通过 LEAN 中的上下文证明学习发现新定理

A PREPRINT

Kazumi Kasaura
OMRON SINIC X Corporation
AutoRes
kazumi.kasaura@sinicx.com

Naoto Onda
OMRON SINIC X Corporation
NexaScience
AutoRes
naoto.onda@sinicx.com

Yuta Oriike
CyberAgent
AutoRes
oriike_yuta_xa@cyberagent.co.jp

Masaya Taniguchi
RIKEN AIP
AutoRes
masaya.taniguchi@riken.jp

Akiyoshi Sannai
Kyoto University
RIKEN AGIS
Shiga University
NII
NISTEP
AutoRes
sannai.akiyoshi.7z@kyoto-u.ac.jp

Sho Sonoda
RIKEN AIP
CyberAgent
AutoRes
sho.sonoda@riken.jp

arxiv:2509.14274v1 中译本

ABSTRACT

大型语言模型在形式化定理证明中表现出显著的潜力。然而，之前的工作主要集中在解决现有问题上。本文重点研究了 LLMs 发现新定理的能力。我们提出了一个自动生成数学猜想并以 Lean 4 格式进行证明的猜想-证明循环管道。我们的方法的一个特点是，在包括先前生成的定理及其证明在内的上下文中生成和证明进一步的猜想，这使得通过上下文学习证明策略而不改变 LLMs 参数的情况下能够生成更难的证明。我们展示了我们的框架通过验证重新发现了过去的数学论文中发表且尚未形式化的定理。此外，其中至少一个定理即使在自然语言中也无法被不使用上下文学习的 LLM 证明，这意味着上下文学习对神经网络定理证明是有效的。

源代码可在 https://github.com/auto-res/ConjecturingProvingLoop 获得。

1 介绍

大型语言模型（LLMs）在定理证明方面展示了巨大的潜力。由于 LLMs 可能会产生幻觉，且难以在自然语言中检测到这种现象，因此使用 LLM 生成形式证明并利用交互式定理证明器（ITP），如 Lean[1]，进行验证的研究受到了关注。然而，先前的工作主要集中在解决现有问题上。本文则侧重于研究 LLMs 发现新定理的能力。

我们提出了猜想-证明循环，一个用于自动生成数学猜想并在 Lean 4 格式下证明它们的管道。通过将猜想和证明阶段分开，我们避免了生成定理的收敛，并鼓励证明更难的定理。

[1] https://lean-lang.org/

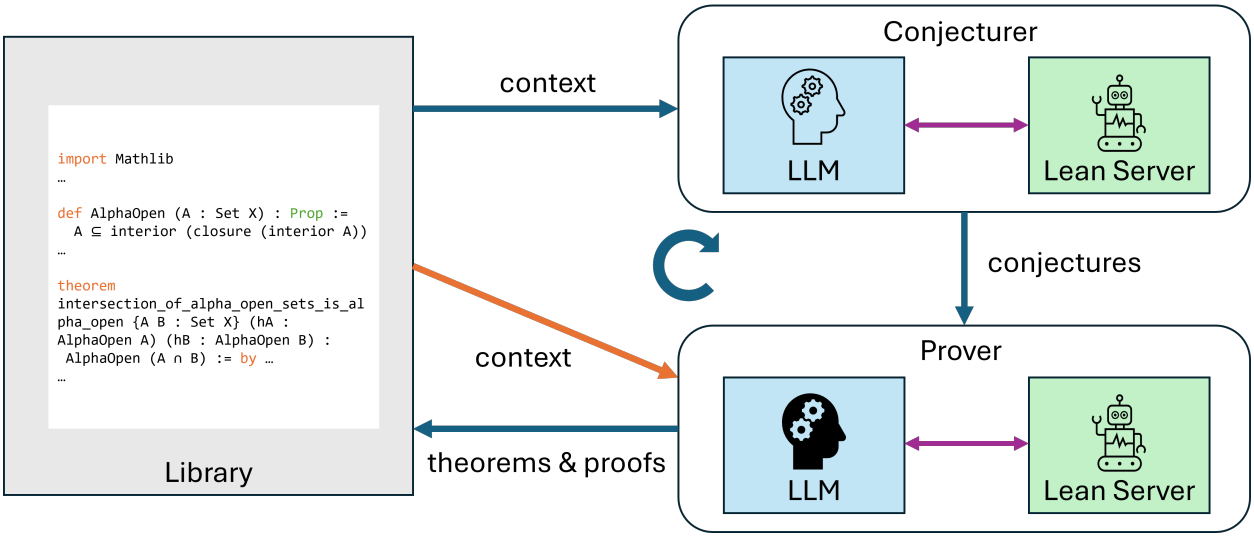

图 1: 猜想-证明循环概述。猜想者使用库作为上下文生成猜想，证明者尝试证明这些猜想，并且已证实的猜想及其证明被存储在库中作为定理。库还为证明者提供上下文。猜想者和证明者的流程都包括大型语言模型与 Lean 服务器之间的交互。

我们方法的一个特点是生成并证明包含已证定理及其证明的上下文中的进一步定理，这使得通过上下文学习证明策略而不训练大语言模型即可生成更难的证明。由于闭源大语言模型（如 chatGPT）在推理和生成 Lean 代码方面的能力最近得到了提高，我们将其用作猜想者和验证者。使用闭源大语言模型的缺点是无法自由训练这些模型，但在我们的框架中，可以通过从先前验证过的证明进行上下文学习来提高大语言模型的证明能力。

在实验中，当给定数学概念作为种子时，我们验证了这些概念的重要性质是否可以在我们的框架中被重新发现。更具体地说，我们关注了一些仅由 Mathlib[2] 中的概念定义的拓扑概念，但这些概念并未包含在 Mathlib 中。我们使用我们的框架生成了关于这些概念的定理。结果表明，我们的框架能够重新发现已发表在数学论文中的重要定理。此外，我们验证了基于上下文的学习证明策略的有效性：一个没有上下文甚至自然语言也无法被 LLMs 证明的定理，在我们的框架生成的上下文中得到了证明。

总结而言，本文的贡献如下。首先，我们提出了猜想与证明循环，这是一种用于自动生成数学猜想并在 Lean 4 格式下进行证明的流水线。其次，我们展示了我们的框架能够重新发现一个研究级别的定理。第三，我们验证了当通过 LLMs 自身生成的已验证证明作为上下文时，可以通过上下文学习提高 LLMs 的证明能力。

我们的工作还表明，使用人工智能自动扩展形式化数学库的潜力。形式化的数学只是用自然语言表达的一部分数学内容，而扩大诸如 Mathlib 这样的形式化图书馆对于验证和自动化数学至关重要。另一方面，应该包含在图书馆中的命题集可能并不总是可以通过自然语言获得的。我们的框架可以在学习给定概念的同时生成有关这些概念的命题。

2 相关工作

有一些工作使用大语言模型进行数学推理，既包括自然语言 [1]，也包括形式化的 ITP 语言 [11, 7]。它们主要集中在解决现有问题，并采用监督微调（SFT）和/或具有验证奖励的强化学习（RLVR）来提高大语言模型的问题解决能力。为克服训练数据集有限的问题，一些先前的工作也提出了通过 AI 生成待解决问题的方法 [2, 6, 14]。这些工作在以下两个方面与我们的方法有所不同：首先，我们的方法专注于生成和证明有意义

[2] https://github.com/leanprover-community/mathlib4

的定理，而那些工作的重点是训练大语言模型作为证明器。其次，虽然这些工作基于强化学习，但我们的方法基于上下文学习，可以应用于闭源的大语言模型。

若干研究表明，在提示中包含适当的问答示例可以提高大语言模型的数学推理能力 [13, 15, 4] 。虽然这些研究使用手工制作的例子或从数据库中搜索的结果作为上下文学习来源，但我们的框架使用来自大语言模型自身的输出，就像上下文强化学习一样 [9] 。利用 ITP 反馈生成形式证明的技术被提出 [5, 12, 8] ，我们也采用了该技术（见 § 3.3）。然而，我们在这里强调的是从其他命题的验证证明中学习策略。

3 方法

3.1 管道概述

图 1 说明了我们的框架。猜想-证明循环（CPL）由四个主要部分组成：猜想者（LLM 代理）、证明者（LLM 代理）、Lean 服务器和库（Lean 代码数据）。首先，用户初始化库。

1. 猜想者基于库生成以 Lean 4 格式表示的数学猜想，并通过访问 Lean 服务器来检查所生成猜想的语法正确性和新颖性。
2. 对于每个生成的猜想，证明器会生成这些猜想的证明，并访问 Lean 服务器来验证生成的证明。在这个步骤中，库也被用作上下文。
3. 验证过的猜想对及其证明被添加到图书馆中。我们回到第一步。

关于第二和第三步的详细内容，请参见以下小节。

通过分离猜想和证明阶段，我们避免了生成定理的收敛，并鼓励证明更难的定理。

将图书馆喂给猜想生成器作为上下文的目的在于防止产生重复的猜想以及通过类比已证明的定理来生成新的猜想。

将库馈送到证明器作为上下文的目的在于使已证明的定理在证明过程中可用，并通过上下文学习来学习证明策略。

3.2 猜想环路

为了生成各种猜想，在每个猜想步骤中，我们使用以下过程。

1. 猜想列表初始化为空。
2. 猜想者 LLM 根据图书馆和猜想列表生成猜想。
3. 对于每个生成的猜想，Lean 服务器检查该猜想是否在语法上有效且新颖。经过验证的猜想将被添加到猜想列表中。
4. 我们回到第二步直到达到预定义的迭代次数。

通过反复调用猜想生成器并在一步中产生许多猜想，我们诱导生成偏离图书馆的猜想，从而防止猜想收敛。

猜想的新颖性检查是通过在 Lean 中使用 `精确 ?` 命令来完成的，该命令检查猜想是否可以通过上下文中的现有定理进行证明。请注意，此命令是在导入整个 Mathlib4（Lean4 标准库）并包含该库和猜想者列表的情况下执行的。因此，经过检查的新颖性意味着该猜想不在 Mathlib4、已生成的库以及猜想者列表中。

给出给猜想大语言模型的系统提示如下：

> 你是一名 mathlib4 库的编写者。请根据给定的库生成新的猜想，格式为 Lean 4 格式，这些猜想不一定必须是真实的。不要生成列表中已有的陈述。不包含证明、注释或导入内容。新定

> 理以 'theorem' 开头，没有任何注释。它们应以 ':= sorry' 结束。此外，请使用标准的数学符号（例如，∀,∃,√）而不是 Unicode 转义序列（例如，\u2200）。

在实验中，迭代次数设置为 16。

3.3 证明循环

对于每个生成的猜想，证明器尝试按照以下过程进行证明。

1. 证明器 LLM 生成了猜想的证明代码。如果 LLM 判断该猜想不可证，则证明器以失败状态退出循环。
2. Lean 服务器验证生成的证明。如果证明被验证通过，证明者以成功状态退出循环。
3. 如果已经进行了足够的试验，证明者将以失败退出循环。否则，错误消息将返回给大语言模型，并回到第一步。

上下文同时提供给证明者和 Lean 服务器。因此，不仅证明者可以从上下文中学习证明策略，还可以将上下文中的定理用作引理。

给证明器 LLM 的系统提示如下：

> 你是一名 mathlib4 库的证明者。请用 Lean 4 格式证明给定内容中的最后一个定理。你应该编写直接跟在最后一个定理的 ":=" 后面的 Lean4 代码。它应该以 'by' 开始或直接表示一个项。你可以使用给定内容中的定理作为引理。证明中不要使用 sorry。如果你判断该定理不可证，请返回空字符串而不是证明。不要包含任何其他文本。

在实验中，最大试验次数设置为 16。

4 实验

我们证明了研究级别的定理可以被我们的框架重新发现，并验证了上下文学习在我们的框架中有效工作。

这些实验的脚本和生成的库存储在 `https://github.com/auto-res/ConjecturingProvingLoop` 中。

4.1 设置

在我们的实验中，我们关注一般拓扑学中的次要概念：半开性、alpha 开性和预开性。我们使用了以下包含这些概念定义的文件作为初始库，文件格式为 Lean 4 格式。

```
import Mathlib
import Aesop
namespace Topology

variable {X : Type*} [TopologicalSpace X]

/-- A set is semi-open if it is a subset of the closure of its interior. -/
def SemiOpen (A : Set X) : Prop :=
  A ⊆ closure (interior A)

/-- A set is alpha-open if it is a subset of the interior of the closure of its interior.
def AlphaOpen (A : Set X) : Prop :=
```

```
  A    interior (closure (interior A))

/-- A set is preopen if it is a subset of the interior of its closure. -/
def PreOpen (A : Set X) : Prop :=
  A    interior (closure A)
```

选择这些概念的原因是，它们可以仅用 Mathlib 中已有的概念来定义，而它们自身尚未包含在 Mathlib 中，并且尽管它们的数学重要性已经被认可并进行了研究，但它们目前还没有足够广为人知以至于 LLM 能够了解其性质。

我们关注以下定理是否能够生成：

> 两个 alpha-开集的交集是 alpha-开集 [10]。

该定理很重要，因为它是在证明 α-开集构成另一种拓扑（[10] 中的命题 2）时最难证明的部分。

我们使用我们的框架以及 30 循环生成了这些概念的定理。我们将 chatGPT-4o[3] 用作猜想者，将 chatGPT-o3[4] 用作证明者。

4.2 基线

为了进行比较，我们还使用了一个简单的循环框架生成了它们的定理及其证明，该框架由 LLM 一次性生成定理和其证明，并使用 chatGPT-3o。

首先，用户初始化库。与 CPL 不同，在这个简单的循环基线中，猜想者和证明者没有分离，单个循环如下：

1. 大型语言模型根据库生成 Lean 4 格式的声明及其证明，并访问 Lean 服务器进行验证。
2. 如果上一步成功，则生成的声明和证明对将被存储在库中。我们返回到第一步。

在实验中，我们执行了这个循环 400 次。

第一步类似于证明循环。其细节如下：

1. 大型语言模型在 Lean 中生成了一个陈述及其证明。
2. Lean 服务器验证生成的内容。如果验证成功，我们退出循环。
3. 如果已经完成了足够的试验，我们以失败退出循环。否则，错误消息返回给 LLM，我们回到第一步。

给大型语言模型的系统提示如下：

> 你是一名 mathlib4 库的编写者。请根据给定的库生成一个带有证明的新定理，格式为 Lean 4 格式。除了 Lean 4 代码外，不要返回任何其他内容。生成的代码应遵循给定的库。生成的代码应仅包含定理和证明。不要生成已存在于库中的定理。新定理应以 'theorem' 开头，不带任何注释。可以在证明中使用给定库中的定理作为引理。证明中不得使用 sorry。另外，请使用标准数学符号（例如，$\forall, \exists, \surd$）代替 Unicode 转义序列（例如，\u2200）。

如同证明循环中设置的最大试验次数为 16。

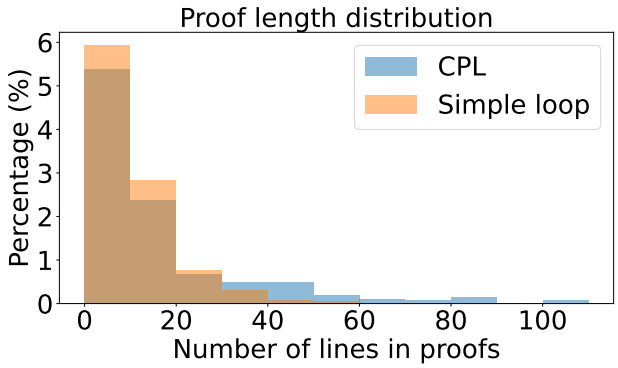

图 2: 由我们的框架和简单循环框架生成的定理证明长度的分布，其中箱体的大小为 10。

4.3 结果

我们为这些概念生成了 269 个定理，其中包括重点定理。该定理的生成证明见附录 A。此证明与 Njstad 的原始证明有所不同，这表明 LLM 自行找到了这个证明。其他在研究论文中显示的重要性质的生成定理包括以下内容：

- 半开集与 alpha-开集的交集是半开集 [10]。
- 预开集与 α-开集的交是预开集 [3]。

另一方面，这些定理没有包含在由简单循环框架生成的 359 个定理中。原因似乎是简单循环框架倾向于生成容易证明的定理。为了证实这一点，我们测量了我们的框架和简单循环框架生成的定理的证明长度。图 2 显示了我们的框架和简单循环框架生成的定理的证明长度分布。可以看出，我们的框架能够生成比简单循环框架更长证明的定理。

我们也验证了将生成的库作为上下文提供给证明器可以提高其证明能力。

4.4 提供上下文的有效性

4.4.1 重新证明生成的定理

首先，我们尝试通过 § 3.3 中的方法在两种设置下重新证明所有生成的定理：一种是上下文包含待证定理生成前产生的库的情况；另一种仅包含相关概念的定义。结果显示，在有上下文的设置中成功证明了 **99% 的定理（267 个定理）**，而在无上下文的设置中仅证明了 **90% 的定理（242 个定理）**。因此，上下文提升了 LLMs 的证明能力。

4.4.2 证明能力对于 Alpha-开交集而言

此外，我们在每种设置中尝试重新证明该焦点定理共 128 次。（生成此定理之前生成的定理数量为 49。）在 Lean 中评估 Prover 的过程与 prover 循环相同，只是系统提示已更改为如下：

> 你是一个 mathlib4 库的证明者。请用 Lean 4 格式证明给定内容中的最后一个定理。你应该编写直接跟随在最后一个定理的 ":=" 后面的 Lean4 代码。它应该以 'by' 开始或直接表示一个项。你可以使用给定内容中的定理作为引理。证明中不要使用 sorry。如果你判断该定理是错误的，请返回空字符串而不是证明。不要包含任何其他文本。

[3] `https://platform.openai.com/docs/models/gpt-4o`
[4] `https://platform.openai.com/docs/models/o3`

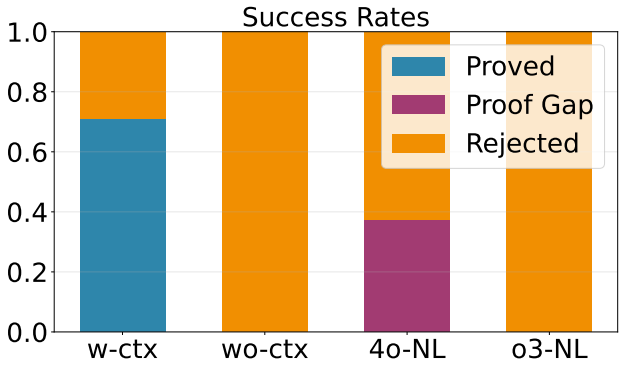

图 3: 大型语言模型通过 Lean（**w-上下文**，**w-上下文**）证明定理的成功率。带有和不带上下文，自然语言中的证明（**4 阶-非局部**，**o3-非本地**）。

请注意，不返回证明的条件从“不可证”更改为了“假”。

为了进行比较，我们也要求大语言模型（chatGPT-4o 和 chatGPT-o3）用自然语言（英语）证明这个定理，并包含了 16 次概念的定义。我们手动检查了这些响应。在自然语言实验中，我们使用了以下系统提示：

> 请证明下列定理。如果你判断该定理是错误的，请返回“错误”而不是证明。

给出大语言模型需要证明的陈述如下：

> 在一个拓扑空间中，一个集合是 α-开集如果它是其内部的闭包的内部的一个子集。任意两个 α-开集的交集是 α-开集。

在形式语言和自然语言设置中，我们提示大语言模型如果给定的定理是假的，则不要返回证明。因此，输出结果分为三类：正确证明、包含缺口的证明或被拒绝为不正确的。

结果如图 3 所示。结果显示该定理仅在使用生成的库的情况下被证明。大多数以自然语言做出的判断错误地认为该定理为假，这意味着发现的定理并未包含在 chatGPT 的知识中。chatGPT-4o 生成的带缺口的证明示例如附录 B 所示。ChatGPT-3o 在 Lean 和自然语言中从未生成过错误的证明，但它总是错误地判断该定理为假。

该定理的生成证明，如附录 A 中所示，并没有使用其他生成的定理作为引理。因此，生成的库被用于证明策略的上下文学习，而不是作为证明的一系列引理集合。

5 结论与未来工作

我们提出了猜想-证明循环，一个自动生成数学猜想并在 Lean 4 格式下证明它们的流程。我们展示了我们的框架可以重新发现一个研究水平的定理。我们也验证了在我们框架中的情境学习证明策略是有效的。

本研究中突出的命题很容易预测。未来的工作应集中在完善猜想生成过程，以产生更深刻和更具洞察力的数学陈述，可能通过采用引导 LLM 朝向未探索的数学理论领域的技术来实现。

致谢

本工作得到了以下支持：JST 月光科研计划 JPMJMS2236，JST BOOST JPMJBY24E2，JST CREST JPMJCR2015，JSPS 科研费 24K21316, 24K16077，以及先进科学通用智能项目（AGIS），RIKEN TRIP 计划。

参考文献

[1] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. DeepSeek-V3 technical report, 2025.

[2] Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving, 2025.

[3] Julian Dontchev. Survey on preopen sets. *arXiv preprint math/9810177*, 1998.

[4] Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119, 2022.

[5] Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1229–1241, 2023.

[6] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

[7] Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia LI, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving. In *Second Conference on Language Modeling*, 2025.

[8] Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025.

[9] Amir Moeini, Jiuqi Wang, Jacob Beck, Ethan Blaser, Shimon Whiteson, Rohan Chandra, and Shangtong Zhang. A survey of in-context reinforcement learning. *arXiv preprint arXiv:2502.07978*, 2025.

[10] Olav Njästad. On some classes of nearly open sets. *Pacific journal of mathematics*, 15(3):961–970, 1965.

[11] Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025.

[12] Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. *arXiv preprint arXiv:2310.04353*, 2023.

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[14] Shaoxiong Zhan, Yanlin Lai, Ziyu Lu, Dahua Lin, Ziqing Yang, and Fei Tang. Mathsmith: Towards extremely hard mathematical reasoning by forging synthetic problems with a reinforced policy. *arXiv preprint arXiv:2508.05592*, 2025.

[15] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A 生成的聚焦定理证明

```
theorem intersection_of_alpha_open_sets_is_alpha_open {A B : Set X} (hA : AlphaOpen A) (hB
    -- We unfold the definition of ‘AlphaOpen‘ and start with a point
    -- ‘x   A   B‘.
    intro x hx
    rcases hx with hxA, hxB
    -- From the alpha-openness of ‘A‘ and ‘B‘ we obtain the two interior
    -- memberships below.
    have hA_int : x   interior (closure (interior A)) := hA hxA
    have hB_int : x   interior (closure (interior B)) := hB hxB
    ------------------------------------------------------------------
    -- Consider the open neighbourhood
    --   ‘U := interior (closure (interior A))
    --           interior (closure (interior B))‘ of ‘x‘.
    ------------------------------------------------------------------
    have hU_open :
        IsOpen (interior (closure (interior A))
                interior (closure (interior B))) :=
      isOpen_interior.inter isOpen_interior
    have hxU :
        x   interior (closure (interior A))
            interior (closure (interior B)) := hA_int, hB_int
    ------------------------------------------------------------------
    -- We prove that ‘U   closure (interior (A   B))‘.
```

```
------------------------------------------------------------------
have hU_subset :
    (interior (closure (interior A))
      interior (closure (interior B)) : Set X)
      closure (interior (A   B)) := by
  intro y hyU
  rcases hyU with hyA_int, hyB_int
  -- From ‘hyA_int‘ and ‘hyB_int‘ we obtain the two closure
  -- memberships below.
  have hyA_cl : y   closure (interior A) :=
    (interior_subset :
        interior (closure (interior A))   closure (interior A)) hyA_int
  have hyB_cl : y   closure (interior B) :=
    (interior_subset :
        interior (closure (interior B))   closure (interior B)) hyB_int
  ------------------------------------------------------------------
  -- To show ‘y   closure (interior (A   B))‘
  -- we use the neighbourhood characterisation of the closure.
  ------------------------------------------------------------------
  apply (mem_closure_iff).2
  intro V hV hyV
  ------------------------------------------------------------------
  -- First step: work inside
  --   ‘V := V   interior (closure (interior B))‘.
  ------------------------------------------------------------------
  have hV_open :
      IsOpen (V   interior (closure (interior B))) :=
    hV.inter isOpen_interior
  have hyV : y   V   interior (closure (interior B)) :=
    hyV, hyB_int
  -- As ‘y   closure (interior A)‘, this neighbourhood meets
  -- ‘interior A‘.
  have h_nonempty :
      ((V   interior (closure (interior B)))   interior A).Nonempty :=
    (mem_closure_iff).1 hyA_cl _ hV_open hyV
  rcases h_nonempty with
    y , hyV, hyB_int, hyIntA
  ------------------------------------------------------------------
  -- Now ‘y   V   interior A‘ **and**
  -- ‘y   interior (closure (interior B))‘.
  -- Hence ‘y   closure (interior B)‘.
  ------------------------------------------------------------------
  have hy_clB : y   closure (interior B) :=
    (interior_subset :
        interior (closure (interior B))   closure (interior B)) hyB_int
```

```
  ------------------------------------------------------------
  -- Second step: work inside
  --   `V := V ∩ interior A`.
  ------------------------------------------------------------
  have hV_open : IsOpen (V ∩ interior A) :=
    hV.inter isOpen_interior
  have hyV : y ∈ V ∩ interior A := ⟨hyV, hyIntA⟩
  -- Because `y ∈ closure (interior B)`, the set `V` meets
  -- `interior B`.
  have h_nonempty :
      ((V ∩ interior A) ∩ interior B).Nonempty :=
    (mem_closure_iff).1 hy_clB _ hV_open ⟨hyV⟩
  rcases h_nonempty with
    ⟨z, ⟨hzV, hzIntA⟩, hzIntB⟩
  ------------------------------------------------------------
  -- The point `z` lies in `V`, `interior A`, and `interior B`;
  -- hence it belongs to `interior (A ∩ B)`.
  ------------------------------------------------------------
  have hzIntAB : z ∈ interior (A ∩ B) := by
    -- Use maximality of the interior to pass from
    -- `interior A ∩ interior B` to `interior (A ∩ B)`.
    have h_sub :
        (interior A ∩ interior B : Set X) ⊆ interior (A ∩ B) :=
      interior_maximal
        (by
          intro t ht
          rcases ht with ⟨htA, htB⟩
          exact ⟨interior_subset htA⟩ (interior_subset htB))
        (isOpen_interior.inter isOpen_interior)
    exact h_sub ⟨hzIntA, hzIntB⟩
  -- We have found a point in `V ∩ interior (A ∩ B)`.
  exact ⟨z, hzV, hzIntAB⟩
------------------------------------------------------------------
-- Maximality of the interior gives the desired inclusion,
-- and we conclude with the membership of `x` in `U`.
------------------------------------------------------------------
have hU_subset_int :
    (interior (closure (interior A)) ∩
      interior (closure (interior B)) : Set X) ⊆
        interior (closure (interior (A ∩ B))) :=
  interior_maximal hU_subset hU_open
exact hU_subset_int ⟨hxU⟩
```

B 生成的错误证明示例

以下是一个由 chatGPT-4o 生成的关于两个 alpha-开集的交集是 alpha-开集的“证明”示例。

> 为了证明给定的定理，我们首先定义在一个拓扑空间中集合是什么意思被称作 alpha-开集。
> 设 (X, τ) 是一个拓扑空间，并设 $A \subseteq X$。集合 A 被称为 α-开集如果
>
> $$A \subseteq \text{int}(\overline{\text{int}(A)}),$$
>
> 其中 $\text{int}(A)$ 表示 A 的内部，而 $\overline{\text{int}(A)}$ 表示 $\text{int}(A)$ 的闭包。
> 我们旨在证明任意两个阿尔法开集的交集是阿尔法开集。设 $A, B \subseteq X$ 是两个阿尔法开集；
> 因此，根据定义，我们有：
> 1. $A \subseteq \text{int}(\overline{\text{int}(A)})$ 2. $B \subseteq \text{int}(\overline{\text{int}(B)})$
> 我们需要证明 $A \cap B$ 也是 alpha-开的，即，
>
> $$A \cap B \subseteq \text{int}(\overline{\text{int}(A \cap B)}).$$
>
> 首先，观察到 $\text{int}(A \cap B) \subseteq \text{int}(A) \cap \text{int}(B)$。
> 闭包和内部运算都是单调的，因此：
>
> $$\overline{\text{int}(A \cap B)} \subseteq \overline{\text{int}(A) \cap \text{int}(B)} \subseteq \overline{\text{int}(A)} \cap \overline{\text{int}(B)}.$$
>
> 再次取内部得到：
>
> $$\text{int}(\overline{\text{int}(A \cap B)}) \subseteq \text{int}(\overline{\text{int}(A)} \cap \overline{\text{int}(B)}) \subseteq \text{int}(\overline{\text{int}(A)}) \cap \text{int}(\overline{\text{int}(B)}).$$
>
> 根据假设 $A \subseteq \text{int}(\overline{\text{int}(A)})$ 和 $B \subseteq \text{int}(\overline{\text{int}(B)})$，可得
>
> $$A \cap B \subseteq \text{int}(\overline{\text{int}(A)}) \cap \text{int}(\overline{\text{int}(B)}) \supset \text{int}(\overline{\text{int}(A \cap B)}).$$
>
> 因此，
>
> $$A \cap B \subseteq \text{int}(\overline{\text{int}(A \cap B)}).$$
>
> 这表明 $A \cap B$ 确实是 α-开集。因此，任意两个 α-开集的交集仍是 α-开集。
> 因此，该定理是正确的。

此证明显然存在一个漏洞，因为在从倒数第二个关系推导最后一个关系时，子集关系的方向被反转了。