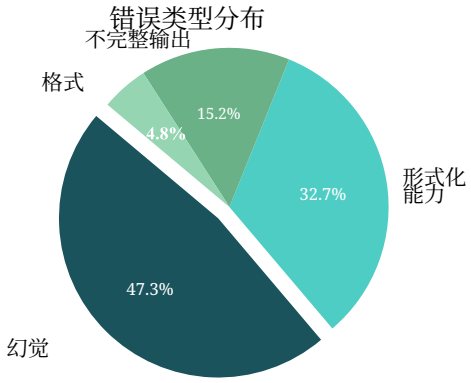

图 *2.* 错误分析分布。

6. 结论

在这项工作中，我们提出了一个全面的 AMBER 基准，专为 Lean 4 中的应用数学设计。该基准通过将评估范式从存在性验证转向显式的解构造，解决了现有数据集中一个关键的不匹配问题。除了定理证明之外，我们提出的任务还涵盖数值求值、算法设计和表示变换，促使智能体不再局限于单纯的定理证明，而是先综合可计算定义并进行结构建模，再开展验证。大量评估表明，这种构造性范式对最先进的模型构成了重大挑战。尤其值得注意的是，评估揭示了一种“策略过拟合”现象：专用定理证明器虽然擅长闭合局部逻辑目标，却表现出僵化，难以遵循多阶段构造性指令；相比之下，通用模型则展现出更强的灵活性。这些结果凸显了

超越纯粹证明搜索、转向神经符号系统的必要性。这类系统将高层次算法推理与严谨的形式化验证相结合，可为未来拓展形式化应用数学的前沿奠定基础。

参考文献

Azerbayev, Z.、Piotrowski, B.、Schoelkopf, H.、Ayers, E. W.、Radev, D. 与 Avigad, J. Proofnet：本科数学的自动形式化与形式化证明。*arXiv* 预印本 *arXiv:2302.12433*，2023。

Bentkamp, A.、Fern´andez Mir, R. 与 Avigad, J. 面向优化的已验证归约。载于系统构造与分析工具和算法国际会议，第74–92页。施普林格，2023。Boyd, S. 与 Vandenberghe, L. 凸优化。剑桥大学出版社，2004年。Chen, L.、Gu, J.、Huang, L.、Huang, W.、Jiang, Z.、Jie, A.、Jin, X.、Jin, X.、Li, C.、Ma, K. 等。Seed-Prover：面向自动定理证明的深度与广度推理。*arXiv* 预印本 *arXiv:2507.23726*，2025年。Chen, M. 评估在代码上训练的大语言模型。*arXiv* 预印本 *arXiv:2107.03374*，2021。de Moura, L. 与 Ullrich, S. Lean 4 定理证明器与编程语言（系统描述）。2015年。Gloeckle, F.、Limperg, J.、Synnaeve, G. 与 Hayat, A. Abel：面向神经定理证明的样本高效在线强化学习。载于 *NeurIPS'24* 第四届数学推理与人工智能研讨会，2024年。Golub, G. H. 与 Van Loan, C. F. 矩阵计算。约翰斯·霍普金斯大学出版社，2013年。Harrison, J. HOL Light：教程导论。载于 计算机辅助设计形式化方法国际会议，第265–269页。施普林格，1996年。Huet, G.、Kahn, G. 与 Paulin-Mohring, C. Coq 证明助手教程。技术报告，178:113，1997年。Jiang, J.、He, W.、Wang, Y.、Gao, G.、Hu, Y.、Wang, J.、Guan, N.、Wu, P.、Dai, C.、Xiao, L. 等。FATE：面向多个难度级别前沿代数的形式化基准测试系列。*arXiv* 预印本 *arXiv:2511.02872*，2025年。

Lample, G., Lacroix, T., Lachaux, M.-A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., and Martinet, X. 用于神经定理证明的超树证明搜索。神经信息处理系统进展，35：26337–26349，2022年。

Li, Y., Du, D., Song, L., Li, C., Wang, W., Yang, T., and Mi, H. Hunyuanprover：用于自动定理证明的可扩展数据合成框架与引导式树搜索。*arXiv*预印本 *arXiv:2412.20735*，2024年。

Lin, H., Sun, Z., Welleck, S., andYang, Y. Lean-star：学习交错进行思考与证明。*arXiv* 预印本 *arXiv:2407.10040*，2024年。

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., 等。Goedel-prover-v2：通过脚手架数据合成与自我纠正扩展形式化定理证明。*arXiv* 预印本 *arXiv:2508.03613*，2025年。

Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., 等。Deepseek-v3. 2：推动开放式大语言模型的前沿发展。*arXiv* 预印本 *arXiv:2512.02556*，2025a。Liu, C., Shen, J., Xin, H., Liu, Z., Yuan, Y., Wang, H., Ju, W., Zheng, C., Yin, Y., Li, L., 等。Fimo：用于自动定理证明的挑战性形式化数据集。*arXiv* 预印本 *arXiv:2309.04295*，2023。Liu, J., Lin, X., Bayer, J., Dillies, Y., Jiang, W., Liang, X., Soletskyi, R., Wang, H., Xie, Y., Xiong, B., 等。Com-bibench：对大语言模型组合数学能力进行基准测试。*arXiv* 预印本 *arXiv:2505.03171*，2025b。Liu, Q., Zheng, X., Xia, R., Qi, X., Cao, Q., and Yan, J. 超越定理证明：形式化问题求解的表述、框架与基准测试。*arXiv* 预印本 *arXiv:2505.04528*，2025c。Miranda, B., Zhou, Z., Nie, A., Obbad, E., Aniva, L., Fronsdal, K., Kirk, W., Soylu, D., Yu, A., Li, Y., 等。Veribench：用于 Lean 4 人工智能代码生成的端到端形式化验证基准测试。载于第 *2* 届 *AI for Math* 研讨会@ 国际机器学习大会 *2025*，2025。Moura, L. d. and Ullrich, S. Lean 4 定理证明器与编程语言。载于国际自动演绎会议，第 625–635 页。施普林格，2021。Paulson, L. C. *Isabelle*：一种通用定理证明器。施普林格，1994。Polu, S. and Sutskever, I. 用于自动定理证明的生成式语言建模。*arXiv* 预印本 *arXiv:2009.03393*，2020。Ren, Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., 等。Deepseek-prover-v2：通过用于子目标分解的强化学习推进形式化数学推理。*arXiv* 预印本 *arXiv:2504.21801*，2025。

Team, K., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., 等。Kimi K2：开放式智能体智能。*arXiv* 预印本 *arXiv:2507.20534*，2025年。

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jen- nings, M., Thakur, A., and Chaudhuri, S. Putnambench：评估神经定理证明器在普特南数学竞赛中的表现。神经信息处理系统进展，37:11545–11569，2024年。Vershynin, R. 高维概率论：导论及其在数据科学中的应用，第47卷。剑桥大学出版社，2018年。Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., et al. Kimina-prover 预览版：借助强化学习构建大型形式化推理模型。*arXiv* 预印本 *arXiv:2504.11354*，2025年。Wu, Z., Huang, S., Zhou, Z., Ying, H., Yuan, Z., Zhang, W., Lin, D., and Chen, K. Internlm2. 5-stepprover：通过评论器引导搜索推进自动定理证明。发表于第二届 *AI for Math* 研讨会@ *ICML 2025*，2025年。Xin, H., Ren, Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., et al. Deepseek-prover-v1. 5：利用证明助手反馈进行强化学习和蒙特卡洛树搜索。*arXiv* 预印本 *arXiv:2408.08152*，2024年。Xin, R., Xi, C., Yang, J., Chen, F., Wu, H., Xiao, X., Sun, Y., Zheng, S., and Ding, M. Bfs-prover：面向基于大语言模型的自动定理证明的可扩展最佳优先树搜索。发表于计算语言学协会第*63*届年会论文集（第*1*卷：长篇论文），第32588–32599页，2025年。Xiong, B., Lv, H., Shan, H., Wang, J., Yang, Z., and Zhi, L. 用于自动定理生成的组合恒等式基准。*arXiv* 预印本 *arXiv:2502.17840*，2025年。Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. LeanDojo：借助检索增强语言模型进行定理证明。神经信息处理系统进展，36:21573–21612，2023年。Zheng, K., Han, J. M., and Polu, S. MiniF2F：形式化奥林匹克级数学的跨系统基准。*arXiv* 预印本 *arXiv:2109.00110*，2021。Zhou, Y., Zhao, J., Zhang, Y., Wang, B., Wang, S., Chen, L., Wang, J., Chen, H., Jie, A., Zhang, X., et al. 通过分解和迭代反思求解形式化数学问题。*arXiv* 预印本 *arXiv:2507.15225*，2025年。

A. 形式化与 Lean 简介

数学形式化是将自然语言（非形式化数学）中的数学定义、定理和证明转化为严格形式语言的过程，这种形式语言可由计算机进行机械检查。本附录简要介绍 Lean 定理证明器，重点讲解理解本文所述基准测试所必需的概念。

A.1. 形式化验证与 Lean

Lean 是由微软研究院开发的交互式定理证明器（ITP）和函数式编程语言。与侧重于数值或符号计算的计算机代数系统（例如 MATLAB、Mathematica）不同，Lean 侧重于逻辑正确性。它验证推导过程是否无歧义地遵循数学公理。

Lean 的核心底层逻辑是归纳构造演算（CIC）。在该系统中，数学陈述与证明分别被视为类型和程序。这种对应关系被称为柯里–霍华德同构，它意味着：

- 命题（数学陈述）是一种类型。
- 该命题的证明是一个具有该类型的项（一个程序）。
- 验证证明等价于对程序进行类型检查。

如果用户编写的证明项能够针对某个定理陈述成功通过类型检查，那么该定理就必然为真（相对于系统的一致性而言）。

A.2. 定义与定理

Lean 中的一个关键区别，也是我们 AMBER 基准测试的核心主题，是定义数据（定义）与证明事实（定理）之间的差异。在应用数学中，我们通常两者都需要：构造一个解（算法或值），然后证明它是正确的。

定义（定义）。定义用于构造数学对象、函数或算法。这些属于计算性或结构性实体。例如，在自然数上定义平方函数：

```
定义 平方 (n: ℕ) : ℕ := n * n
```

在我们的基准测试中，构造阶段（例如，定义最优的 x^*，或编写梯度下降循环）对应于创建一个定义。

定理（定理）。定理用于断言命题。它们不产生数据，而是承载逻辑证据。例如，证明一个正数的平方为正：

```
定理 平方_正 (n : ℕ)(h : n > 0): 平方 n > 0 := 通过-- 证明写在
这里sorry
```

在我们的基准测试中，验证阶段对应于建立一个定理，以验证先前定义的定义所具有的性质。

A.3. 策略与自动化

直接编写证明项（例如构造复杂的 λ演算项）对人类而言十分繁琐。Lean 提供了策略模式，通常由关键字 by 引入。策略是用于操纵证明状态并自动生成底层证明项的元程序。

常用策略包括：rw（重写）：使用等式替
换项。

simp（化简）：使用代数规则数据库化简表达式。apply：应用引理，
将当前目标化为子目标。linarith：自动求解线性算术不等式。

在我们的实验中，大语言模型必须预测这些策略序列，以关闭逻辑目标。sorry 关键字是一个占位符，用于在不提供证明的情况下接纳目标。该基准测试的目标是将每个 sorry 替换为有效代码或策略。

A.4. Mathlib：数学库

Lean 由 Mathlib 提供支持。Mathlib 是一个统一的、由社区驱动的数学库，包含涵盖代数、拓扑学、分析、概率论等领域的形式化定义和定理。对于应用数学，Mathlib 提供了必要的基础设施，例如：

线性代数：矩阵、向量、特征值和内积空间的定义。**分析：**导数、梯度、凸性和
连续性的概念。**概率论：**测度论、积分和随机变量。

我们的 AMBER 基准测试高度依赖 Mathlib。例如，数据集中的一个优化问题会调用 Mathlib 对正定矩阵的定义。因此，大语言模型的表现不仅取决于其推理能力，也取决于它对 Mathlib 特定应用程序接口和命名约定的熟悉程度。

A.5. 为什么将 Lean 用于应用数学?

Lean 4 虽然历来用于纯数学（例如验证费马大定理），但它本身是为通用编程语言而设计的。这使其特别适合本文所涉及的算法与计算问题。

与纯逻辑系统不同，Lean 允许我们定义可执行算法、在输入上运行这些算法，甚至同时证明算法会收敛。这种双重特性使我们的基准测试能够评估智能体在应用数学的完整范围内的表现：从建模与计算（编程）到严谨论证（证明）。

B. 形式化框架的详细信息与示例

在本节中，我们将介绍 第 3 节中形式化框架的详细信息。我们首先展示详细的 类 定义 n

B.1. 表示变换类

为了系统地评估模型执行表示变换的能力，我们在 Lean 4 中开发了一套全面的模块化类型类基础设施。该框架旨在严格形式化应用数学中通常隐含的问题重述过程，将其从启发式步骤转化为精确的类型映射任务。与标准数值计算不同，表示变换要求将非标准原问题重构为计算上易于处理的标准形式，例如线性规划或半定规划，同时保持数学等价或松弛性质。为此，我们建立了分层的类结构。我们首先定义了一个通用接口 OptProble
m，用以封装决策变量域、约束谓词集合和目标函数。在此基础上，特别针对优化理论的基石之一——对偶理论，我们设计了专门的类结构来形式化对偶问题的推导。这一设计不仅封装了拉格朗日函数的构造，还明确规定了对偶变量空间及其相关约束的类型。通过强制采用这种结构化定义，我们促使智能体超越直观的符号推导，转而构造经过验证的结构变换，并在 Lean 类型系统中给出正确性的严格证明。

```
类 原始问题，其中
 n_变量 :  ℕ
 约束 :  (有限类型 n_变量  → ℝ) → 命题
 目标函数 :  (有限类型 n_变量  → ℝ)  → ℝ

类 优化问题扩展原始问题，其中
 n_等式 :  ℕ
 方程_约束 : (有限类型 n_变量  → ℝ) → (有限类型 n_方程  → ℝ)
 n_ieqs : ℕ
 不等式_约束 : (有限类型 n_变量  → ℝ)  →(有限类型 n_不等式  → ℝ)
 约束 := 函数 x  => 方程_约束 x  = 0 ∧ 不等式_约束 x  ≤ 0
 h_约束 : 约束  = 函数 x  =>方程_约束 x  = 0 ∧ 不等式_约束 x  ≤ 0:
     = 通过simp

类 线性规划 继承优化问题，其中
 c : 有限类型 n    _变量 → ℝ
 A_等于 : 矩阵 (有限类型 n_方程组) (有限类型 n_变量) ℝ
 b_等于 : 有限类型 n_方程组 → ℝ
 A_不等式 : 矩阵(有限类型 n_不等式组) (有限类型 n_变量)  ℝ
 b_不等式 : 有限类型 n_不等式组 → ℝ
 目标函数 := 函数x =>c ·ᵥ x
 等式_约束 := 函数 x  => A_等式 *ᵥ x  - b_等式
 不等式_约束 := 函数 x  => A_不等式 *ᵥ x  - b_不等式
 h_目标函数:目标函数 =函数x => c  ·ᵥ x := 通过 simp
 h_等式 :  等式_约束  = 函数 x  => A_等于 *ᵥ x  - b_等式 :  = 由 simp
 h_不等式 :  不等式_约束  = 函数 x  => A_不等式  *ᵥ x  -b_不等式 :  = 由 simp

类 半定规划 扩展原始问题，其中
 c: 有限类型 n    _变量 → ℝ
 n_等式 :  ℕ
 A_等式 :  矩阵(有限类型 n_等式)(有限类型 n_变量)  ℝ
 b_等式 :  有限类型 n_等式 → ℝ
 等式_约束 :  = 函数 x  => A_等式 *ᵥ x  - b_等式
 n_不等式 :  ℕ
 A_sd :  有限类型 n_变量  → 矩阵 (有限类型 n_不等式)(有限类型 n_不等式)  ℝ
 B_sd :  矩阵(有限类型 n_不等式)(有限类型 n_不等式)  ℝ
 ieq_约束 := fun x => Σ i, x i  ·A_sd i  + B_sd
 约束 := fun x  => eq_约束 x  = 0 ∧(ieq_约束 x).正定
 h_约束 : 约束  = fun x  =>eq_约束 x  = 0 ∧ (ieq_约束
     x).正定:= bysimp
 目标函数 := funx => c ·ᵥ x
 h_目标函数:目标函数 =funx => c  ·ᵥ x := by simp

定义 子等价 (p q : 原始问题): 命题 :=
 ∀ (x : 有限类型 p.n_变量  → ℝ), (p.约束 x)  →
 ∃ (y : 有限类型 q.n_变量  → ℝ), (q.约束 y)  ∧
 q.目标函数 y = p.目标函数 x

定义 等价(pq:原始问题) : 命题 :=
  子等价 p q ∧ 子等价 q p

类 对偶问题(p: 优化问题)其中
 对偶_目标函数: (Finp.n_等式 → ℝ) → (Fin p.n_不等式  → ℝ)  → 扩展实数
 对偶_领域 : 集合 ((Fin p.n_等式  → ℝ)  ×(Fin p.n_不等式  → ℝ))
 h_目标函数 : 对偶_目标函数  = 函数 lam mu  =>(⨅x : (Fin p.n_变量  → ℝ), ((lam  ·ᵥ
     p.eq_约束 x) +(mu ·ᵥ p.ieq_约束 x)  + p.目标函数 x).转为扩展实数) := 由 simp
 h_领域 : 对偶_领域 = {(lam,mu) | 对偶_目标函数 lam mu  ≠ ⊥} := 由 simp
```

B.2. 不同问题类别的示例

本节将通过具体示例，说明 AMBER 基准测试中定义的不同问题类别。

B.2.1. 构造—验证与定理证明

构造—验证问题要求求解器给出数学问题的一个明确解，而不只是证明这样的解存在。下面是一个具有代表性的示例，内容为.....

非形式化陈述。 求解该优化问题

$$f = \|x\|, \text{ s.t. } x \in \left\{(x, y, z) | x^2 + y^2 + z^2 \leq 1\right\}.$$

定理证明方法是将优化问题转化为存在性定理，然后使用 use 策略提取见证，如下所示。

```
不可计算的 定义 f : (EuclideanSpace ℝ(有限类型 3)) → ℝ :=fun x ↦ |x||定
义 ball := Metric.closedBall(0:EuclideanSpace ℝ (有限类型 3)) 1 定理 优化
_问题′ : ∃ x ∈ball, IsMinOn f ball x :=by use 0 constructor




 · simp [球]
 · 引入 x hx
 simp [f]
```

在实践中，由大语言模型生成的 Lean 代码往往默认采用一种便于自动化的模式：它们通过将目标改写为标准的存在性陈述（例如借助极值论证）来消去目标，从而得到对 $\exists x, \ldots$ 的证明，而不是对具体极小元的显式构造。因此，即便底层优化问题具有简单的闭式解，生成的证明脚本也可能偏向于证明某个 x 的存在性，而不是定义一个具体的 x 并证明其可行性与最优性。在下面的例子中，该证明仅通过说明可行集是紧的来确立极小元 x 的存在性，而没有给出极小元 x 的显式构造。

```
不可计算的 定义 f : (EuclideanSpace ℝ(有限类型 3)) → ℝ :=fun x ↦ |x||定义 ball :=
Metric.closedBall (0 : EuclideanSpace ℝ (有限类型 3)) 1 定理 优化_问题 {n : ℕ} [非
零 n]:∃ x ∈ ball, IsMinOn f ball x :=-- 证明 ball 是一个紧集 have hcompact :
IsCompact ball :=isCompact_closedBall 0 1 hcompact.exists_isMinOn (by simp [
ball])continuous_norm.continuousOn
```

B.2.2. 求值问题

求值问题要求明确构造一个解项，而不仅仅是给出存在性证明。这类任务分为两个子类：数值计算问题，其中问题参数是具体数值；以及符号推导问题，其中解必须用抽象变量表示。下面给出一个涉及二次规划（QP）的数值计算问题示例。

非形式化陈述。 求以下优化问题的一个最优点。

$$\begin{array}{ll}\text{minimize} & (1/2)x^T Px + q^T x + r \\ \text{subject to} & -1 \le x_i \le 1, \quad i = 1, 2, 3,\end{array}$$

其中

$$P = \begin{bmatrix} 13 & 12 & -2 \\ 12 & 17 & 6 \\ -2 & 6 & 12 \end{bmatrix}, \quad q = \begin{bmatrix} -22.0 \\ -14.5 \\ 13.0 \end{bmatrix}, \quad r = 1.$$

上面的示例是一个典型的数值计算问题。在该问题中，我们提供了若干数值矩阵和向量，任务是以数值方式构造一个最优点。要完成 Lean 内容，需要算出结果并将其填入第一个 *sorry*，然后在定理中证明该结果的最优性。

形式化陈述。

```
定义 answer : 有限类型 3 → ℝ:=sorry

定理 num 3 S E : let x :=answer _ _ _ let P : 矩阵 (有限类型 3) (有限类型3) ℝ:= !![13, 12, -2;
12,17,6; -2, 6, 12]let q : 有限类型 3 → ℝ := ![-22, -14.5, 13]let r := 1IsMinOn (fun y => (1 /
2) * (y ·ᵥ P *ᵥ y) + (q ·ᵥ y) + r) {x | ∀ i : 有限类型 3, -1 ≤ x i ∧ x i ≤ 1} x∧ ∀ i : 有限类型
3, -1 ≤x i ∧ x i ≤ 1:=by sorry
```

形式化挑战。 此示例代表典型的数值计算任务。问题的输入（矩阵P和向量 q、r）以具体的数据结构形式给出。智能体必须计算出使目标函数最小化的具体数值向量 x^*，并将其填入定义块中（解决第一个 sorry）。同时，智能体随后必须在定理块中给出严格证明，证明这个特定候选解 x^* 在给定约束下满足最优性条件。与数值问题不同，符号推导任务要求智能体根据抽象参数构造通用的闭式表达式。

非形式化陈述。 推导无约束二次最小化问题的解析解：

$$\min_{x \in \mathbb{R}^n} \quad f(x) = \frac{1}{2} x^T Ax + b^T x,$$

其中 $A \in \mathbb{R}^{n \times n}$ 是一个对称正定矩阵（$A \succ 0$），而 $b \in \mathbb{R}^n$ 是任意常向量。

形式化陈述。

```
变量 (n : 自然数) (A : 矩阵 (有限类型 n)(有限类型 n) ℝ)(b : 有限类型 3 → ℝ)

定义 an swer (n: 自然数)(A: 矩阵(有限类型 n) (有限类型 n) ℝ) (b : 有限类型 3 → ℝ) : 有限类型 n → ℝ := sorry

定理 num 3 S E: letx:=answern A b _ _ _ IsMinOn (fun y => (1 / 2) * (y ·ᵥ A *ᵥ y) +
(b ·ᵥ y) + c) .univ x := by sorry
```

形式化挑战。 这个问题考查智能体进行符号代数运算而非算术计算的能力。智能体必须定义一个函数，以 A 和 b 作为参数，并返回项 $-A^{-1}b$（或以涉及线性求解运算的等价形式表示）。对于满足类型约束的任意 A 和 b，该定义在结构上都必须有效。接下来的定理要求证明所构造的项确实是唯一的全局最小值点。这需要运用正定矩阵和梯度的性质（例如证明 $\nabla f(x^*) = 0$ 或进行配方），而不是检查数值不等式。

B.2.3. 算法设计

非正式陈述。 假设 $L \in \mathbb{R}^{n\times n}$ 是下三角的，且 $B \in \mathbb{R}^{n\times n}$。给出一种计算 $X \in \mathbb{R}^{n\times n}$ 的算法，使得 $LX = B$。

形式化陈述。

```
open Matrix WithLp variable {n : ℕ} {L : Matrix (Fin n)(Fin n) ℝ} (B : Matrix (Fin n) (Fin n) ℝ)
variable (hL : L.BlockTriangular OrderDual.toDual) 定义 algorithm (hL : L.BlockTriangular
OrderDual.toDual) (B : Matrix (Fin n) (Fin n) ℝ) : ℕ →Matrix (Fin n) (Fin n) ℝ := fun k =>
match k with| 0 => 0| k + 1 => sorry 定理 algorithm_prop (B : Matrix (Fin n)(Fin n) ℝ) : L *
algorithm hL B n = B := by sorry
```

形式化挑战。 此问题类别用于评估智能体综合算法逻辑而非静态解答的能力。与输出为闭式项的评估任务不同，此处的目标是状态转移规则。为解决第一个 sorry，智能体必须明确实现一个利用下三角结构的前向代入算法。这要求将标准的命令式循环（例如根据先前结果计算第 个分量）转换为 Lean 的函数式递归风格，从而定义矩阵逐步演化的过程。验证阶段（第二个 sorry）要求证明算法正确性。智能体必须通过归纳法证明，所构造的序列在经过 步后恰好得到精确解。这检验了对控制流建模并建立循环不变量的能力，弥合数值算法设计与形式逻辑之间的差距。

B.2.4. 表示变换

我们遵循第 3.3 节确立的理论框架，并采用附录 B.1中提供的基础类定义，这些定义在 Lean 中实现了上述概念。本类别评估智能体识别底层数学结构并将其映射为易于处理的标准形式的能力。下面，我们给出一个代表性示例，说明如何将矩阵逼近问题化简为线性规划（LP）。

非形式化陈述 ment. 给定 $k+1$ 矩阵 $A_0, \ldots, A_k \in \mathbb{R}^{m\times n}$，我们需要找到 $x \in \mathbb{R}^k$ 使其最小化 es

$$\|A_0 + x_1 A_1 + \cdots + x_k A_k\|_\infty.$$

将此问题表示为线性规划。**形式化陈述。**

表4. 基准数据集在不同领域的详细统计数据。

	难度		问题类别				
领域	**Easy**	**Hard**	**证明**	**评估**	**算法**	**变换**	**总计**
优化	24	26	18	12	1	19	50
凸分析	26	24	36	13	1	0	50
数值代数	25	25	19	12	19	0	50
高维概率	25	25	45	5	0	0	50
总计	100	100	118	42	21	19	**200**

```
开启 作用域 矩阵。范数。算子

变量 (m n k : ℕ) (A : 有限类型 k →矩阵 (有限类型 m) (有限类型 n) ℝ)

不可计算的 定义P: 优化问题 where
  n_变量 : = k
  n_等式 : = 0
  等式_约束: = 0
  n_不等式 : = 0
  不等式_约束: = 0
  目标函数 : = 函数x => ‖Σi, xi ·(Ai)‖

定义 answer (m n k : ℕ) (A : 有限类型 k → 矩阵 (有限类型 m)(有限类型 n) ℝ) : 线性规划 := sorry

定理 num 7 T H: _ _ _令 Q : = answermnkA令 P : = P m nkA 等价P。
toOriginalProblemQ。toOriginalProblem : = 通过 sorry
```

形式化挑战。 与直接计算相比，这个问题引入了一个不同的复杂层次。要解决第一个 sorry，智能体必须显式构造线性规划形式。这涉及一个并不简单的建模步骤：理解到，最小化一个 L_∞ 范数等价于在满足线性约束（$-t \le (A(x))_{ij} \le t$）的条件下最小化一个标量松弛变量 t。因此，智能体必须正确定义一个包含辅助松弛变量的新决策空间（从 $\mathbb{R}^k$ 映射到 $\mathbb{R}^{k+1}$）。智能体还需要用根据输入矩阵 A_i推导出的正确分块矩阵表示来填充 LP 结构体的字段。此外，验证定理要求展开附录中定义的自定义 LP 定义。这要求大语言模型不仅处理当前的局部上下文，还要检索并理解全局类型定义，以构造有效的等价性证明。这一挑战考验了智能体在自然语言数学意图与特定代码库约束之间进行多模态交互的能力)

C. AMBER 基准测试的细节与分析

C.1. 术语分析

要构建一个评估深度数学推理的基准测试，仅靠文本抓取是不够的。与自然语言不同，Lean 中的形式化数学具有严格的依赖结构，其中每个符号都有明确无歧义的定义。我们开发了一个自定义静态分析引擎来提取这些语义依赖，确保数据集中的每个定理都附带其所使用的数学概念的精确定义。

C.1.1. 环境重建与AST遍历

我们流程的第一阶段是重建每个源文件的编译环境。我们利用 Lean 的编译器基础设施来 elaboration 源代码，将原始文本转换为类型完整的抽象语法树（AST）。这一步至关重要，因为它会解析导入和命名空间，使我们能够区分那些实际上完全相同的

名称（例如，Matri x.trace 与 LinearMap.trace），并依据它们在全局环境中的确切身份加以区分 t.

对于每个目标定理 T，我们提取其陈述（类型签名），并对其表达式树进行递归遍历。令 E_T表示定理 T的表达式树。我们收集陈述中出现的所有唯一常量的集合 $C_T = \{c \mid c \in \text{leaves}(E_T)\}$。此过程不仅捕获顶层概念，也涵盖论证或返回类型中使用的嵌套依赖项。

C.1.2. 语义停用词过滤

直接提取常量会得到高层次数学概念与底层逻辑基础设施的混合内容。为聚焦于特定领域的推理，我们实现了一个语义停用词过滤器。

我们将常量集合 C_T 划分为两个子集： C_{infra} 和 C_{domain}。

基础设施术语（C_{infra}）。这些包括逻辑联结词（∧、∨、→）、相等类型（Eq）、基本数据类型（Nat、String）以及编译器内部辅助定义。与自然语言处理预处理类似，这些内容被视为停用词，因为它们起的是结构性连接作用，而非数学内容。

领域概念（C_{domain}）。这些是与数学子领域相关的定义、结构和类（例如凸性、测度、特征值）。

我们的分析器会过滤掉 C_{infra}，仅保留模型必须理解才能证明该定理的领域专属概念。

该流程为每个定理 T生成结构化表示，由元组 $(S_T,\ \mathcal{D}_T)$构成，其中 S_T是形式化陈述，$\mathcal{D}_T$ 是已解析依赖项的集合，每个依赖项都包含其形式化定义、源代码位置和自然语言文档。下方列出了特定定理对应的 JSON 结构示例，展示了已解析依赖项及其相关元数据。


```
{
 “定理名称”: “是单位元_单位元_子_斜_对称”,
 “陈述”: “∀ {n : ℕ} [非零 n] (S : 矩阵 (有限类型 n)(有限类型 n) ℝ), .....”,
 “文件路径”: “基准测试/NAlg/数值_2_P_E。lean”,
 “依赖项”: [
  {
   “名称”: “矩阵.单位元” ,
   “模块”: “Mathlib.数据.矩阵.对角矩阵”,
   “种类”: “定义” ,
   “文档字符串”: null,
   “深度”: 1,

   “定义”: “{n : 类型 u_3} → ..... →单位元 (矩阵 n n α)”,
   “定义范围”: {“起始行”:199,“结束行”: 200}
  },
  {
   “name”: “矩阵.转置”,
   “module”: “Mathlib.LinearAlgebra.矩阵.Defs”,
   “kind”: “定义” ,
   “文档字符串”: “矩阵的转置。,
   “深度”: 1,
   “定义”: “{m : 类型 u} ... → 矩阵 m n α →矩阵 n m α”,
   “定义范围”: {“起始行”: 133, “结束行”: 135}
  },
  {
   “名称”: “是单位元”,
   “模块”: “Mathlib.代数.群.单位元.定义”,
   “种类”: “定义” ,
   “文档字符串”: “一个元素 `a : M`属于一个 `幺半群`是一个单位元，当且仅当.....”,
   “深度”: 1,
   “定义”: “{M : Type u} → [幺半群 M] → M →命题”,
```

```
    "定义范围": {"起始行":362,"结束行": 369}
  },
  ...
 ]
}
```


C.2. 领域统计与基准指纹分析

利用上述依赖项提取流程，我们对基准测试涵盖的数学领域进行了定量分析，并与现有数据集进行比较，具体包括 FATE（侧重于代数）和 MiniF2F（侧重于高中竞赛）。由于 Mathlib 的目录结构严格遵循数学层级，将筛选后的领域常量 C_{domain} 映射回其定义所在的模块，可以为数据集的领域侧重点提供高分辨率的特征。

我们统计了每个数据集中所有定理所导入模块的出现频率，并排除了常见的实用工具文件。结果汇总于表5，揭示了鲜明的领域特征，与各基准测试预期的范围相符。

•**FATE（代数侧重）：** 正如代数中心型基准测试所预期的那样，其依赖项高度集中于抽象代数结构。主要模块包括 Mathlib.Algebra.Lie、Mathlib.Algebra.Polynomial、Mathlib.RingTheory.Ideal 和 Mathlib.Algebra.EuclideanDomain。这证实了它在评估交换代数与非交换代数推理方面的专门用途。

•**MiniF2F（奥林匹克竞赛侧重）：** 其分布反映了大学预科数学竞赛的特点，这类竞赛优先考查初等数论、组合学和离散数学。占主导地位的模块包括 Mathlib.Data.Nat（自然数）、Mathlib.Data.Finset（有限集）和 Mathlib.Data.Nat.Cast，这表明其侧重于算术运算和有限结构，而非连续分析。

• **ProofNet（本科分析学侧重）：** 依赖概况表明，其重点强烈偏向标准的本科分析学与拓扑学课程。主要导入包括 Mathlib.Analysis.Normed.Group、Mathlib.Topology.Defs 和 Mathlib.Topology.MetricSpace。这一特征反映了以度量空间、连续性以及赋范向量空间基本性质为中心的任务。

• **AMBER（应用数学）：** 相比之下，我们的数据集呈现出独特的特征，其特点是连续优化与高维分析。最主要的依赖包括 Mathlib.LinearAlgebra.Matrix 和 Mathlib.Analysis.InnerProductSpace，它们是数值线性代数与凸优化的基础。此外，Mathlib.MeasureTheory.MeasurableSpace 的显著出现反映了高维概率任务的纳入。这一分布从经验上验证了我们的 AMBER 基准测试成功针对了应用计算数学中的空白。

C.3. 难度分析

为确保该基准测试能够作为适用于不同数学能力水平的可靠诊断工具，我们在收集过程中实施了严格的难度分层。我们根据解决源材料所需的学术水平来划分问题难度。简单问题取自标准本科课程习题，通常只需直接应用定义或进行一步推理。困难问题则选自研究生教材和博士资格考试，需要多步推理、创造性地运用辅助构造，或具备深刻的结构性洞见。表4 展示了各领域的难度分布。我们的基准测试有一项显著特点，即实现了严格的平衡：数据集恰好包含 100 道简单问题和 100 道困难问题。关键的是，这种对称性不仅体现在整体上，也体现在每个子领域内部。这种均匀的难度分层可避免评估结果受到特定领域难度突增的影响，并使研究人员能够评估智能体在困难任务上的性能下降，是否在不同数学学科中表现一致。此外，难度分类也会根据第3节介绍的具体范式进行调整。对于定理证明任务，难度与逻辑深度以及所需搜索路径的稀疏程度相关。相比之下，对于构造验证任务（评估、算法、

表 5. 不同基准测试中 Mathlib 模块依赖关系排名前三的统计。统计结果凸显了各自不同的专长：FATE（抽象代数）、MiniF2F（离散数学）、ProofNet（拓扑学/分析）和 AMBER（数值分析/应用分析）。

基准测试	顶级模块（领域签名）	数学重点
FATE	Mathlib.Algebra.Category.ModuleCat	模范畴
	Mathlib.RingTheory.Ideal.Quotient	商环
	Mathlib.FieldTheory.IntermediateField	域论
MiniF2F	Mathlib.Data.Nat	数论
	Mathlib.Data.Finset	组合学
	Mathlib.Data.Nat.Cast	算术
ProofNet	Mathlib.Analysis.Normed.Group	赋范空间
	Mathlib.Topology.Defs	一般拓扑学
	Mathlib.Topology.MetricSpace	度量空间
AMBER	Mathlib.LinearAlgebra.Matrix	数值线性代数
	Mathlib.Analysis.InnerProductSpace	优化
	Mathlib.Analysis.Convex	凸分析
	Mathlib.MeasureTheory.MeasurableSpace	概率论

变换），难度由目标对象的结构复杂性决定。例如，一个简单的构造任务可能涉及寻找标量根，而一个困难的任务则需要设计迭代矩阵分解算法或构造复杂的半正定松弛。这种多维难度结构确保基准测试能够有效区分仅记忆句法模式的模型与能够进行真正数学综合的模型。

C.4. 大语言模型验证详情

用于初步语义一致性检查的确切提示如下。

系统提示。

> 你是一位资深数学家和 Lean 4 形式化工程师。你的任务是验证自然语言数学问题与其形式化 Lean 代码之间是否严格保持语义一致。你必须找出任何逻辑遗漏、语义差异，或变量定义、约束条件及最终目标中的不一致。

用户提示。

> 请审查以下配对：1. 自然语言问题：{自然语言问题}原始自然语言问题开始{自然语言问题}原始自然语言问题结束 2. Lean 形式化：原始 LEAN 文件开始{Lean 源代码}原始 LEAN 文件结束 验证说明：变量映射：验证文本中的每个变量都存在于代码中，且类型正确（例如，实数与整数、非负约束）。假设检查：确保所有条件（例如“互异”、“素数”、“x¿0”）都明确出现在 Lean 假设中。目标对齐：检查 Lean 中的证明目标是否与文本中提出的问题严格匹配。Mathlib 使用：验证标准概念（素数、阶乘、几何定义）使用标准的 Mathlib 定义

而非不正确的自定义定义。
输出：如果代码正确，输出： [CONSISTENT]。如果存在错误，输出： [MISMATCH] ，并简要说明缺失或不正确的逻辑。

D. 实验详情

D.1. 基线设置

为构建全面的评估体系，我们选取了多种模型，涵盖最先进的通用推理引擎和专用开源定理证明器。我们通过官方应用程序接口调用专有模型，具体使用 DeepSeek-V3.2-Thinking 和 GPT-5.1。此外，我们还在本地部署了针对形式化数学微调的知名开源模型，包括 Kimina-Prover 和 Goedel-Prover，以评估领域专用训练的影响。为确保结果可复现且能代表标准用法，我们对所有模型均采用各自官方文档推荐的默认推理参数，包括温度和 top-p。最大生成长度统一设为 32,768 个词元，以适应形式化证明和构造性定义篇幅较长的特点。对于支持调整计算资源分配的增强推理模型，我们明确将推理强度参数设为中等。

D.2. 提示词

我们提供用于评估现有模型的详细提示词。

系统提示。

你是一名 Lean 4 定理证明助手。任务：- 你将收到一个数学问题对应的 Lean 4 文件内容。- 你必须生成一个完整的 Lean 4 源文件，并确保它能在给定项目中编译。严格的输出要求：- 给出该命题的完整证明。输出中不得包含 sorry。- 除 sorry 外，不得更改文件中的任何内容。- 仅输出最终文件的 Lean 4 源代码。- 不得添加任何解释、关于你正在做什么的注释，或代码之外的自然语言文本。- 不得用 Markdown 代码围栏包裹代码。- 不得更改源文件中的定义和类。- 输出必须是一个独立完整的 Lean 4 文件，可保存为 .lean 文件并直接编译。- 如果收到的是一道求解问题，你必须给出正确且完整的解答。不得用问题本身来回答问题。- 如果收到的是一道设计算法的问题，你必须给出一个正确且完整的算法。你需要考虑该算法是否能够实现，而不能只考虑算法的正确性。

用户提示。

下面是一个数学问题的 Lean 4 源文件内容。原始 Lean 文件开始{Lean 源代码}原始 Lean 文件结束请生成最终完整的-Lean 4 源文件。你的任务是将每个标记“sorry”替换为正确的 Lean 代码。你不得修改文件的任何其他部分：“sorry”之外的所有行、导入、注释、名称和格式都必须保持完全相同。不得引入任何新的“sorry”或“submit”

仅输出原始 Lean 代码，不要使用 Markdown 代码围栏，也不要添加解释。

D.3. Pass@k 结果的无偏估计

我们使用无偏估计量报告 Pass@k 指标（Chen，2021）。若仅生成 k 个样本并检查是否至少有一个正确，则评估 Pass@k 会产生较高方差。为解决这一问题，我们生成一个更大的样本池，共 n 个样本（$n \geq k$），并针对每个问题估计如下概率：若抽取 k 个来自该池的样本（不放回抽样），则至少有一个样本正确。令 n 为针对某一问题生成的样本总数，令 c 为其中的正确样本数。Pass@k 的无偏估计量计算如下：

$$\text{Pass@}k := \mathbb{E}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{4}$$

其中 $\binom{n}{k}$ 表示二项式系数。在我们的实验中，我们设定 $k = 1$、2、4、8、16，总样本量为 $n = 16$。当 $n = k$ 时，该公式简化为经验准确率（若 $c > 0$ 则为 1，否则为 0）。对于 $n - c < k$ 的情形，项 $\binom{n-c}{k}$ 定义为 0，从而得到 Pass@k 为 1。

表6 展示了所有受评模型的详细 Pass@k统计数据（$k = \{1, 2, 4, 8, 16\}$）。这些指标使用上述无偏估计量计算，依据的是每个问题的 $n = 16$ 个样本。

表 *6.* 所有模型在完整基准测试上的 Pass@k（$k = 1$、2、4、8、16）统计数据。这些数值表示在给定 k 预算的情况下解决问题的百分比。

模型	Pass@1	Pass@2	Pass@4	Pass@8	Pass@16
通用推理模型					
DeepSeek-V3.2-Thinking	1.68%	2.71%	4.04%	4.86%	6.00%
GPT-5.1	1.40%	2.27%	3.27%	4.20%	4.50%
Gemini-3.0 Pro	2.62%	3.52%	4.46%	4.41%	6.00%
专用定理证明器					
Goedel Prover-32B	0.97%	1.70%	2.66%	3.50%	4.00%
Kimina Prover-72B	1.03%	1.65%	2.51%	3.50%	4.00%

D.4. 提示消融研究

为了严格评估提示工程对模型性能的影响，我们针对从优化和凸分析领域中选取的 20 个专业问题子集开展了有针对性的消融研究。本研究旨在确定，我们的构造—验证框架中的结构约束对于应用数学推理是否必不可少，或者标准形式化提示是否已经足够。我们比较了提供给模型的三种不同提示策略。

- **构造—验证**。 模型必须先通过 def 块定义一个可计算的解项，然后在定理块中证明其正确性。
- **标准代码补全**。 模型会收到一个带有 sorry 占位符的 Lean 文件，但没有明确的结构指引，也未强调两阶段工作流。
- **传统纯证明**。 模型的任务仅是证明给定的陈述，无需定义明确的解对象。

如表 3所示，构造—验证提示词取得了最高成功率，而另外两种策略则显著误导了模型。我们的分析表明，与我们的结构化方法相比，传统纯证明提示词往往会导致有效性降低 40%。如果没有明确要求进行构造，模型常常会采用非构造性策略，例如`Classical.choice` 或 apply is compact.exists min。尽管这些方法

在逻辑上证明了存在性定理，却不具备计算价值，因而无法实现应用数学的核心目标。此外，缺乏结构性约束的标准补全提示词常常会导致“策略过拟合”：模型完全忽略 def 块，试图仅凭证明搜索策略来完成目标。这些结果证实，显式的构造—验证代表了一种独特范式，需要量身定制的形式化接口，以防止模型绕过对解进行严格推导的过程。

E. 分析后的指令遵循能力详情

我们在实验中发现一个违反直觉的现象：模型在形式化证明方面的专精程度与其在应用数学任务上的表现呈负相关。如表 1 所示，专用模型在标准定理证明任务上的表现与通用大语言模型相当，甚至更胜一筹。然而，它们在算法设计和表示变换任务上的能力却急剧下降。我们将这一现象称为“策略过拟合”效应。专用模型经过大量基于状态-策略对的微调，这虽然最大限度地提升了它们完成逻辑目标的能力，但似乎也损害了它们在应用数学中至关重要的两方面指令遵循能力。

问题结构的刚性。 应用数学问题通常具有第 3 节所述的复杂结构，智能体必须先综合出一个构造性定义，再证明其性质。通用模型保留了基础预训练所赋予的强大指令遵循能力，因此能够遵循这一结构要求。相比之下，专用模型往往会忽略定义数据结构的指令，转而试图在单个 定理 块中解决整个问题。这种“锤子钉子”偏见——将每个提示都视为纯粹的定理证明任务——会导致它们在构造性基准测试中不合格。

应用数学直觉的丧失。 通用模型在将非形式化的数学概念转化为形式化规范方面展现出更强的能力。例如，当被要求对半定规划中的松弛进行形式化时，通用模型往往能正确推断出所需的矩阵不等式。专用模型尽管在形式化之后处理代数表达式方面更为擅长，却难以根据自然语言描述生成初始的形式化定义。它们实际上缺乏弥合非形式化意图与形式语法之间鸿沟的常识，而这种能力很可能在针对纯证明语料库进行密集微调的过程中被削弱。因此，尽管专用证明器在独立的逻辑验证步骤中仍然更胜一筹，却缺乏应用数学家所需的灵活性，无法胜任其高度依赖结构综合和跨领域建模的端到端工作流。

我们对模型行为的考察揭示，不同架构在处理应用数学形式化的多阶段要求时存在显著差异。专用证明器虽然擅长独立的逻辑演绎，却常常无法完成端到端的构造性工作流。

E.1. 策略过拟合现象

我们将专用模型的性能下降概括为“策略过拟合”。当模型仅针对证明状态-策略对进行微调时，就会出现这种现象：模型倾向于完成证明目标，却以牺牲结构综合为代价。

在复杂的结构化推理任务中，可以从 Kimina-Prover 等专用模型的行为观察到“策略过拟合”效应的一个显著例子。在下面的实例中，模型试图使用 intro 策略来调用 let 语句中已建立的前置变量；这在传统的声明式证明中是一种普遍做法，但在这一特定上下文中却属于结构性错误。这一错误凸显了模型对情境缺乏关键的感知能力；由于模型一直在针对状态-策略对进行高强度微调，因此它会默认采用高频的证明搜索例程，却无法正确识别形式化环境或现有的变量作用域。

这类失败体现了一种更普遍的“锤子与钉子”偏见：专用证明器把每个提示都当成纯粹的定理证明任务，实际上削弱了弥合非形式化数学意图与形式语法之间鸿沟所必需的常识性直觉。就像只会用锤子的熟练木匠一样，模型把每个结构性挑战都视为一颗要用证明策略敲进去的钉子，最终缺乏应用数学家完成端到端工作流程所需的灵活性。因此，尽管专用证明器在完成特定演绎步骤方面仍然更胜一筹，却无法满足通用推理模型能够更准确地应对的多阶段构造性要求。

```
变量 (n m : ℕ) [非零 n] [非零 m] (A : 矩阵 (有限类型 m) (有限类型 n) ℝ) (b : 有限类型 m → ℝ) (x : 有限类型 n →
    ℝ)

不可计算的 定义P: 优化问题 where
 n_变量 := n
 n_方程 := 0
 方程_约束:= 0
 n_不等式组 := m
 不等式_约束 := 函数 x => A *ᵥ x - b
 目标函数 := 函数 x => -Σ i, 对数 ((b - A *ᵥ x) i)

定义 Q : 对偶问题(PnmAb) : =
 {
  对偶_目标函数: = 函数(lam: 有限类型 0 → ℝ) (mu : 有限类型 m → ℝ) =>
   如果 h : ∀ i : 有限类型 m, mui ≥ 0 则
    扩展实数.实数嵌入 (- Σ i : 有限类型 m, (对数 (mu i) + 1))
   else
    ⊥,
  对偶_领域 := 集合.filter (fun (p : (有限类型 0 → ℝ) ×(有限类型 m → ℝ)) =>
   令 (λ, μ) : = p
   (∀ i: 有限类型 m, μ i ≥ 0)
  ) (集合.univ : 集合 ((有限类型 0 → ℝ) × (有限类型 m → ℝ))),
  h_目标函数 := by
   引入 lam mu
   simp [P、扩展实数、扩展实数.实数嵌入]
   所有_目标, 尝试 {
    Aesop
   }
  ,
  h_领域 := 通过
   外延性 ⟨lam, mu⟩
   simp [P, 扩展实数, 扩展实数.实数嵌入]
   所有_目标都尝试 {
    Aesop
   }
 }
```

E.2. 应用数学直觉的丧失

应用数学家的一项关键能力，是识别底层数学结构并将其转化为易于处理的形式，例如从原始问题表述推导出对偶问题。我们的实验表明，专用定理证明器往往缺乏建模直觉，无法准确完成这种转换。虽然这些模型在目标确立后擅长代数运算，却难以弥合非形式化数学意图与形式语法之间的鸿沟。

当 Kimina-Prover 被要求推导对偶问题时，其行为中出现了一种典型的失败。模型没有正确构造拉格朗日函数，也没有识别恰当的对偶变量和约束，而是试图给出一个从根本上偏离问题意图的解。模型没有评估对偶空间的结构要求，而是默认生成看似正确、实则逻辑错误的定义，模仿常见的证明模式。

相比之下，通用模型在从自然语言描述中推断必要的矩阵不等式和结构约束方面表现出更强的能力。本案例研究强调，应用数学的端到端工作流高度依赖结构综合和跨领域建模，而这对于那些仅针对逻辑验证进行过度优化的模型而言，仍是一项重大挑战。

```
/-
为以下问题推导一个对偶问题
\[
\begin{array}{ll}
\text{最小化} \quad &\sum_{i=1}^{N} \|y_i\|_2 + (1/2)\|x - x_0\|_2^2. \\
\text{约束条件} & y_i = A_ix+b_i, i=1, 2, \cdots n
\end{array}
\]
问题数据为 \( A_i \in \mathbb{R}^{m_i \times n}, b_i \in \mathbb{R}^{m_i} \) , and \( x_0 \in \mathbb{R}^n \).
-/
不可计算的 定义P: 优化问题 where
  ···--省略 P 的详细定义

定义 Q : 对偶问题 (P n m k A b x0):= 通过
 对偶_目标函数:= 函数lam _ =>
  若 h : ∀ i : 有限类型 k, 平方根 ((b i) ·ᵥ (b i)) ≤ 1 则
   扩展实数.实数嵌入 (Σ i : 有限类型 k, (b i) ·ᵥ (b i) + (A i *ᵥ x0) ·ᵥ (b i) – (1 / 2) * ((Σ i : 有限类型 k, (A i : 矩阵
    (有限类型 m) (有限类型 n) ℝ) ^* * (b i)) ·ᵥ (Σ i : 有限类型 k,(A i : 矩阵 (有限类型 m)(有限类型 n) ℝ) ^* * (b i))) : ℝ)
   --这里的对偶目标有误。
  else
   ⊥
 ···--省略 Q 的详细定义
```

F. 错误分析详情

在本节中，w 我们提供了全面的案例研究，以展示 Lean 4 编译器给出的错误信息。

F.1. 大语言模型幻觉。

大语言模型在形式化推理场景中经常会幻觉出不存在的定理或定义。下面给出一个 Lean 4 中的典型示例：模型试图调用一个不存在的定理 convexHull eq sInter 来完成证明，并自信地将其视为有效结果。这种行为表明，模型并非在形式系统本身中进行推理，而是依赖于从先前见过的数学文本中记忆下来的模式。因此，当前大语言模型表面上的定理证明能力不太可能真正具备迁移性，因为它们无法泛化到训练期间未曾明确接触过的领域、库或形式化内容。

```
--证明集合 \( S \) 的凸包等于--所有包含 \( S \) 的凸集的交集。 定理 num_4_P_E (
n : ℕ) [非零 n](C : 集合 (有限类型 n → ℝ)) :∩ (S : 集合 (有限类型 n → ℝ)) (_ : C ⊆
S) (_ : 凸性 ℝS) , S =convexHull ℝ C := by exact ( ~~~~~~~~~~~~~~~~~~
convexHull eq sInter C).symm
```

F.2. 形式化能力。

```
变量 (n : ℕ) [非零 n](a : 有限类型 n → ℝ) (b1 b2 : ℝ) 不可计算的 定义 answer
(n : ℕ) [非零 n](a : 有限类型 n → ℝ) (b1 b2 : ℝ): ℝ :=|b1 – b2| / ||a||引理 toLp2_
范数_等于 (v : 有限类型 n → ℝ) : ||toLp 2 v|| = ||v|| :=(EuclideanSpace.equiv 2 (
有限类型 n) ℝ).symm.范数_map v 定理 num_5_S_E (h : a ≠ 0): let A := {x | a ·ᵥ
x = b1}
```

```
设 B := {x | a ·ᵥ x = b2}
∀ x y, A x → B y → ‖toLp 2 (x -y)‖ ≥ answer n a b1 b2 ∧
∃ x y, A x ∧ By ∧ ‖toLp2(x - y)‖ =answer n a b1 b2 := by
引入 A B x y hx hy
有 hA : A x := hx
证明 hB : B y := hy
 simp rw [集合.mem setOf eq] at hA hB
证明 h1 : a ·ᵥ (x -y) = b1 -b2 := by
 rw [inner_sub_right, hA, hB, sub_sub_cancel]
证明 hnorm_pos : ‖a‖ > 0 := norm_pos_当且仅当。mpr h
构造器
· 有 h3 : |b1 - b2| ≤ ‖a‖ * ‖x - y‖ := 由
  rw [← 绝对值_内积_小于等于_范数 a (x -y)]
  精确 congr_arg abs h1 ▸ le_refl_
 rw [toLp2_范数_等式, answer]
 精确 (div_le_div_right hnorm_pos).mp h3
· 设置 x0 : = (b1/ ‖a‖ ^ 2) ·a
 集合 y0 := (b2/ ‖a‖ ^2) ·a
 有 hx0 : Ax0 := 由
  simp [A, Set.mem_setOf_eq, x0]
  rw [inner_smul_right, inner_self_eq_norm_sq, mul_div_cancel']
  exact pow_ne_zero 2 (norm_ne_zero_iff.mpr h)
 有 hy0 : By0 := 由
  simp [B, 集合.mem_setOf_eq, y0]
  rw [inner_smul_right, inner_self_eq_norm_sq, mul_div_cancel']
  exact pow_ne_zero 2 (norm_ne_zero_iff.mpr h)
 有 hnorm : ‖toLp 2 (x0 -y0)‖ = answer n a b1 b2 := 通过
  rw [toLp2_norm_eq, answer]
  calc
   ‖x0 - y0‖ = ‖((b1 - b2) / ‖a‖ ^ 2) · a‖ :=通过 simp [x0, y0, sub_smul, smul_sub]
   _ = |(b1 - b2) / ‖a‖ ^ 2| * ‖a‖ := 范数_数乘 _ _
   _ = |b1 - b2| / ‖a‖ ^ 2 * ‖a‖ := 通过 rw [abs_除法, abs_在_正 (幂_正 hnorm_正 2)]
   _ = |b1 - b2|/ ‖a‖:=通过field_simp [ne_在_gt hnorm_正]; 环归约
 exact ⟨x0, y0, hx0, hy0, hnorm⟩
```

形式化能力不足的典型表现之一，是高亮行中的错误“simp 未取得进展”。这里，模型对假设 hA 和 hB 应用简化策略 simp rw，隐含地假定该策略会展开集合隶属关系并得到可用的等式。然而，该策略未能以任何有意义的方式简化目标，表明没有触发任何适用的重写规则。这说明模型错误地预判了 simp 策略的效果，把它当作一种通用的推进工具，而没有推理其确切的前提条件和重写行为。这类错误表明，模型并未具体理解策略在证明状态层面上的运作方式，而是依据表面关联来选择策略，而非正确把握其形式语义。

F.3. 不完整输出。

形式化能力不足的一个密切相关表现是，模型频繁依赖诸如 sorry、submit 之类的特殊占位符，甚至引入新的公理，迫使不完整的证明通过编译器检查。如例子所示，模型成功构造了高层次的数学对象，并勾勒出预期的证明结构，但在遇到技术要求较高的论证时——例如验证最优性条件或完成归纳推理——便放弃了形式推导。模型没有解决剩余的证明义务，而是诉诸这些逃避机制；它们只是在形式上掩盖尚未解决的目标，而非完成这些目标。这种行为表明，模型将证明助手视为一个可以通过强行操作使其接受部分解答的宽松环境，而不是一个强制确保逻辑完备性的系统。因此，尽管生成的代码在语法上看似有效，却并不构成真正经过机器检验的证明，这进一步凸显了模型的概念推理能力与其执行精确、底层形式化证明的能力之间的差距。

```
变量 (n : ℕ) [非零 n](X : 矩阵 (有限类型 n) (有限类型 n) ℝ) (hX : X.正定) 定义 answer (n : ℕ) [非零 n]
(X : 矩阵 (有限类型 n) (有限类型 n) ℝ) (hX : X.正定) : 矩阵 (有限类型 n) (有限类型 n) ℝ :=令 ⟨P, D, hD_
diag, hPD, hPorth, hPorth'⟩ :=hX.1.正交对角化 令 Λ : 矩阵 (有限类型 n) (有限类型 n) ℝ := 矩阵.对角矩
阵 (fun i => (D i i + 实平方根 ((D i i)^2 + 4)) / 2) P * Λ * P⊤定理 num_43_S_H(n : ℕ) [非零 n] (X : 矩阵
(有限类型 n) (有限类型 n) ℝ) (hX : X.正定) : 令 answer := answer n X hX IsMinOn (fun U => –对数
(det U) + ((X – U) * (X – U)).迹 / 2) .univ answer := by intro answer~~~~~~sorry
```

F.4. 格式与语法。

此类别涵盖因输出格式不当或基本语法错误而导致的问题，这些问题会阻止编译器正确解析生成的文件。典型情况包括未能将代码置于适当的 Markdown 或代码块中、生成格式错误的章节标题或环境标题，以及引入缺失或不匹配的分隔符，例如圆括号、方括号或花括号。在这些情况下，生成的内容在数学上可能有意义，甚至单独来看在形式上也是正确的，但由于违反了证明助手或文档系统规定的表层结构要求，仍无法处理。尽管这些错误相对浅层，并不意味着缺乏数学理解，但它们仍表明模型未能始终内化或遵循形式化验证场景所要求的严格语法和格式规范。因此，即使呈现层面上的细微偏差，也可能导致编译彻底失败，这凸显了自动化形式化流程在底层输出正确性方面的脆弱性。

G. LLM 的使用

我们将大语言模型作为研究对象进行评估，以考察其在我们提出的框架中的形式化能力；同时也使用大语言模型来润色手稿的语法并提升其清晰度。