超越记忆：测试大语言模型对未见过的计算理论任务的推理能力

Shlok Shelat
艾哈迈达巴德大学
古吉拉特邦，印度

Jay Raval艾哈迈达巴德大学 古吉拉特邦，印度

Souvik Roy艾哈迈达巴德大学 古吉拉特邦，印度

Manas Gaur马里兰大学巴尔的摩县分校，美国马里兰州巴尔的摩

arXiv:2601.13392v1 [cs.CL] 19 Jan 2026

摘要

大语言模型（LLM）在形式语言任务上表现出色，但这究竟反映了真正的符号推理能力，还是对熟悉构造的模式匹配，仍不清楚。我们推出了一项基准测试，用于评估从正则语言构造确定有限自动机（DFA）的能力，其中包含事实性知识问题、来自公开来源的已见构造问题，以及两类未见问题：包含多个相互作用约束的手工构造实例，以及通过 Arden 定理系统生成的问题。模型在事实性问题上准确率达到 100%，在已见任务上的准确率为 84% 至 90%。然而，在未见问题上的准确率大幅下降（下降 30 至 64 个百分点）；错误源于对语言约束的系统性误解、对 Kleene 星号语义的错误处理，以及未能保持全局一致性。我们评估了一种三阶段提示协议，该协议能够纠正浅层错误，但无法可靠地解决全局不一致或结构有缺陷的自动机。我们对多种提示策略（直接提示、思维链、思维树）的分析表明，无论采用哪种提示方法，错误都会持续存在，这揭示了大语言模型生成语法上看似合理的确定有限自动机的能力，与其进行语义正确的形式推理的能力之间存在根本差距。

1 引言

大语言模型（LLM）在各类推理基准测试中表现出色，涵盖数学问题求解（Lewkowycz 等人，2022年；Welleck 等人，2021年；Azerbayev 等人，2023年）到代码生成（Wu 等人，2022年）。然而，一个根本问题仍未解决：这些模型究竟是在进行真正的符号推理，还是主要依赖对记忆样例进行模式匹配？近期研究揭示了持续存在的缺陷，这些任务需要进行结构化的符号操作（Katz 等人，2025年；Yue 等人，2024年），这表明出色的基准测试表现未必反映了稳健的推理能力。

我们从根据正则语言构造确定有限自动机（*DFA*）这一视角来探讨这个问题。该问题是计算理论（ToC）中的核心问题。此外，这也体现了词法分析器的任务，而这类任务通常借助 Lex、Flex 等工具解决，但存在一定局限（Aho 等人，2006）。这项任务作为推理探针具有独特优势：（1）正确性可通过穷举测试进行形式化验证；（2）解决方案需要进行多步符号操作，并满足全局一致性约束；（3）可能的问题实例空间在组合意义上极为庞大；（4）确定有限自动机是最简单的非平凡计算模型，因此模型的失败不能归因于问题复杂性或规范含糊。我们聚焦于基于提示的评估（而非微调），以反映大语言模型的实际使用方式。值得注意的是，尽管现有的计算理论基准测试（Golesteanu 等人，2024年；Zahraei 和 Asgari，2024年）评估事实性知识和证明验证，却未系统地区分记忆与组合泛化；模型可能通过回忆训练数据中的相似问题而非基于第一性原理进行推理，从而取得成功。

为了单独考察真正的推理能力，我们设计了一个精心构建的基准测试，包含三个部分：(1) 用于检验是否掌握 DFA 定义和性质的知识检验数据集；(2) 由 90 道公开可用的 DFA 问题组成的已见构造数据集；(3) 包含 180 道新颖问题的未见构造数据集，这些问题通过两种互补的方法生成。第一种方法是数学艺术，即手动构造包含多个相互作用的约束、禁用子串和基于叙述的规范（例如，编码

国际象棋棋步）。第二种方法是，数学工程通过 Arden 定理系统地生成问题（ Sipser，2013 ），从而生成结构复杂且不太可能出现在训练数据中的正则表达式。已见/未见划分使我们能够衡量由记忆而非推理所导致的性能差距。

我们评估了前沿大语言模型：GPT-5.1、Gemini- 2.5-Flash 和 Grok-4.1-fast-reasoning，考察它们在多种提示策略下的表现，包括思维链（CoT）和思维树（ToT）（Wang 和 Zhou，2024年；Yao 等人，2023年）。我们还提出了一种三阶段提示协议，逐步揭示构造错误，使我们能够评估大语言模型在引导下能否自我纠正。

主要发现。 结果揭示了知识与推理之间存在显著脱节：所有模型在事实性问题上均达到 100% 的准确率，在已见构造任务上达到 84–90%，但在未见问题上的准确率则大幅下降（直接提示下为 20.67–59.12%，下降 30–64 个百分点）。详细的错误分析显示出系统性的失败模式：错误简化克林星语义、未能在连接时保留约束，以及引入多余状态。值得注意的是，这些失败在所有提示策略下均持续存在；提示协议主要纠正浅层错误，却无法纠正全局不一致的自动机。

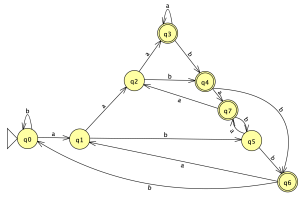
贡献。 本研究作出以下贡献：**(i) 全新基准测试：** 我们推出首个具有系统化已见/未见划分的确定有限自动机构造基准测试，包含 50 个知识问题、90 个已见问题和 180 个未见问题（其中 60 个为人工设计，120 个通过 Arden 定理生成）。**(ii) 受控记忆研究：** 通过评估结构相似的已见/未见问题对，我们首次证明，大语言模型在计算理论任务上的成功主要反映了记忆，而非组合推理。**(iii) 全面的提示工程评估：** 我们评估了思维链、思维树（采用四种构造方法：直接构造、最小化、基于导数的方法、Thompson算法），以及一种全新的基于提示的自我纠正协议。**(iv) 系统化失败分类：** 通过详细分析500+ 错误的确定有限自动机，我们识别出六种反复出现的失败模式（如导数规范化错误、约束组合失败等），揭示了符号状态跟踪方面的根本局限。所有数据集、提示词、评估代码和模型输出均已发布于 https://anonymous.4open.science/r/dfa-llm-evaluation-B82D/以支持可复现性，并推动大语言模型形式推理领域的未来研究。

2 已见数据集与任务形式化

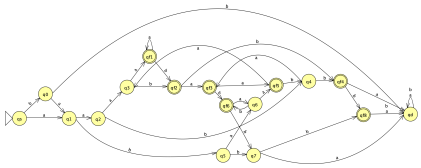
2.1 任务定义

我们评估大语言模型根据形式语言规范构造确定有限自动机（DFA）的能力。给定一个目标语言 L，其由正则表达式（RE）或自然语言描述指定，模型必须生成一个 DFA $D = \langle Q, \Sigma, \delta, q_0, F \rangle$，使其恰好识别 L。其中，Q 是有限状态集合，Σ 是输入字母表，$\delta : Q \times \Sigma \to Q$ 是转移函数，$q_0 \in Q$ 是初始状态，$F \subseteq Q$是接受状态集合（Sipser，2013）。如果存在状态序列 $s_0, s_1, \ldots, s_n$，满足（i）$s_0 = q_0$，（ii）对于所有 $1 \le i \le n$，$s_i = \delta(s_{i-1}, w_i)$，以及（iii）$s_n \in F$，则 DFA$D$ 接受字符串 $w = w_1 w_2 \cdots w_n \in \Sigma^*$。如果 $L = \{w \in \Sigma^* \mid D \text{ 接受 } w\}$，我们称 D 识别语言L。如果存在一个能够识别该语言的 DFA，则该语言是正则的。正确的 DFA 构造要求模型：（1）解析并解释形式语言规范；（2）确定能够捕获所有约束的最小状态结构；（3）设计转移，以确保接受全部且仅接受有效字符串；（4）在所有状态和符号之间保持全局一致性。至关重要的是，对于任意语言，都存在指数级数量的无效 DFA；而状态语义或转移指派中的微小错误，都可能使整个构造失效。

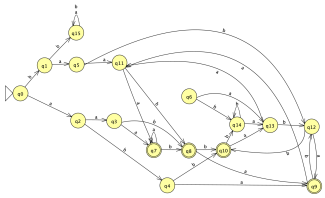
问题实例。 每个问题都以以下两种格式之一指定目标语言：*(a)* 自然语言：L_1 = {在 $\{a, b\}$ 上构造一个确定有限自动机，使其接受所有倒数第三个符号为 'a' 的字符串}。如果问题以自然语言格式给出，就无法使用 Lex、Flex（Aho 等人，2006）设计确定有限自动机。*(b)* 正则表达式：$L_1 = (a+b)^*a(a+b)(a+b)$。两种格式指定的是同一种语言（图1(a)）。模型必须输出完整的确定有限自动机规格，包括状态、转移、初始状态和接受状态。我们尽可能关注最小 确定有限自动机（状态数最少），因为最小化要求识别语义等价的状态——这是一项超越简单枚举有效转移的推理挑战。不过，在评估大语言模型的表现时，我们只考察确定有限自动机是否正确（尤其不要求是最小确定有限自动机）。



(a) L_1的正确最小确定有限自动机



(b) L_2的正确最小确定有限自动机



Gemini-2.5 对 L_2 的输出

图 1： (a)：语言 L_1 的最小确定有限自动机；(b) 和 (c)：未见语言 L_2 的正确最小确定有限自动机（见 b）与具有代表性的错误输出（见 c）的对比。符号说明： ◯ = 状态， ▷ = 初始状态， ◎ = 接受状态。转移 $q_0 \xrightarrow{a} q_1$ 表示在输入 a 时，确定有限自动机从状态 q_0 转移到状态 q_1（放大以获得更佳可读性）。

数据集	提示工程	GPT-5.1	Grok-4.1	Gemini-2.5
知识	零样本	100%	100%	100%
已见 DFA	零样本	84.2%	89.5%	85.0%

表 1：知识与已见 DFA 数据集的结果。

2.2 知识与已见 DFA 数据集

在评估构造能力之前，我们先验证模型是否具备自动机理论的基础知识。该数据集包含 50 道题：25 道选择题和 25 道简答题，涵盖确定有限自动机定义、正则表达式性质，以及确定有限自动机与非确定有限自动机（NFA）之间的关系。所有问题均选自标准计算理论教材和公开可用的材料。 **结果：** 所有模型的准确率均达到 100%（表 1），这证实它们完全掌握了相关定义和性质。因此，构造任务中的错误不能归因于缺乏概念知识。

已见数据集包含 90 个确定有限自动机构造问题，收集自在线大学习题集、教科书和可公开访问的计算理论资源。对于每个问题，我们都确认其题目陈述和解答中的确定有限自动机均可在线公开获取，以确保这些问题代表预训练期间可能遇到的问题模式。

语言特征。 已见数据集包含一些标准模式，例如：(i) 后缀/前缀约束（例如“以 ab结尾”）；(ii) 模 k 计数（例如“ a的数量为偶数”）；(iii) 基于位置的约束（例如“倒数第3^{rd}个符号是a”）；以及 (iv) 布尔组合（例如“包含 aa 或 bb”）。

评估协议。 我们使用零样本直接提示（不提供完整示例）评估模型。通过官方 API 查询模型，并将温度设置为 0，以进行确定性解码。每个模型都会收到问题规格说明，并须以结构化 JSON 格式输出完整的确定有限自动机，其中需指定状态、字母表、转移、初始状态和接受状态（见附录 D）。

结果。 表 1 显示，所有模型的表现都很出色：Grok-4.1-fast-reasoning 的成功率最高（89.5 %），其次是 Gemini-2.5-Flash（85.0%）和 GPT-5.1（84.2%）。对于示例语言 L_1，这三个模型都生成了正确的最小确定有限自动机。这些结果表明，模型能够成功地为熟悉的问题模式构造确定有限自动机。

记忆问题。 在已见问题上取得高准确率，并不意味着具备真正的推理能力。模型可能是从训练数据中检索相似示例，而非执行组合式符号操作，从而取得成功。为了将推理与记忆区分开来，接下来我们将介绍一个新颖性经过精心控制的未见构造数据集。

2.3 记忆问题

在已见数据集上的出色表现（84–90%的准确率）可能表明大语言模型具备稳健的 DFA 构造能力。然而，这一结论还为时过早：模型可能是通过检索记忆中的解题模式取得成功，而非进行真正的符号推理。要区分这两种解释，我们必须评估模型在结构上新颖的问题上的表现，这些问题要求组合泛化。**动机示例：** 考虑两种密切相关的语言： L_1 **（已见）：** 接受所有满足以下条件的字符串：在 $\{a, b\}$上定义，倒数第三个符号为‘a’； L_2 **（未见）：** 接受所有满足以下条件的字符串：在 $\{a, b\}$ 上定义，且（i）倒数第四个符号为‘a’，并且（ii）在任何‘a’之前都不会出现子串‘bb’。语言L_2 在 L_1 的基础上增加了一个约束（禁止在‘a’之前出现‘bb’），并对位置约束进行了小幅修改（4^{th}数位而非 3^{rd}数位）。两种语言都需要相似的推理——跟踪符号位置，同时满足顺序约束——

然而， L_2 的构造方式使其未出现在公开题集和教材解答中。

实证证据。 尽管在 L_1 上的准确率达到 100%，所有模型在 L_2 上使用直接提示时均告失败。图 1(c) 展示了 Gemini-2.5-Flash 的输出：虽然构造出的确定有限自动机接受了一些有效字符串（例如，“aaaa”“aaab”），但也接受了无效字符串，例如“aaa”（违反位置约束）和“aabbaba”（同时违反两个约束）。GPT-5.1 和 Grok-4.1-fast-reasoning 也出现了类似的系统性错误（见附录 A）。这一失误尤其值得注意，因为 L_1 和 L_2的差异仅在于约束的组合方式，而非根本的推理要求。模型能够为较简单的位置约束（训练中见过的约束）正确构造确定有限自动机，但在这些约束与顺序限制结合时却会失败——这表明模型在已见任务上的成功反映的是模式检索，而非稳健的符号推理。

研究问题。 这些观察促使我们提出核心研究问题： 大语言模型在形式推理任务上的表现，在多大程度上依赖于对训练示例的记忆，而非组合式符号推理？ 要回答这一问题，就需要在未见过的问题上进行系统评估，同时控制结构新颖性，并保留任务和推理要求。**未见数据集设计。** 为开展这种受控评估，我们采用两种互补方法构建未见确定有限自动机数据集： （i）**数学艺术（60 个问题）：** 人工设计的问题，包含多个相互作用的约束、禁用子串、位置模式和基于叙述的规范（例如，编码国际象棋开局、迷宫导航）。这些问题要求创造性地组合约束，而标准课程中并未涉及此类组合。（ii）**数学工程（120 个问题）：** 通过 Arden 定理系统生成问题（Sipser，2013）。我们构造随机非确定有限自动机，通过代数消元推导其接受语言，并将由此得到的正则表达式（通常具有高度嵌套且非标准的结构）用作问题规范。这种方法确保了结构的多样性和可扩展性。

所有未见过的问题均经过人工核验，确认未出现在公开的问题库、教科书和在线课程材料中。我们根据约束数量（艺术）或嵌套深度（工程）对问题难度进行分类（简单、中等、困难），以便细致分析推理在哪些方面会失效。详细的构造过程和代表性示例见第 3节。

评估框架。 除了衡量模型在未见问题上的准确率外，我们还评估了多种提示策略（思维链、包含四个构造分支的思维树），并引入三阶段提示协议来评估自我纠正能力。这项全面评估旨在区分失败究竟源于最初的误解、无法保持符号一致性，还是即使得到纠正性指导也依然存在的根本性推理缺陷。

3 未见确定有限自动机构造数据集

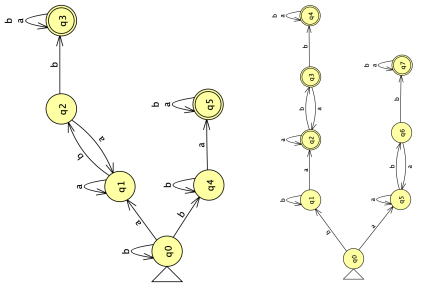
我们采用两种生成策略：手动约束组合 （60 个问题），用于创造性地设计问题；以及系统性地逆用 *Arden* 定理 （120 个问题），用于大规模生成并确保结构多样性。

3.1 手动约束组合

该方法通过有控制地增加约束复杂度，扩展已见 DFA 模式。遵循$L_1 \rightarrow L_2$ 变换（第 2.3节）所阐释的设计原则，我们系统地结合了标准课程中很少同时出现的多重约束：(i) **乘积构造：** 多个合取/析取约束，需要进行状态空间乘积（例如，“以‘aba’或‘bab’开头，并以其逆序结尾的字符串”）。(ii)**相互作用的约束：**无法在局部验证的独立条件（例如，“倒数第四个符号是‘a’，且子串‘bb’从不出现在‘a’之前”——语言 L_2）。(iii) **结构限制：** 位置模式和禁用子串（例如，“前三个位的异或等于最后一位”；“每个 4 位窗口的乘积都能被 4 整除”）。(iv) **叙事编码：** 需要形式化的现实场景（例如，将国际象棋开局序列编码为着法；将水果混合配方表示为糖浆组合；其他示例见附录B ）。全部 60 道问题均经过人工核验，确认未出现在谷歌搜索结果前 100 条、标准教科书（Sipser，2013；Hopcroft 等，2001）以及热门在线代码库（`GitHub`、`StackOverflow`、课程网站）中。这种方法能够生成高质量、用于测试创造性约束整合的问题，但其可扩展性受到人工投入的限制。

3.2 利用 Arden 定理生成

为了解决可扩展性问题，同时确保结构上的新颖性，我们采用逆向工程方法：生成随机的非确定有限自动机，通过 Arden 定理以代数方式推导其所接受的语言，并将所得正则表达式用作问题规格说明。这一系统化过程确保每个生成的问题都具有由随机非确定有限自动机拓扑结构决定的独特结构，而不是源于记忆的模式。具体生成过程如下：**(i) 随机 NFA 构造：** 生成具有 $5-8$ 个状态、$2-3$ 个接受状态以及随机转移密度 $0.3-0.6$的随机 NFA。非确定转移函数 $\delta: Q\times\Sigma\to 2^Q$ 允许每个输入符号对应多个后继状态。**(ii)通过 Arden 定理推导语言：** 应用 Arden 定理（Sipser，2013）——如果正则表达式 P和Q 满足 $R=Q+RP$且 P不包含 ϵ，则 $R=QP^*$——以迭代方式消除状态并推导闭式正则表达式。**(iii)最小 DFA 构造：** 通过子集构造法将 NFA 转换为 DFA，以获得基准真值解。



(a) (b)

图 2：**(a)：** $\{a,b\}$上的随机非确定有限自动机。**(b)：** 识别所得语言 L_3 的最小确定有限自动机。

详细示例：推导语言 L_3。现在，我们使用图2(a)所示的随机非确定有限自动机，演示完整的推导过程。

步骤 1：由非确定有限自动机列出状态方程。根据图2(a)中的入边递归表示每个状态对应的语言：

$$q_5 = q_5a + q_5b + q_4a \tag{1}$$
$$q_4 = q_0b + q_4b \tag{2}$$
$$q_3 = q_3a + q_3b + q_2b \tag{3}$$
$$q_2 = q_1b \tag{4}$$
$$q_1 = q_0a + q_1a + q_2a \tag{5}$$
$$q_0 = \epsilon + q_0b \tag{6}$$

这里，q_0 是初始状态（接受状态 ϵ），而 q_3、q_5是接受状态。我们的目标是推导出闭合形式的正则表达式，用 q_3 和 q_5 表示，并且只使用输入字母表 $\{a,b\}$。

步骤 2：利用 Arden 定理求解 q_0。方程（6）的形式为 $R=Q+RP$，其中 $R=q_0$，$Q=\epsilon$，以及 $P=b$。由于 b 不包含 ϵ，根据 Arden 定理可得：$q_0=\epsilon\cdot b^*=b^*$

步骤 3：求解 q_1。将方程（$q_2=q_1b$）代入方程（4）中：5)

$$\begin{aligned} q_1 &= q_0a + q_1a + q_2a = q_0a + q_1a + (q_1b)a \\ &= q_0a + q_1(a+ba) \end{aligned}$$

应用 Arden 定理，使用 $R=q_1, Q=q_0a, P=(a+ba)$：$q_1=q_0a(a+ba)^*=b^*a(a+ba)^*$。

按照类似的方法应用 *Arden* 定理，我们得到终态对应的以下正则表达式：$q_3=b^*a$（$a+ba$）$^*bb$（$a+b$）*以及 $q_5=b^*bb^*a$（$a+b$）*

最终语言：因此，该非确定有限自动机识别的语言为：$L_3=q_3\cup q_5$。于是，

$$L_3 = b^*a(a+ba)^*bb(a+b)^* + b^*bb^*a(a+b)^*$$

所得表达式具有深层嵌套结构（包含克林闭包的并集之间相互连接），且结构非标准，不太可能出现在教科书示例中。图2（b）展示了 L_3 的最小确定有限自动机，它是通过子集构造法转换该非确定有限自动机并进行状态最小化得到的。采用这一流程，我们生成了 120 道涵盖不同结构复杂度的问题。每个随机非确定有限自动机种子都会生成一个不同的正则表达式，从而全面覆盖各种表达式模式，同时通过算法化确定有限自动机构造确保结果正确且可验证[1]。

3.3 数据集统计与难度等级

我们根据结构复杂度将 180 个未见问题按难度分类：(i) 简单（41 个问题，占 22.8%）：嵌套较浅的简单表达式（例如，$L_4=b(a+b)^*ab$）– 1 个克林闭包、1 个并集、3 个连接，最小确定有限自动机有 ≤ 5 个状态）。(ii) 中等（65 个问题，占 36.1%）：具有 $2-4$ 个嵌套运算符、组合性适中的表达式（例如，$L_5=(a^*b+((a^*a+a^*b)b^*)a+(((a^*a+a^*b)b^*a)b^*)b)a^*$；最小确定有限自动机有 $6-12$ 个状态）。(iii) 困难（74 个问题，占 41.1%）：需要进行全局一致性跟踪的深层嵌套表达式（例如，$L_6=(((a^*b)(a+b((\epsilon+a)(ba)^*(a+bb)+b)b^*a)^*)b((\epsilon+a)(ba)^*(a+bb)+b))b^*$；最小确定有限自动机有 ≥ 13 个状态）。对于人工编写的问题

[1]所有数据集、文件格式和其他实现细节均见附录B。

数据集组成部分	数量	Easy	Med	Hard
未见过的构造	**180**	**41**	**65**	**74**
人工组合	60	15	22	23
Arden 生成	120	26	43	51

表 2：数据集统计与难度分类

，难度取决于相互作用的约束数量（1−2简单问题，3−4 中等问题，5+困难问题）。对于 Arden 生成的问题，难度对应于正则表达式嵌套深度以及确定有限自动机中的状态数量。表2 总结了完整的基准测试。难度分布确保了对不同复杂度级别的全面评估，并略微偏重困难问题（41%），以对推理能力进行压力测试。详细分类请参见附录C。

新颖性验证。 为确保问题确实未曾出现，我们：(1) 手动搜索了谷歌顶级搜索结果和在线代码库，查找完全匹配及结构相似的内容；(2) 验证了由 Arden 生成的正则表达式不匹配标准教科书中的模式；(3) 确认手工设计的问题以非标准方式组合了约束。虽然我们无法保证这些问题完全不存在于所有预训练语料库中，但受控生成流程和验证步骤为其新颖性提供了有力证据。

4 评估方法

在此，我们通过官方 API 访问所有模型，并将温度设置为 0，以实现确定性解码；所有实验均使用完全相同的提示模板，以确保公平比较 [2]。我们评估了五种提示配置，涵盖零样本、显式推理、多分支探索和引导式自我纠正：

直接输入-输出（零样本）。 模型仅接收问题规格说明（正则表达式或自然Kojima 等人，2022年语言），并必须输出完整的确定有限自动机，不得提供中间推理过程或演算示例（）。该基线在没有辅助框架的情况下，单独考察纯粹的构造能力。

思维链模型会先按照指示逐步推理，再生成最终的确定有限自动机（Wang 和 Zhou，2024年）。确定有限自动机的构造自然分为：（1）解读语言规格说明，（2）识别所需状态及其语义角色，以及（3）设

[2]关于确定性控制、解码设置和运行环境的其他详细信息，见附录I和J。

计确保正确接受或拒绝的转移。例如，给定 $L_7 = (a + b)^* a(a + b)^*$，思维链提示会鼓励模型明确推理：“状态 q_0 跟踪不含 ‘a’ 的字符串；状态 q_1 （接受状态）跟踪至少含有一个 ‘a’ 的字符串；转移 $\delta(q_0,\ a) = q_1$ 处理第一个 ‘a’；等等。”。**思维链（单样本）**。通过在目标问题之前提供一个完整示例（问题 +、解答 +、推理轨迹），扩展了思维链。这用于测试明确演示是否能减少推理结构中的歧义。**思维树（ToT）**。将确定有限自动机的构造分解为四个不同的推理分支，分别对应自动机理论中的标准方法（Yao 等人，2023年）。提示模型并行探索多种构造方法：（i）**直接法（直觉式）：**从语义上解释正则表达式，根据语言属性确定必要的状态，并直接设计转移。此方法测试缺少算法辅助框架时的高层符号推理能力。（ii）**基于最小化的方法：**先构造一个初始确定有限自动机（可能包含冗余状态），再通过划分细化合并等价状态。此方法测试模型是否理解状态等价，而不仅仅是局部转移的正确性。（iii）**基于导数的方法：**应用 Brzozowski 方法（Sipser，2013），其中每个状态都对应于正则表达式相对于输入前缀的一个不同导数。此方法测试符号代数运算以及等价表达式的规范化能力。（iv）**Thompson 构造：**遵循算法流程：通过结构递归将正则表达式转换为 ϵ-非确定有限自动机，然后通过子集构造法将其确定化。此方法测试模型在多阶段形式化算法中的程序执行正确性。Lex、Flex 等词法分析器工具采用这种方法自动从正则表达式构造确定有限自动机，但存在局限（Aho 等人，2006）。这些分支涵盖了本质上不同的推理方式（语义解释、等价优化、代数运算、算法执行），能够细致地诊断模型在哪些方面表现出色或存在不足。所有提示模板见附录D。

基于提示的自我纠正协议。 对于在直接提示下回答错误的问题，我们评估模型在获得结构化反馈后能否自我纠正。我们提出了一种包含三个阶段、指导程度逐步增强的提示协议：**阶段1：反例反馈**。 模型会收到揭示其确定有限自动机错误的具体反例：**假阴性：**

属于 L 但被确定有限自动机拒绝的字符串；**假阳性：**不属于 L 但被确定有限自动机接受的字符串。反例的选择旨在覆盖自动机中的所有主要结构路径，确保错误无法通过局部修补解决。 **阶段2：错误定位**。 如果错误仍然存在，模型会被告知哪个推理阶段 存在错误（语言解释、状态设计或转移），并收到高层次的描述（例如："你的确定有限自动机错误地简化了克林闭包约束"）。 **阶段3：明确披露错误**。 如果模型仍然失败，它会收到错误的确切位置和性质（例如："缺少转移： $\delta(q_2, a) \to q_4$ ，以处理 aba"）。这种渐进式信息披露旨在检验失败究竟源于浅层误解（可通过反例纠正）、推理不完整（可通过定位纠正），还是无法维持符号一致性的根本缺陷（即使有明确指导也无法纠正）。附录D包含提示模板。

验证与评分。所有确定有限自动机输出均通过两阶段流程进行验证，该流程结合了自动化行为测试和人工结构检查：(i)**自动化验证：**自定义验证器会检查：(a)语法正确性：JSON模式有效，状态／转移结构格式正确；(b) 全定义性：每个状态针对每个输入符号都恰好有一个出边；(c) 行为等价：该确定有限自动机接受的语言与真实正则表达式所表示的语言相同。我们通过穷举所有长度不超过 6 的字符串，并随机抽取 2000 个长度为 $7-15$ 的字符串来测试行为等价性。任何不匹配都会触发明确的反例记录。(ii)**人工检查：**对于所有确定有限自动机，研究团队至少安排两名独立审查者人工核验：(a) 是否正确理解语言规格说明；(b) 状态语义是否合理且与正则表达式一致；(c) 转移逻辑是否有效。所有评分均采用模型盲法，以避免偏差。只有同时通过两项测试，确定有限自动机才会被判定为正确（验证代码、评估脚本和执行脚本见附录 E 和 F）。

5 结果与分析

结果。 表 3报告了不同数据集、提示策略和模型的 DFA 构造成功率。（i）**已见与未见：**与已见数据集的结果（第 2.2 节）相比，在未见确定有限自动机构造

数据集	提示工程	GPT-5.1	Grok-4.1	Gemini-2.5
已见 DFA	零样本	84.2%	89.5%	85.0%
未见确定有限自动机	零样本	20.67%	59.12%	29.33%
	CoT	16.67%	51.90%	23.33%
	思维链（单样本）	19.33%	55.87%	28.10%
	ToT	24.10%	–	54.00%

表3：不同数据集、提示策略和模型上的 DFA 构造成功率。"–"表示推理时超时。

数据集上，所有模型的表现都显著下降。在直接提示下，*Grok-4.*1-fast-reasoning 表现最佳，其次是 Gemini-2.5-Flash 和 GPT-5.1。与已见任务相比，性能下降了 30–64%。关键在于，已见和未见设置中的任务表述完全相同。因此，观察到的性能差距反映的是组合泛化能力不足，而非任务理解能力不足。（ii）**CoT效果：** 在所有模型中，思维链都会持续降低未见 DFA 构造任务的表现。在此情形下，显式的逐步推理会增加有效状态空间的复杂度，并放大早期语义错误的影响。为评估示例能否缓解这一失败模式，我们测试了思维链（单样本）提示。提供一个完整示例，相较于标准思维链，能够提升性能。（iii）**思维树效果：**在复杂未见实例上，思维树取得了最佳表现。与直接提示相比，Gemini-2.5 和 GPT-5.1 的表现分别提升了 24.67% 和 3.43%；与思维链提示相比，分别提升了 30.67% 和 7.43%。相比之下，Grok-4.1-fast-reasoning 经常无法在固定的推理时预算 [3]内返回结果。

分析。 在不同模型和提示策略中，主要失败模式源于难以维持全局一致的符号结构，而非缺乏形式知识。为作说明，我们分析以下四种思维树框架方法（涵盖直接方法和思维链）的代表性输出。(i)**汤普森构造：** 对于嵌套有限的中等复杂度正则表达式，大语言模型通常能成功运用此方法，因为该方法完全依照算法执行。然而，对于涉及深层嵌套的复杂表达式，中间的 ε-非确定有限自动机会迅速膨胀。这种状态爆炸 是大语言模型失败的原因之一。重要的是，这些失败并非反映了对方法的误解，而是体现了模型难以可靠管理大型、

[3]第7节提供了对超时行为、推理成本和可部署性权衡的详细分析。

高度密集连接的中间结构。(ii) 直接构造：在此情境下，大语言模型尝试通过非正式推理语言约束来构造确定有限自动机。这种设置呈现出以下反复出现的失败模式。(a) 语言解释不完整或过度简化。要构造正确的确定有限自动机，逻辑上最重要的第一步是正确理解语言。然而，有大量证据表明，大语言模型在这一步存在不足（例如，对于 $L_8 = a^*(a+b)(a+b)^*a$，模型将其错误地理解为“以 a 结尾”，忽略了 $(a+b)$ 的存在）。对于“克林闭包”，情况也一样。(b) 模型所陈述的理解与最终构造不一致。（例如，起初正确理解为“以 ab 结尾”，但最终给出的答案却对应“以 ba 结尾”。）这表明模型难以在构造过程中维持符号不变量。(c) 拼接时未能保留约束。（例如，无法处理 L_8，但能够处理单独的组成部分，即 a^*、$(a+b)$、$(a+b)^*$ 和 a。）(d) 过度接受目标语言之外的字符串。模型往往只用有限个被接受的字符串验证其构造，却忽略应拒绝的情形。此外，大语言模型还常常引入不可达或冗余状态，进一步导致答案无效[4]。(iii) 基于最小化的方法构造。此处，大多数错误在应用最小化之前就已产生，并在算法化最小化过程中延续。(iv) 基于导数的方法构造。大语言模型经常无法规范化语义等价的正则表达式导数，导致产生伪确定有限自动机状态。（例如，对于 $L_9 = b(a+b)^*ab$[5]，导数 $(a+b)^*ab + \varepsilon$ 是正确的中间步骤，体现了可空性。然而，大语言模型经常将 ε 替换为 $(a+b)^*$，从而得到错误的中间正则表达式 $(a+b)^*ab + (a+b)^*$。该正则表达式规范化为 $(a+b)^*$，因此丢弃了原有的语言约束。）重要的是，这一失败表明模型缺乏进行基本代数化简和语言包含推理的能力。

基于提示的协议分析。 基于提示的协议旨在评估大语言模型在获得结构化指导后，能否从错误的确定有限自动机构造中恢复。表4总结了模型在不同难度级别和提示阶段的表现。(i) **简单：** 大多数错误都能

[4]附录 G 包含具有代表性的错误汇编。[5]模型生成的输出请参见附录 H 。

阶段	GPT-5.1			Grok-4.1			Gemini-2.5		
	Easy	中等	困难	Easy	中等	困难	Easy	中等	困难
第一个提示	60.13%	10.13%	30.76%	85.25%	39.20%	52.93%	57.45%	49.13%	31.79%
第二条提示	15.50%	16.10%	15.38%	14.75%	27.67%	7.83%	42.55%	12.47%	6.88%
最终提示	24.37%	36.17%	–	–	13.33%	–	–	12.37%	16.31%
未解决	–	37.60%	53.86%	–	19.80%	39.24%	–	26.03%	45.02%

表 4：基于提示的DFA构造性能。

在第一个提示 后得到纠正，且这一情况适用于所有模型，这表明该难度级别下的失败主要源于表层误解，而非根本性的推理错误。(ii) **中等：** 在此难度下，模型的恢复分布在多个提示阶段，而不是集中于某一个指导级别。此外，即使在最终提示之后，仍有相当一部分结果是**困难：** 即使提供了所有提示，仍有很大比例的问题未能解决，GPT-5。1 和 Gemini-2。5-Flash 尤其如此。未解决案例持续存在，这表明对于复杂的正则表达式，指导不足以克服符号状态转移跟踪以及全局自动机结构方面的内在局限。总体而言，基于提示的分析表明，尽管所有模型都能借助指导纠正浅层错误，但随着确定有限自动机复杂度增加，它们的恢复能力会迅速下降。提示主要有助于重新解决局部误解，却无法可靠地纠正全局不一致或结构有缺陷的自动机。

6 结论

本文对大语言模型在确定有限自动机构造任务上的表现进行了系统评估，重点考察符号正确性，而非表层模式匹配。我们发现，尽管当前大语言模型能够可靠地复现熟悉的构造，但在面对组合复杂的输入时，其性能会显著下降。我们的分析表明，这些失败是系统性的，而非随机的。这些局限性在不同提示策略下依然存在，说明问题源于符号推理的结构性弱点，而不是特定提示的不足。基于提示的框架表明，对于较简单的错误，实现有限的自我纠正是可能的；然而，更深层的语义不一致很少能够得到解决。通过引入具有受控新颖性的严谨基准测试，并识别反复出现的失败模式，本文为未来通过改进评估方法、开展针对性训练或采用符号—神经混合方法来提升符号鲁棒性的研究奠定了基础。

7 局限性

在解读结果时，应考虑本研究存在的若干局限性。

模型访问与评估限制。 所有实验均通过公共 API 或标准模型接口进行，并采用固定的推理时间预算。因此，部分模型——尤其是使用思维树提示的 Grok-4.1-fast-reasoning——经常无法在规定预算内返回结果。此类情况被视为尝试失败，反映的是实际接口和可部署性限制，而非底层推理能力的确切局限。

令牌与状态空间的可扩展性。 从复杂正则表达式构造确定有限自动机，往往会产生庞大的中间表示，尤其是在采用思维链和思维树提示时。令牌预算耗尽和隐式状态空间爆炸，限制了这些方法在处理高度嵌套或组合式表达式时的可靠性，即使其底层构造在理论上定义明确也是如此。

有限验证范围。 通过穷举枚举（仅限于长度不超过某个上限的字符串），并辅以针对较长字符串的随机测试，来检验生成的确定有限自动机与真实正则表达式在行为上是否等价。虽然这种方法提供了有力的经验性保证，但并不能构成对所有可能字符串均等价的形式化证明。因此，测试长度范围之外的罕见反例可能仍未被发现，从而可能导致对构造正确性的估计偏高。

数据集格式限制。 所有确定有限自动机构造数据集均以 PDF 格式发布，包含问题陈述、参考转移表和确定有限自动机图。这种设计优先考虑了人类可解释性并保留了教学结构，但下游复用需要手动或半自动解析，也限制了直接的机器可读性。

任务范围与形式体系。实验仅关注正则表达式、确定有限自动机/非确定有限自动机的构造，以及基于 Arden 定理的逆向综合。因此，所报告的结果未必适用于更丰富的自动机模型、非正则语言或其他形式系统。因此，我们的研究结果不应推广至上下文无关文法、下推自动机或图灵完备模型等更丰富的形式体系。

不同界面之间的模型差异。 通过 API 和网页界面提交的相同提示词可能会因未公开的系统级差异而产生不同输出。所有报告的结果均对应基于 API 的评估，不应被解读为网页界面行为的精确复现。

缺少微调或自适应提示。除论文中所述的固定提示协议外，未进行任何特定于模型的微调、自适应解码或提示优化。虽然这确保了不同模型之间受控的可比性，但在专门化或经过调优的评估设置下，这可能低估可达到的性能。

这些限制主要影响我们结论的外部效度，但不会改变不同模型和提示策略之间观察到的相对性能趋势。

8 伦理考量

本研究评估大语言模型（LLM）执行形式语言理论推理任务的能力，具体包括确定有限自动机（DFA）构造及相关逆问题。本研究不涉及人类受试者、个人数据或敏感信息。

数据集来源。 本研究所用数据集完全由符号化的数学对象构成，包括正则表达式、确定有限自动机、非确定有限自动机、转移表，以及选择题或判断题。

KnowledgeChecking 数据集和已见 DFA 构造数据集取自公开可用的教育材料及可在线访问的标准自动机理论习题集。这些材料通常用于教学和评估，且不受限制性许可协议约束。

未见 DFA 构造数据集（包括数学工程和数学艺术子集）由作者手动构建，未取自在线代码仓库。所有数据集均在配套存储库中明确注明数据许可。

所有新创建的数据集均依据知识共享署名 4.0 国际许可协议（CC BY 4.0）发布。公开获取的教育数据集仅用于研究和评估，原始版权仍归各自所有者所有。完整的许可文本和使用条款见配套存储库。

模型使用与评估。 所有受评估模型均仅通过其官方公共 API，在标准使用条件下进行访问。我们未尝试提取专有模型内部信息、训练数据或隐藏的推理轨迹。提示词仅用于引出模型的任务表现，并未要求提供个人信息、敏感信息或受版权保护的信息。

可复现性与透明度。 为促进透明度和负责任的研究实践，我们公开了完整的提示词、实验脚本、验证代码以及具有代表性的输出。我们采用自动化验证和有针对性的人工检查，以降低报告虚假或误导性结果的风险。

潜在风险与滥用。 所研究的任务纯属形式化和数学性质，公开发布的材料预计不会助长有害应用，也不会用于面向部署的系统。一个可能的风险是对结果产生误解或过度泛化，例如将观察到的失败或成功视为模型在更广泛的语言或现实世界领域中具备或缺乏一般推理能力的证据。我们明确将主张限定于正则语言形式体系和符号自动机构造，并提醒不要将结论外推至这些范围之外。

人工智能助手的使用。 在辅助代码的开发和调试过程中，有限度地使用了人工智能助手。它们未用于生成数据集、标准答案、实验结果、分析或结论，也未对本研究的科学主张作出贡献。

总体而言，我们未发现与本研究相关的重大伦理风险，并遵守会议关于数据来源、透明度和负责任报告的准则。

参考文献

Alfred V. Aho、Monica S. Lam、Ravi Sethi 和 Jef- frey D. Ullman。2006年。编译原理、技术与工具（第2版）。Addison-Wesley Longman Publishing Co., Inc.，美国。Zhangir Azerbayev、Bartosz Piotrowski、Hailey Schoelkopf、Edward W. Ayers、Dragomir Radev 和 Jeremy Avigad。2023年。Proofnet：本科数学的自动形式化与形式化证明。预印本，arXiv:2302.12433。

Matei A.Golesteanu、Garrett B.Vowinkel 和 Ryan E. Dougherty。2024年。ChatGPT 能否通过计算理论课程？载于*ACM* 虚拟全球计算机教育大会论文集，第 33–38 页。计算机协会。

John E. Hopcroft、Rajeev Motwani 和 Jeffrey D. Ullman。2001年。《自动机理论、语言与计算导论》，第 2 版。Addison-Wesley。

Michael Katz、Harsha Kokel 和 Sarath Sreedharan。2025年。看似简单的规划问题在计算上颇具挑战：倒计时游戏。预印本，arXiv:2508.02900。

Takeshi Kojima、ShixiangShane Gu、Machel Reid、Yu- taka Matsuo 和 Yusuke Iwasawa。2022年。大语言模型是零样本推理器。预印本，arXiv:2205.11916。

Aitor Lewkowycz、Anders Andreassen、David Dohan、Ethan Dyer、Henryk Michalewski、Vinay Ramasesh、Ambrose Slone、Cem Anil、Imanol Schlag、Theo Gutman-Solo、Yuhuai Wu、Behnam Neyshabur、Guy Gur-Ari 和 Vedant Misra。2022年。使用语言模型解决定量推理问题。载于第*36*届神经信息处理系统大会（*NeurIPS*）论文集。

Michael Sipser。2013年。计算理论导论，第3版。CourseTechnology。

Xuezhi Wang 和 Denny Zhou。2024年。无需提示的思维链推理。预印本，arXiv:2402.10200。

SeanWelleck、JiachengLiu、RonanLe Bras、Hannaneh Hajishirzi、Yejin Choi 和 Kyunghyun Cho。2021年。Naturalproofs：自然语言中的数学定理证明。预印本，arXiv:2104.01112。

Yuhuai Wu、Albert Q. Jiang、Wenda Li、Markus N. Rabe、Charles Staats、MatejaJamnik 和 Christian Szegedy。2022年。使用大型语言模型进行自动形式化。载于第*36*届神经信息处理系统大会（*NeurIPS*）论文集。

Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Thomas L. Griffiths、Yuan Cao 和 Karthik Narasimhan。2023年。思维树：深思熟虑地

使用大语言模型解决问题。在第 *37* 届神经信息处理系统大会（*NeurIPS*）论文集。

Xiang Yue、Yuansheng Ni、Kai Zhang、Tianyu Zheng、Ruoqi Liu、Ge Zhang、Samuel Stevens、Dongfu Jiang、Weiming Ren、Yuxuan Sun、Cong Wei、Botao Yu、Ruibin Yuan、Renliang Sun、Ming Yin、Boyuan Zheng、Zhenzhu Yang、Yibo Liu、Wenhao Huang及另外3位作者。2024年。MMMU：面向通用人工智能专家的大规模多学科多模态理解与推理基准测试。载于*IEEE/CVF* 计算机视觉与模式识别会议论文集*(CVPR)*。

Pardis Sadat Zahraei 和 Ehsaneddin Asgari。2024年。Turingq：计算理论中的人工智能理解能力基准测试。预印本，arXiv:2410.06547。

A 附录：特定模型的确定有限自动机输出

本附录报告了每个受评模型针对已见任务（L_1）和未见任务（L_2）生成的确切确定有限自动机构造。所有输出均按模型返回的原样展示，未经过后处理或修正。

A.1 已见问题（L_1）

A.1.1 GPT-5.1 对 L_1的输出


```
{
  "状态": [
    "q0",
    "q1",
    "q2",
    "q3",
    "q4",
    "q5",
    "q6",
    "q7"
  ],
  "字母表": [
    "a",
    "b"
  ],
  "起始_状态": "q0",
  "accept_状态": [
    "q4",
    "q5",
    "q6",
    "q7"
  ],
  "转移": {
    "q0": {
      "a": "q1",
      "b": "q0"
    },
    "q1": {
      "a": "q3",
      "b": "q2"
    },
    "q2": {
      "a": "q5",
      "b": "q4"
    },
    "q3": {
      "a": "q7",
      "b": "q6"
    },
    "q4": {
      "a": "q1",
      "b": "q0"
    },
    "q5": {
      "a": "q3",
      "b": "q2"
    },
    "q6": {
      "a": "q5",
      "b": "q4"
    },
    "q7": {
      "a": "q7",
      "b": "q6"
    }
  }
}
```


A.1.2 Grok-4-1-fast-reasoning 输出 L_1


```
{
  "状态": [
    "q0",
    "q1",
    "q2",
    "q3",
    "q4",
    "q5",
    "q6",
    "q7"
  ],
  "字母表": [
    "a",
    "b"
  ],
  "起始_状态": "q0",
  "接受_状态": [
    "q4",
    "q5",
    "q6",
    "q7"
  ],
  "转移": {
    "q0": {
      "a": "q1",
      "b": "q0"
    },
    "q1": {
      "a": "q3",
      "b": "q2"
    },
    "q2": {
      "a": "q5",
      "b": "q4"
    },
    "q3": {
      "a": "q7",
      "b": "q6"
    },
    "q4": {
      "a": "q1",
      "b": "q0"
    },
    "q5": {
      "a": "q3",
      "b": "q2"
    },
    "q6": {
      "a": "q5",
      "b": "q4"
    },
    "q7": {
      "a": "q7",
      "b": "q6"
    }
  }
}
```


A.1.3 Gemini-2.5-flash 对 L_1的输出


```
{
  "状态": [
    "q0",
    "q1",
    "q2",
    "q3",
    "q4",
```

```
      "q5",
      "q6",
      "q7"
    ],
    "字母表": [
      "a",
      "b"
    ],
    "起始_状态":        "q0",
    "接受_状态": [
      "q4",
      "q5",
      "q6",
      "q7"
    ],
    "transitions": {
      "q0": {
        "a": "q1",
        "b": "q0"
      },
      "q1": {
        "a": "q2",
        "b": "q3"
      },
      "q2": {
        "a": "q4",
        "b": "q5"
      },
      "q3": {
        "a": "q6",
        "b": "q7"
      },
      "q4": {
        "a": "q4",
        "b": "q5"
      },
      "q5": {
        "a": "q6",
        "b": "q7"
      },
      "q6": {
        "a": "q2",
        "b": "q3"
      },
      "q7": {
        "a": "q1",
        "b": "q0"
      }
    }
}
```


A.2 未见问题（L_2）

A.2.1 GPT-5.1 对 L_2的输出


```
{
    "states": [
      "q0",
      "q1",
      "q2",
      "q3",
      "q4",
      "q5",
      "q6",
      "q7",
      "q8",
      "q9",
      "q10",
      "q11",
      "q12",
      "q13",
      "q14",
      "q15",
      "q16",
      "q17",
      "q18",
      "q19",
      "q20",
      "q21",
      "q22",
      "q23",
      "q24",
      "q25",
      "q26",
      "q27",
      "q28",
      "q29",
      "q30",
      "q31",
      "qDead"
    ],
    "字母表": [
      "a",
      "b"
    ],
    "起始_状态":        "q0",
    "接受_状态": [
      "q24",
      "q25",
      "q26",
      "q27",
      "q28",
      "q29",
      "q30",
      "q31"
    ],
    "转移": {
      "q0": {
        "a": "q1",
        "b": "q2"
      },
      "q1": {
        "a": "q3",
        "b": "q4"
      },
      "q2": {
        "a": "q1",
        "b": "qDead"
      },
      "q3": {
        "a": "q5",
        "b": "q6"
      },
      "q4": {
        "a": "q3",
        "b": "q7"
      },
      "q5": {
        "a": "q8",
        "b": "q9"
      },
      "q6": {
        "a": "q5",
        "b": "q10"
      },
      "q7": {
        "a": "q6",
        "b": "q11"
      },
      "q8": {
        "a": "q12",
        "b": "q13"
      },
      "q9": {
        "a": "q8",
        "b": "q14"
      },
      "q10": {
        "a": "q9",
        "b": "q15"
      },
      "q11": {
        "a": "q10",
        "b": "q16"
      },
      "q12": {
        "a": "q17",
        "b": "q18"
      },
      "q13": {
        "a": "q12",
        "b": "q19"
      },
      "q14": {
        "a": "q13",
        "b": "q20"
      },
      "q15": {
        "a": "q14",
        "b": "q21"
      },
      "q16": {
```

```
      "a": "q15",
      "b": "q22"
    },
    "q17": {
      "a": "q23",
      "b": "q24"
    },
    "q18": {
      "a": "q17",
      "b": "q25"
    },
    "q19": {
      "a": "q18",
      "b": "q26"
    },
    "q20": {
      "a": "q19",
      "b": "q27"
    },
    "q21": {
      "a": "q20",
      "b": "q28"
    },
    "q22": {
      "a": "q21",
      "b": "q29"
    },
    "q23": {
      "a": "q23",
      "b": "q30"
    },
    "q24": {
      "a": "q23",
      "b": "q31"
    },
    "q25": {
      "a": "q24",
      "b": "q31"
    },
    "q26": {
      "a": "q25",
      "b": "q31"
    },
    "q27": {
      "a": "q26",
      "b": "q31"
    },
    "q28": {
      "a": "q27",
      "b": "q31"
    },
    "q29": {
      "a": "q28",
      "b": "q31"
    },
    "q30": {
      "a": "q30",
      "b": "q31"
    },
    "q31": {
      "a": "q30",
      "b": "q31"
    },
    "qDead": {
      "a": "qDead",
      "b": "qDead"
    }
  }
}
```


A.2.2 Grok-4-1-fast-reasoning 对 L_2的输出


```
{

  "状态": [
    "q0",
    "q1",
    "q2",
    "q3",
    "q4",
    "q5",
    "q6",
    "q7",
    "q8",
    "q9",
    "q10",
    "q11",
    "q12",
    "q13",
    "q14",
    "q15",
    "q16",
    "q17",
    "q18",
    "q19",
    "q20",
    "q21",
    "q22",
    "q23",
    "q24",
    "q25",
    "q26",
    "q27",
    "q28"
  ],
  "字母表": [
    "a",
    "b"
  ],
  "起始_状态": "q0",
  "接受_状态": [
    "q13",
    "q14",
    "q15",
    "q16",
    "q17",
    "q18",
    "q19",
    "q20"
  ],
  "转移": {
    "q0": {
      "a": "q1",
      "b": "q2"
    },
    "q1": {
      "a": "q3",
      "b": "q4"
    },
    "q2": {
      "a": "q5",
      "b": "q12"
    },
    "q3": {
      "a": "q6",
      "b": "q7"
    },
    "q4": {
      "a": "q8",
      "b": "q9"
    },
    "q5": {
      "a": "q10",
      "b": "q11"
    },
    "q6": {
      "a": "q13",
      "b": "q14"
    },
    "q7": {
      "a": "q15",
      "b": "q16"
    },
    "q8": {
      "a": "q17",
      "b": "q18"
    },
    "q9": {
      "a": "q19",
      "b": "q20"
    },
    "q10": {
      "a": "q21",
      "b": "q22"
    },
    "q11": {
      "a": "q23",
      "b": "q24"
    },
    "q12": {
      "a": "q12",
      "b": "q12"
    },
```

```
    "q13": {
      "a": "q13",
      "b": "q14"
    },
    "q14": {
      "a": "q15",
      "b": "q16"
    },
    "q15": {
      "a": "q17",
      "b": "q18"
    },
    "q16": {
      "a": "q19",
      "b": "q20"
    },
    "q17": {
      "a": "q21",
      "b": "q22"
    },
    "q18": {
      "a": "q23",
      "b": "q24"
    },
    "q19": {
      "a": "q25",
      "b": "q26"
    },
    "q20": {
      "a": "q27",
      "b": "q28"
    },
    "q21": {
      "a": "q13",
      "b": "q14"
    },
    "q22": {
      "a": "q15",
      "b": "q16"
    },
    "q23": {
      "a": "q17",
      "b": "q18"
    },
    "q24": {
      "a": "q19",
      "b": "q20"
    },
    "q25": {
      "a": "q21",
      "b": "q22"
    },
    "q26": {
      "a": "q23",
      "b": "q24"
    },
    "q27": {
      "a": "q25",
      "b": "q26"
    },
    "q28": {
      "a": "q27",
      "b": "q28"
    }
  }
}
```


A.2.3 Gemini-2.5-Flash 的输出 L_2


```
{
  "状态": [
    "q0",
    "q1",
    "q2",
    "q3",
    "q4",
    "q5",
    "q6",
    "q7",
    "q8",
    "q9",
    "q10",
    "q11",
    "q12",
    "q13",
    "q14",
    "q15"
  ],
  "字母表": [
    "a",
    "b"
  ],
  "起始_状态": "q0",
  "接受_状态": [
    "q7",
    "q8",
    "q9",
    "q10"
  ],
  "转移": {
    "q0": {
      "a": "q2",
      "b": "q1"
    },
    "q1": {
      "a": "q5",
      "b": "q15"
    },
    "q2": {
      "a": "q3",
      "b": "q4"
    },
    "q3": {
      "a": "q7",
      "b": "q8"
    },
    "q4": {
      "a": "q9",
      "b": "q10"
    },
    "q5": {
      "a": "q11",
      "b": "q12"
    },
    "q6": {
      "a": "q13",
      "b": "q14"
    },
    "q7": {
      "a": "q7",
      "b": "q8"
    },
    "q8": {
      "a": "q9",
      "b": "q10"
    },
    "q9": {
      "a": "q11",
      "b": "q12"
    },
    "q10": {
      "a": "q13",
      "b": "q14"
    },
    "q11": {
      "a": "q7",
      "b": "q8"
    },
    "q12": {
      "a": "q9",
      "b": "q10"
    },
    "q13": {
      "a": "q11",
      "b": "q12"
    },
    "q14": {
      "a": "q13",
      "b": "q14"
    },
    "q15": {
      "a": "q15",
      "b": "q15"
    }
  }
}
```

B 附录：数据集

本附录介绍了我们在实验中使用的数据集，包括其组成、规模、格式和公开情况。数据集构建的详细流程、设计动机和验证协议详见正文。

B.1 概述

我们使用四个数据集评估大语言模型的性能，涵盖事实知识、已见 DFA 构造以及两类未见 DFA 构造任务。所有数据集均以形式语言理论和确定有限自动机为基础。标准答案通过人工构造和自动验证流程进行核验，具体说明见正文。

B.2 数据集组成与规模

•知识检查数据集（50 个问题）：

该数据集由选择题和判断题组成，用于评估对计算理论核心概念的事实性知识，包括正则语言、确定有限自动机、非确定有限自动机以及 DFA 最小化。每道题均只有一个明确无歧义的正确答案。

• **已见 DFA 构造数据集**（90 道题）：

该数据集包含从正则表达式到确定有限自动机以及从语言到确定有限自动机的构造问题，其结构模式与常见教材练习和公开可获取的在线示例高度相似。对于每道题，我们还提供了明确证据，证明公开互联网中存在结构完全相同或等价的问题，以此作为此前接触过这些问题的证明。

•未见 DFA 构造数据集（180 道题）：

该数据集旨在评估超越记忆模式的泛化能力，并经过专门设计，以避免与常见教学示例重叠。它被划分为两个生成原则不同的子集–

数学艺术子集(60 道题题）：

该子集包含通过组合多个相互作用的约束生成的问题，包括对自动机的结构限制、符号之间的条件依赖关系，以及基于叙述的任务形式化。

结果

这些语言被有意设计得高度不规则且非典型，旨在考验符号一致性、约束整合和长时程推理。为了设计这一未见子集中的语言，我们采用以下原则性构造策略：(a) **使用乘积构造**。通过乘积式构造组合多个约束（例如，反复使用合取和析取来提高约束复杂度）；例如，L_a = {构造一个确定有限自动机，使其接受 {a,b} 上所有满足以下条件的字符串：以“aba”或“bab”开头，并分别以“bab”或“aba”结尾。}L_b = {构造一个确定有限自动机，使其接受 {a,b} 上所有满足以下条件的字符串：“ab”后面必须跟着“ba”，“aa”后面必须跟着“bb”，并且“a”和“b”的总数量为偶数。} (b) **多个相互作用的约束**。同时施加相互独立的约束（例如，将前缀条件与计数约束或排除约束结合），使得单靠局部推理无法得出正确的构造；例如，L_c = {构造一个在 {a,b} 上的确定有限自动机，使其接受所有满足以下条件的字符串：从末尾倒数第四个符号必须是“a”，且在任何“a”之前都不会出现子串“bb”。}

L_d = {构造一个在 {a,b,c} 上的确定有限自动机，使得

存在一个长度不超过 5 的 A 字符块，其后紧接着一个长度为该 A 字符块两倍的 B 字符块；字符 C 可以在这两个连续字符块之前或之后的任意位置出现，且满足 $|C| \equiv 0$ (模 3)，而这两个 AB 字符块之间不能出现 C。}



图 3：鲁伊·洛佩兹开局序列之前的初始棋盘配置。

(c) **结构限制**。 结构约束，例如相邻规则、禁用子串或位置模式，要求精确跟踪符号之间的关系；例如，

L_e = { 构造一个字母表为 {0,1} 的确定有限自动机，使其接受满足以下条件的字符串 w：前三个数字进行异或运算所得的数字等于字符串 w 末尾的数字。 }L_f = { 构造一个字母表为 {0,1} 的确定有限自动机，使字符串长度大于或等于 4，并且在每个长度为 4 的子串中，前两位比特被解释为一个十进制数，后两位比特被解释为另一个十进制数，且二者的乘积可被 4 整除。 }

输入字母表定义为 Sigma = {pxyi}，其中 p 表示玩家（白方或黑方），x 表示棋子，y 表示棋盘格字母，i 表示棋盘格编号。例如，黑方骑士走到 c6 记作 bkc6。假设遵循标准国际象棋规则、双方轮流走棋且初始棋盘配置固定，构造一个最小化后的确定有限自动机，使其接受路径对应于恰好五步内（不包括最初两步）完成的西班牙开局。}

L_h = {你是一家果汁制造公司的老板，有三种口味：甜味、酸味和清淡味。有五种糖浆可供选择：0-橙子、1-柠檬、2-芒果、3-香蕉和 4-葡萄。口味组合定义如下：甜味 =芒果 +葡萄；酸味 =橙子 +柠檬 +

(d) **叙事型任务**。 叙事驱动的问题以现实世界或游戏场景为背景进行表述，必须将其转化为精确的符号化自动机约束；例如，L_g ={ 在国际象棋中，考虑西班牙开局。状态对应于棋盘上棋子的位置。

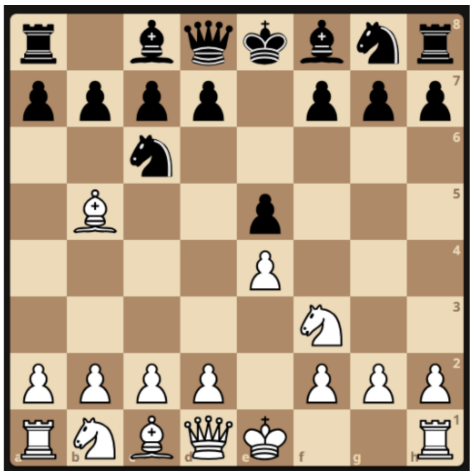


图 4: L_g中所考察的西班牙开局序列示意图。

葡萄；清淡味 = 香蕉 +芒果。构造一个确定有限自动机，使其仅接受按照这些规则构成特定口味的字符串。} 在这些策略中，我们纳入了具体实例，例如迷宫求解类比、现实场景编码以及复合逻辑要求，以说明它们对 DFA 构造难度的影响。数据集中的每种策略都由多个示例语言体现。

– **数学工程子集**（120 道题）：

该子集以 Arden 定理为核心生成工具构建。通过系统设计的确定有限自动机和非确定有限自动机转移结构推导正则表达式，从而生成符号层面精确但不常见的确定有限自动机构造问题，着重考查代数运算和形式推理，而非表面熟悉度。

B.3 数据格式

所有确定有限自动机构造数据集（已见和未见）均以**PDF 格式**提供。每个问题实例包括：

- 问题陈述（正则表达式或形式语言规范），
- 相应的确定有限自动机状态转移表，
- 供参考和验证的确定有限自动机示意图。

KnowledgeChecking 数据集采用相同的 PDF 格式提供，包含问题和明确标注的正确答案。

B.4 真实值与验证

所有真实值确定有限自动机及其对应答案均由作者手动构建，并通过形式推理和自动化行为等价性测试进行验证，具体方法见正文。数据集的构建与验证历时三个月以上，基于高阶计算理论课程作业开展。整个过程可概括如下：

• **课程背景：** 数据集开发在两门高阶计算理论课程中进行：

– **CSE 406（计算理论）：** 193 名本科生。– **CSE 525 （高阶计算理论）：** 107 名学生，其中包括本科生和研究生。

总计而言，教学材料汇集了**300 名学生**在多轮课程开设中的贡献。

•**学生贡献的性质：** 学生贡献严格限于课程作业和评估活动的常规范围内。未专门为本研究收集、分析或标注任何学生生成的数据，学生也未被视为研究参与者。

• **教学审查与验证：** 教学材料经过系统审查和验证，参与人员包括：

– **五名助教** 负责中期检查、一致性验证和教学审查。– **两名教职人员** 负责最终检查和验证。– 专门的**研究助理团队**负责整理、标准化和交叉验证。

随后，作者对数据集中包含的所有最终问题实例和标准答案进行了整理、标准化和独立验证，以确保其正确性、一致性以及对研究用途的适用性。

对于已见 DFA 构造数据集，代码仓库还包含支持性文档，

其中展示了公开可访问的在线来源中存在结构等价问题。这为这些问题此前已存在提供了明确证据，并证实这些实例并非为本研究新引入。

B.5 可用性与许可

所有数据集、提示词、实验脚本、验证代码、关于已见问题可联网获取的证据以及代表性样本输出均公开发布于：

https://anonymous.4open.science/r/dfa-llm-evaluation-B82D/

代码仓库包括：

- 四个数据集的 数据集 PDF 文件，
- 描述数据集属性的机器可读的元数据文件，
- 提示模板和实验脚本，
- 自动化验证与评估代码，
- 已见数据集问题公开可用的证明，
- 模型样本输出和验证报告。

所有数据集均依据明确的数据许可发布，该许可包含在代码仓库中，允许用于研究，并支持按照会议指南进行复现。

C 附录：难度标注标准

C.1 数学工程数据集

对于基于正则表达式的数据集，难度标签采用直接根据每个正则表达式的语法得出的量化结构复杂度分数进行分配。该分数综合了以下五项可测量属性：

- 最大嵌套深度，用于描述层级结构和长程依赖；
- 并集运算符的数量（|），体现分支和非确定性；
- 克林星号的数量（*），表示无界重复；
- 隐式连接，表示顺序组合的复杂性；
- 表达式长度，捕获整体描述大小（经对数缩放）。

这些特征经过线性组合，形成一个标量分数。随后，根据该分数的数据集范围的三分位数，将正则表达式划分为简单、中等和困难 类别。这一过程能够生成自适应且可复现的难度标签，无需手动调整阈值。我们实验中使用的完整实现如下所示。

```
导入 math 导入 json # -------- 正则表达式分析 -------- def analyze_regex(regex): length = len(regex) union = 0 star = 0concat = 0嵌套 = 0max_嵌套 = 0prev = None for c in regex: if c == '(': 嵌套 += 1max_嵌套 = max(max_嵌套, 嵌套) elif c == ')': 嵌套 -= 1elif c == '|': union += 1elif c == '*': star += 1elif prev and prev not in '(|and c not in '|)*': concat += 1prev = c return { "length": length, "union": union, "star": star, "concat": concat, "nesting": max_嵌套 } # -------- 困难度评分 -------- def difficulty_score(m): return ( 1 5 [ ] + . * m "嵌套" . * m "union" 1 0 [ ] + 0 5 [ . * m "star" ] + 0 2 [ . * m "concat"] + math.log1p(m["length"]) ) # -------- 数据集分类 --------
```

```
def classify_数据集(entries): analyzed = [] for e in entries: metrics = analyze_regex(e["regex"]) score = difficulty_score(metrics) analyzed.append((e, metrics, score)) scores = sorted(score for _, _, score in analyzed) q1 = scores[int(0.33 * len(scores))] q2 =scores[int(0.66 * len(scores))]output = []for e, metrics, score in analyzed: if score <= q1: difficulty = "简单" elif score <= q2: difficulty = "中等" else: difficulty = "困难" entry = e.copy() entry["metrics"] = metrics entry["score"] = round(score, 2) entry["difficulty"] = difficulty
```

```
            output.append(entry)

    return output

# -------- 示例用法 --------

若名称 ==" main ":__    __    __    __
    数据集 = [
        {
            "id":  "r001" ,
            "正则表达式":
"((b*a) (b*a(a b)*(a b)))a*(a b)a*aa*",
            "字母表": ["a",  "b"]
        }
    ]

    已分类 = 分类_数据集(数据集)
    打印(json.dumps(已分类, indent=2))
```

C.2 MathematicalArt 数据集

数学艺术 数据集由自然语言描述的确定有限自动机构造任务组成，这些任务具有多个相互作用的约束。与基于正则表达式的问题不同，这些任务并非仅由形式语法定义；因此，难度是根据语义结构而非表面形式确定的。

具体而言，难度标签分配如下：

- **简单和中等**任务的区分依据是显式约束的数量及其相互作用的程度（例如位置约束、计数约束和禁用子串）。

- **困难** 任务涉及以叙述为基础的表述，例如网格导航、游戏场景或现实世界类比。这些问题需要多步解释、隐式状态跟踪，以及将非正式描述转换为形式自动机。

这种标注方案反映了叙事背景和约束整合带来的额外推理负担，而这始终会导致受评估模型的错误率更高。

D 附录：提示模板

本附录提供了所有实验中使用的确切提示词。为确保可复现性，提示词均按原文逐字报告。

D.1 用于正则表达式到确定有限自动机构造的提示词

D.1.1 直接输入–输出提示词

系统：你是形式语言和自动机方面的专家。

用户：给定以下正则表达式和字母表：

正则表达式：{{REGEX}} 字母表：{{ALPHABET}} 任务：构造一个正确的确定有限自动机（DFA），使其恰好识别由该正则表达式表示的语言。约束：- DFA 必须是全定义的（每个状态对每个符号都恰有一个转移）。- 使用简短的状态名称：q0、q1、q2、……- 不要包含解释、推理、推导或中间步骤。- JSON 对象之外不要包含任何文本。仅输出一个符合以下模式的 JSON 对象，且必须完全匹配：{ " states": ["q0", "q1", "..."], "alphabet": ["a", "b", "..."],"start_状态": "q0", "accept_状态": ["q1", "..."],"transitions": { "q0": { " a": "q1", "b": "q0" }, "q1": { "a": "q1", "b": "q2" } } }

D.1.2 思维链（CoT）提示词

系统：
你是形式语言和有限自动机方面的专家。当被要求生成 DFA 转移表时，请在内部逐步思考（思维链），但不要输出任何推理过程。仅输出一个符合下述模式的有效 JSON 对象。用户：给定以下正则表达式和字母表：正则表达式：{{REGEX}} 字母表：{{ALPHABET}} T

ask：1. 在内部（思维链）推导出一个正确的确定有限自动机（DFA），用于识别该正则表达式所表示的语言。2. 不要输出任何中间推理过程或解释。3. 仅输出一个包含 DFA 转移表且不含其他内容的 JSON 对象。JSON 必须符合以下确切模式（包括键和类型）。

所需的 JSON 模式（确切的键和类型）：
{
"状态":字母表": ["q0", "q1", "b", ...],"起始_状态":
"q0", " 接受 _ 状态 ": ["q1", ...]," 转移 ": { "
q0": { "a": "q1", "b": "q0" }, "q1": { "a": "q1", "
b": "q2" } } }

格式规则和约束（必须严格遵守）：- JSON 对象只能包含上述五个键，不能有其他内容。- 确定有限自动机必须是全定义的：对于每个状态和「字母表」中的每个符号，都必须恰好有一个目标状态。- 使用简短的状态名称，如「q0」「q1」「q2」……- 所有值都必须是有效的 JSON 类型。- 不要包含注释、尾随逗号或额外文本。- 如果构造失败，请输出以下机器可读的失败 JSON：

{"error":"无法_构造","reason":"<单样本-行_原因>"}

边界情况指南（零样本）：- 只输出 JSON 对象（或上述失败 JSON）。- JSON 前后都不得有诊断文本。- 实验脚本最多重试三次。

D.1.3 思维链单样本提示

系统：
你是形式语言和有限自动机方面的专家。当被要求生成 DFA 转移表时，请在内部逐步思考（思维链），但不要输出任何推理内容。仅输出一个符合下述模式的有效 JSON 对象。

用户：
下面是一个示例（单样本），接下来是实际输入。请仔细阅读示例——它展示了精确的 JSON 模式、格式，以及完备性规则。看完示例后，仅针对实际输入生成确定有限自动机 JSON。

===示例（单样本） ===正则表达式：(a|b)* 字母表： ["a","b"]

正确的确定有限自动机（仅 JSON）：

```
{
  "状态": ["q0"],
  "字母表": ["a", "b"],
  "start_状态": "q0",
  "接受_状态": ["q0"],
  "transitions": {
    "q0": { "a": "q0", "b": "q0" }
  }
}
```

===示例结束 ===

现在是实际输入：

正则表达式：{{REGEX}}
字母表：{{ALPHABET}}

任务：
1. 在内部（思维链中）推导出一个正确的确定有限自动机（DFA），该自动机能够识别该正则表达式。
2. 不要输出任何中间推理过程或解释。
3. 仅输出一个 JSON 对象，其中包含 DFA 转移表，除此之外不要包含任何内容。JSON 必须符合以下确切模式（键和类型）。

必需的 JSON 模式（确切的键和类型）：

```
{
  "states": ["q0", "q1", ...],
  "alphabet": ["a", "b", ...],
  "start_state": "q0",
  "accept_状态": ["q1", ...],
  "转移": {
    "q0": { "a": "q1", "b": "q0" },
    "q1": { "a": "q1", "b": "q2" }
  }
}
```

格式规则和约束（必须严格遵守）：- JSON 对象必须且只能包含上述五个键，不得包含其他内容。- 确定有限自动机必须是全定义的：对于每个状态以及“alphabet”中的每个符号，都必须有恰好一个目标状态。
- 使用简短的状态名称，例如 "q0", "q1", "q2", ...（不含空格或特殊字符）。
- 所有值都必须是有效的 JSON 类型（数组、字符串、对象）。请勿包含注释或尾随逗号。
- JSON 对象前后都不得包含任何文本（不加反引号，不使用代码围栏，不要附加额外说明）。
- 如果无法给出正确的确定有限自动机，切勿输出 {}。而应尝试给出一个精简但有效的确定有限自动机。只有在确实无法构造任何确定有限自动机时，才输出一份机器可读的失败 JSON：{"error":"无法_构造","reason":"<单行_原因>"}（这是允许的，但不推荐——建议生成紧凑型确定性有限自动机。）
- 如果确定有限自动机很大，请生成一种紧凑编码使用简短的状态名称（q0、q1、q2.....），并列出每个符号对应的所有转移。

边界情况指南（单样本）：
- 上面的示例展示了确切的输出形状和格式（包括空格和换行并不重要，但内容必须是有效的 JSON）。
- 实验脚本最多会重试 3 次如果首次输出无效。只输出 JSON 对象（或简短的错误 JSON）——不得包含其他文本。

重要：只输出 JSON 对象（或少量错误的 JSON）。不要输出任何其他文本。

D.1.4 思维树（ToT）直接分支提示

系统：你是形式语言和有限自动机领域的专家。当被要求生成 DFA 转移表时，请在内部逐步思考（思维链），但不要输出任何推理内容。只输出一个符合下文所述模式的有效 JSON 对象。用户：给定以下正则表达式和字母表：REGEX: {{REGEX}} ALPHABET: {{ALPHABET}} 任务：1. 在内部推导出一个正确的确定有限自动机（DFA），使其识别的语言恰好是该正则表达式表示的语言。2. 在内部使用任何可靠的形式推理，但不要输出任何中间步骤。3. 只输出一个包含 DFA 转移表的 JSON 对象，不得包含其他内容。

必需的 JSON 模式（键和类型必须完全一致）：

```
{
  "状态": ["q0", "q1", ...],
  "字母表": ["a", "b", ...],
  "初始_状态": "q0",
  "接受_状态": ["q1", ...],
  "转移": {
    "q0": { "a": "q1", "b": "q0" },
    "q1": { "a": "q1", "b": "q2" }
  }
}
```

格式规则和约束：- 仅输出 JSON 对象，不得包含任何额外文本

- 确定有限自动机必须是全定义的。- 请使用简短的状态名称 q0、q1、q2.....- 不要包含注释、解释或尾随逗号。

- 如果构造困难，也请输出一个简洁但有效的确定有限自动机，而不是空对象。

D.1.5 思维树（ToT）最小化提示

系统：
你是确定有限
自动机方面的专家。请在内部设法构造一个
能够识别给定
正则表达式且接近最小化的正确确定有限自动机
在状态数量方面。你可以在内部
应用最小化技术，例如状态
等价或划分细化。不要
输出任何推理过程。

用户：
给定以下正则表达式和
字母表：

正则表达式：{{REGEX}}
字母表：{{ALPHABET}}

任务：。
1 在内部构造一个正确的确定有限自动机，用于识别该
正则表达式。
2. 如有可能，在内部对确定有限自动机进行最小化。
3. 确保确定有限自动机是全定义的。
4. 仅输出最终的最小化后的确定有限自动机，格式为 JSON
对象，并符合下方的模式。

必需的 JSON 模式：
```
{
  "状态": ["q0", "q1", ...],
  "字母表": ["a", "b", ...],
  "start_状态": "q0",
  "accept_状态": ["q1", ...],
  "转移": {
    "q0": { "a": "q1", "b": "q0" },
    "q1": { "a": "q1", "b": "q2" }
  }
}
```

格式约束：
- 仅输出 JSON 对象。
- 不得添加解释或评论。
- 确定有限自动机必须是全定义的。

D.1.6 思维树（ToT）导数法提示

系统：
你是形式语言理论专家。
在内部使用正则
表达式导数（Brzozowski 导数或
Antimirov 导数）构造一个确定有限自动机。每个确定有限自动机状态
都对应于原始正则表达式的一个导数
正则表达式。不要输出任何
中间推理。

用户：
给定以下正则表达式和
字母表：

正则表达式：{{REGEX}}
字母表：{{ALPHABET}}

任务：
1. 在内部计算正则
表达式相对于
字母表。
2. 在内部从正则表达式导数构建相应的确定有限自动机
这些导数。
3. 确保该确定有限自动机是全定义的。
4. 仅输出最终的确定有限自动机，作为一个 JSON 对象
并且符合以下确切模式。

所需的 JSON 模式：
```
{
  "states": ["q0", "q1", ...],
  "字母表": ["a", "b", ...],
  "初始_状态": "q0",
  "接受_状态": ["q1", ...],
  "转移": {
    "q0": { "a": "q1", "b": "q0" },
    "q1": { "a": "q1", "b": "q2" }
  }
}
```

约束：
- 仅输出有效的 JSON。
- 确定有限自动机必须是全定义的。
- 不得包含解释性文本。
- 使用紧凑的状态命名。

D.1.7 思维树（ToT）Thompson 构造提示

系统：
你是自动机理论方面的专家。在内部
遵循 Thompson 构造法：首先
将正则表达式转换为
ε-非确定有限自动机，然后应用子集
构造法得到确定有限自动机，最后
确保该确定有限自动机是全定义的。不要输出任何
中间结构或推理。

用户：
给定以下正则表达式和
字母表：

正则表达式：{{REGEX}}
字母表：{{ALPHABET}}

任务：
1. 使用 Thompson 构造在内部构建一个 ε-非确定有限自动机
使用 Thompson 构造。
2. 使用子集构造法在内部将非确定有限自动机确定化
。
3. 必要时，通过添加一个汇状态，使确定有限自动机全定义
必要时注明状态。
4. 仅输出最终的确定有限自动机，且必须是单个 JSON
对象，并符合以下模式。

必需的 JSON 模式：
```
{
  "states": ["q0", "q1", ...],
  "字母表": ["a", "b", ...],
  "起始_状态": "q0",
  "accept_状态": ["q1", ...],
  "转移": {
    "q0": { "a": "q1", "b": "q0" },
    "q1": { "a": "q1", "b": "q2" }
  }
}
```

格式规则：
- 仅输出 JSON 对象。
- 确定有限自动机必须是全定义的。
- 不要添加解释、注释或额外文本。
- 使用简短的状态名称 q0、q1、q2, . . .

D.2 基于提示的框架提示

本节列出了用于评估模型能否在引导式反馈下纠正自身错误的提示。

D.2.1 初始提示（无提示）

构造一个确定有限自动机（DFA），使其恰好识别字母表 {a, b} 上的以下语言：L =(b)*(a)(a +ba)*(bb)(a +b)* +(b)*b(b)*a(a + b)* 要求：- DFA 必须是全定义的。- 使用简短的状态名称（q0、q1、q2.....）。

参数	GPT-5.1	Gemini-2.5	Grok-4.1
API 端点	公开	公开	公开
模型变体	GPT-5.1	Gemini-2.5-Flash	Grok-4.1-fast-reasoning
温度	0.0	0.0	0.0
最大输出令牌数	4000	4000	4000
推理开关	未公开	未公开	未公开
提示模板	相同	相同	相同
重试次数	3	3	3
超时时间	120s	120s	120s

表5：各模型提供商所使用的 API 参数。未启用任何特定于提供商的推理或审议模式。

- 指定初始状态、接受状态和转移。- 仅输出最终的 DFA。

D.2.2 提示1：反例协议

给定以下语言：L = (b)*(a)(a + ba)*(bb)(a + b)* + (b)*b(b)*a(a +b)* 你之前构造的确定有限自动机无法正确识别该语言。具体而言，以下字符串属于 L，但会被你的确定有限自动机拒绝：- ba - bba - baa 请修改或重新构造确定有限自动机，使其接受 L 中所有有效字符串，同时继续拒绝无效字符串。只输出修正后的确定有限自动机。

D.2.3 提示2：错误定位

给定以下语言：L = (b)*(a)(a + ba)*(bb)(a + b)* + (b)*b(b)*a(a +b)* 你修改后的确定有限自动机仍然存在错误。至少有一个转移未能正确体现语言定义，导致某些字符串被错误地接受或拒绝。请重新检查确定有限自动机的结构和转移指派，并提供一个能够恰好识别 L 的修正后确定有限自动机。只输出确定有限自动机。

D.2.4 最终提示：明确披露错误

给定以下语言：L = (b)*(a)(a + ba)*(bb)(a + b)* + (b)*b(b)*a(a +b)* 剩余错误源于对正则表达式的解读不完整。具体来说，克莱尼星号子表达式 (a +ba)* 允许任意重复和组合，而这些都必须由自动机完整捕获。由于解读不完整，你的确定有限自动机错误地接受了字符串：aabab

请基于对所有子表达式及其相互作用的正确解读，重建确定有限自动机。仅输出最终修正后的确定有限自动机。

E 附录：评估与验证代码

自动化验证流程。 自动化验证流程提供了一种有原则且可复现的机制，用于评估大语言模型生成的确定有限自动机是否正确。该流程结合了（i）严格的结构验证，以确保语法正确、全定义且转换格式规范；以及（ii）与真实正则表达式进行行为等价性测试。行为验证通过穷举所有长度不超过固定上限的字符串来完成，并辅以对较长字符串的随机抽样，从而能够检测局部和全局的接受错误。此设计确保报告的正确性反映的是语义上的语言等价，而非表面合理性。

该验证器与模型无关且具有确定性：给定相同的确定有限自动机和正则表达式，始终会产生相同的结果。检测到不匹配时，它会明确记录反例，以支持透明的错误分析和独立验证。所有验证产物，包括逐实例报告和汇总摘要，都会保存到磁盘，以便复现和事后检查。

与此同时，验证过程也有其固有局限。穷举测试受最大字符串长度限制；尽管随机抽样扩大了覆盖范围，但仅凭有限测试无法判定确定有限自动机与正则表达式是否形式等价。因此，被归类为正确的确定有限自动机应被理解为在已测试范围内很可能正确，而非已被形式化证明等价。此外，

验证器假定宿主正则表达式引擎所解释的正则表达式语义是正确的，而这种语义在某些边缘情况下可能与理论自动机语义不同。主要评估中通过补充人类专家检查缓解了这些局限，但在解读自动化结果时，仍须将其视为重要考量。

E.1 自动化确定有限自动机验证流程

```
#!/usr/bin/env python3
"""
validate_dfa_outputs.py

实验中使用的自动化确定有限自动机验证流程
    。

验证协议：
1. 模式验证（JSON 结构 + 必需
    字段）。
2. 全定义性验证（每个状态对每个
    符号都有一个转移）。
3. 针对
    真实正则表达式的行为等价性测试：
  - 穷举至
    MAX_穷举_长度。
  - 针对更长
    字符串。

输出：
- 每个确定有限自动机的详细验证报告（JSON）。
- 汇总表（CSV + JSON）。

此脚本与模型无关，且
    与数据集无关。
"""

导入 os
导入 json
导入 re
导入 random
导入 csv
从 itertools 导入 product
从 typing 导入 列表,                字典,   元组

#  ==============================
# 配置（在此编辑）
#  ==============================

表格_目录 = "输出表格目录"
原始_目录 = "输出原始数据目录"
验证_目录 = "输出验证目录"

MAX_穷举测试    _长度 = 6          # 穷举测试
    长度为 0..6 的字符串
N_随机数 = 2000                    # 随机数的数量
    测试字符串
随机字符串最大长度 = 15_     _     # 最大长度
    随机字符串
反例最大数量 = 100_                # 上限存储
    每个确定有限自动机的反例数量
随机种子 = 42_                     # 可复现性

#  ==============================
# 初始化
#  ==============================

os.makedirs(VALID_DIR, exist_ok=True)
random.seed(RANDOM_SEED)

#  ==============================
# 实用工具
#  ==============================

def load_json(path: str):
    with open(path,     "r",  encoding="utf-8") as f:
        return json.load(f)

def save_json(path: str,          obj):
    with open(path,     "w",  encoding="utf-8") as f:
        json.dump(obj, f, indent=2, ensure_ascii=False) def is_valid_schema(dfa: Dict) -> Tuple[bool, str]: 必需项 = {"states", "alphabet", "start_state", "accept_states", "transitions"} if not isinstance(dfa, dict): return False, " 不是 _json_ 对象 " if not required.issubset(dfa.keys()): return False, "缺少_必需_键" if not isinstance(dfa["states"], list): return False, "states_不是_列表" if not isinstance(dfa["alphabet"], list): return False, "alphabet_不是_列表" if dfa["start_state"] not in dfa["states"]: return False, "无效的_start_state" if not isinstance(dfa["accept_states"], list): return False, "accept_states_不是_列表" if not isinstance(dfa["transitions"], dict): return False, "transitions_不是_字典" return True, "ok" def check_totality(dfa: Dict): 缺失项 = [] 无效的_目标 = []for s in dfa["states"]: if s not in dfa["transitions"]: missing.append({"state": s, "reason": "没有_transition_区块"}) continue for a in dfa["alphabet"]: if a not in dfa["transitions"][s]: missing.append({"state": s, "symbol": a}) else: tgt = dfa["transitions"][s][a]if tgt not in dfa["states"]: invalid_targets.append({"state": s, "symbol": a, "target": tgt}) return missing, invalid_targets def simulate_dfa(dfa: Dict, word: str) -> bool: cur = dfa["start_state"]for ch in word: if ch not in dfa["alphabet"]: return False cur = dfa["transitions"].get(cur, {}).get(ch) if cur is None: return False return cur in dfa["accept_states"]def generate_exhaustive_strings(alphabet: List[str], max_len: int) -> List[str]: 字符串 = [""]for L in range(1, max_len + 1): for tup in product(alphabet, repeat=L): strings.append("".join(tup)) return strings def generate_random_strings(alphabet: List[str], n: int, max_len: int) -> List[str]: out = set() while len(out) < n: L = random.randint(0, max_len) out.add("".join(random.choice(alphabet) for _ in range(L))) return list(out) #  ==============================# 主要验证循环 #  ==============================def main(): table_files = [ os.path.join(TABLES_DIR, f) for f in os.listdir(TABLES_DIR) if f.endswith(".json")]摘要 = []
```

```
for table_path in table_files: 基本名称 = os.path.splitext(os.path.basename (table_path))[0] print(f"[正在验证] {base}") 报告 = { "id": base, " schema_有效": False, "schema_错误": None, "全定义性_缺失项": [],"全定义性_无效_目标": [],"已测试_字符串": 0, "反例": [],"判定": "" } try: dfa = load_json(table_pat h) except Exception as e: report["schema_错误"] =f"j son_加载_错误: {e}" report["判定"] = "无效_json" save_ json(os.path.join(VALID_DIR, base + "_报告.json"), report) continue ok, reason = is_valid_schema(dfa) report["schema_有效"] = ok report["schema_错误"] = None if ok else reason if not ok: report["判定"] = " schema_无效" save_json(os.path.join(VALID_DIR, base + "_报告.json"), report) continue missing, invalid = c heck_totality(dfa) report["全定义性_缺失项"] = missing report["全定义性_无效_目标"] =invalid # 查找匹 配的原始文件以获取正则表达式 regex = None for rf in os.listdir(RAW_DIR): if rf.startswith(base): raw =load_ json(os.path.join(RAW_DIR, rf)) regex = raw.get(" regex") break if regex is None: report["判定"] ="没有_真 值_正则_表达式" save_json(os.path.join(VALID_DIR, base + "_报告.json"), report) continue try: pattern = re.compile(f"^({regex})$") except Exception as e: report[" 判定"] =f"正则表达式_编译_错误: {e}" save_ json(os.path.join(VALID_DIR, base + "_报告.json"), report) continue 穷举测试 = generate_exhaustive_ strings (dfa["alphabet"], 最大_穷举_长度) 随机测试 =_ generate_random_strings(dfa["alphabet"],随机测试数量, MAX_随机_长度) 测试集 = 穷举测试 + [s for s in 随机测 试_if s not in 穷举测试]for w in 测试集: dfa_接受 = simulate_dfa(dfa, w) 正则表达式_接受 = pattern.fullmatch(w) is not None report["已测试_字符串" ] += 1if dfa_接受 != 正则表达式_接受: report["反例"]. append({ "string": w, "确定有限自动机_接受": dfa_接受, "正 则表达式_接受": regex_接受 })
```

```
                if len(report["反例"])
    >= MAX_反例:
                    中断

        如果缺失或无效:
            report["判定"] = "非_全定义"
        否则如果 report["反例"]:
            报告["判定"] = "错误"
        否则:
            报告["判定"] = "可能_正确"

        保存_json(os.path.join(VALID_DIR,  基本名称 + "_report.json"),  报告)

        摘要.append({
            "id": 基本名称,
            "模式_有效": ok,
            "全定义性_正常": not (missing or invalid),
            "已测试的_字符串": 报告["已测试的_字符串"],
            "反例数量_": len(报告["反例"]),
            "判定": 报告["判定"]
        })

    # ==============================
    # 编写汇总表
    # ==============================

    with open(os.path.join(VALID_DIR, "summary.csv"), "w", newline="", encoding="utf-8") as f:
        写入器 = csv.DictWriter(
            f,
            字段名称=["id", "模式_有效性", "全定义性_ok", "测试过的_字符串", "数量_反例", "判定"]
        )
        写入器.writeheader()
        对摘要中的每一行:
            写入器.writerow(row)

    保存_json(os.path.join(VALID_DIR, "summary.json"), 摘要)

    print("[DONE] DFA 验证已完成。")

if __name__ == "__main__":
    main()
```

F 附录：实验执行代码

实验执行代码。 我们提供了用于查询语言模型并收集 DFA 输出的完整实验执行代码。该脚本与模型无关，并统一用于所有受评估的模型和提示策略，仅模型标识符和提示模板有所不同。它强制采用确定性解码、稳健的重试逻辑、严格的 JSON 提取，并完整记录原始输出、解析后的 DFA 和元数据。无论成功还是失败，所有运行都会写入输出，从而能够对正确性和失败模式进行可复现分析。

优点与局限。 该执行流程通过确定性解码和统一的评估设置确保可复现性，并通过重试和结构化输出验证增强稳健性。然而，它不会尝试

修正或后处理无效的确定有限自动机，而是依赖下游自动化与人工验证来确保语义正确性。超时事件和格式错误的输出均被视为未成功的尝试，这反映的是实际部署约束，而非纯粹的理论能力。

F.1 实验执行代码

```
#!/usr/bin/env python3
"""
run_experiment_tot.py

通用思维树（ToT）实验运行器
    用于基于大语言模型的确定有限自动机构造。

功能：
- 模型无关（OpenAI / 基于 API 的大语言模型）
- 确定性解码（温度 = 0）
- 稳健的 JSON 提取和重试逻辑
- 严格的输出日志记录（原始内容、解析结果、元数据）
- 始终写入输出（无论成功还是失败）
- 可通过常量复现并进行配置

此脚本在各个模型和
    提示策略中通用，
只需更改模型标识符和提示
    模板。
"""

import os
import json
import time
import re
from datetime import datetime
from typing import Optional
from openai import OpenAI, OpenAIError

# ==============================
# 配置（模型无关）
# ==============================

模型 = os.environ.get("LLM_MODEL", "gpt-5.1")
温度 = 0.0
最大令牌数 = 4000

数据集_路径 = "data/analysis_data.json"

提示词_文件 = {
    "direct": "prompts/tot_direct.txt",
    "minimal": "prompts/tot_minimal.txt",
    "导数": "prompts/tot_derivative.txt",
    "汤普森": "提示词/tot_汤普森.txt",
}

输出_根目录 = "outputs/tot_experiments"

重试次数 = 3
重试_间隔 = 1.5
速率_限制_休眠 = 0.3

# ==============================
# 初始化
# ==============================

客户端 = OpenAI()

时间戳 = datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")
OUTPUT_RAW_DIR = os.path.join(OUTPUT_ROOT, MODEL, "raw")
输出_表格_目录 = os.path.join(输出_根目录, 模型, "表格")
输出_元数据_目录 = os.path.join(输出_根目录, 模型, "元数据")

对于 d，取值为 [OUTPUT_RAW_DIR、OUTPUT_TABLE_DIR, OUTPUT_META_DIR]:
    os.makedirs(d, exist_ok=True)

项目_根目录 = os.path.abspath(os.path.join(os.path.dirname(__文件__), ".."))

# ==============================
# 实用工具
# ==============================

定义 now_ts():
    返回 datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")

定义安全_文件名(s: 字符串) -> 字符串:
    返回 re.sub(r"[^0-9A-Za-z._-]", "_", s)[:200]

定义加载_json(路径):
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

def load_prompt(path):
    with open(path, "r", encoding="utf-8") as f:
        return f.read()

def save_json(path, obj):
    使用 open(path, "w", encoding="utf-8") as f:
        JSON 序列化(obj, f, indent=2, 确保_ascii=False)

定义 try_parse_json(s: str) -> Optional[字典]:
    如果 not s:
        返回 None
    s = re.sub(r"```(?:json)?", "", s, flags=re.IGNORECASE).strip("` \n")
    try:
        return json.loads(s)
    except Exception:
        pass
    m = re.search(r"\{[\s\S]*\}", s)
    if m:
        try:
            return json.loads(m.group(0))
        except Exception:
            pass
    return None

# ==============================
# TOKEN DISPATCH
# ==============================

def completion_kwargs():
    if MODEL.startswith("GPT-5"):
        return {"max_completion_tokens": MAX_TOKENS}
    return {"max_tokens": MAX_TOKENS}

# ==============================
# 模型访问检查
# ==============================

def check_模型_访问():
    try:
        r = client.chat.completions.create(
            模型=MODEL，
            消息=[{"role": "user", "内容": "ping"}]，
            温度=0.0,
            **补全_kwargs()
        )
        实际 = getattr(r, "模型", None)
        如果 actual 为 None 或者不 actual.startswith(MODEL.split("-")[0]):
            则引发 RuntimeError(f"模型不匹配：请求的={MODEL}, 实际={actual}")
    except OpenAIError as e:
        raise RuntimeError(f"模型访问失败：{e}")

# ==============================
# 提示工程
# ==============================

def build_messages(模板, 正则表达式, 字母表):
    已填充的 = (
        模板
        .replace("{{REGEX}}", 正则表达式)
        .replace("{{ALPHABET}}", json.dumps(字母表))
    )
```

```
    return [{ "role": "system", "content": ( "你是形式语言和自动机领域的专家。" "请根据需要在内部进行推理，但只输出 DFA JSON。" ) }, {"role": "user", "content": filled}]def call_model(messages): r = client.chat.completions.create( model=MODEL, messages=messages, temperature=TEMPERATURE, **completion_kwargs() ) text = r.choices[0].message.content or "" meta = { "requested_model": MODEL, "actual_model": getattr(r, "model", None), "finish reason":_ r.choices[0].finish_reason, "usage": getattr(r, "usage", None) } return text, meta # ============================# 核心循环 # ============================def run_single_branch(entry, branch, prompt_template): regex_id = entry["id"]regex = entry["regex"] alphabet = entry.get("alphabet", []) ts = now_ts() messages = build_messages(prompt_template, regex, alphabet) raw_text, meta, parsed = "", None, None attempts = 0while attempts <RETRIES: attempts += 1try: raw_text, meta = call_model(messages) except Exception as e: meta = {"error": str(e)} time.sleep(RETRY_SLEEP) continue parsed = try_parse_json(raw_text) if parsed: break messages.append({ "role": "user", "content": "只输出 DFAJSON。不要解释。" }) time.sleep(RETRY_SLEEP) base =safe_filename(f"{regex_id}_{branch}_{ts}") save_json(os.path.join(OUTPUT_RAW_DIR, base + ".json"), { "id": regex_id, "branch": branch, "regex": regex, "alphabet": alphabet, "raw_output": raw_text, "attempts": attempts, "meta": meta, "model": MODEL, "timestamp": ts }) save_json(os.path.join(OUTPUT_META_DIR, base + "_meta.json"), meta) table

        "错误": "生成_失败", "原因": "无效或非 JSON 输出", "正则表达式_id": 正则表达式_id, "分支": 分支, "模型": 模型, "尝试次数": 尝试次数 } 保存_json(os.path.join(OUTPUT_TABLE_DIR, 基本名称 + ".json"), table_obj) # ============================# 主程序 # ============================def main(): 检查_模型_访问权限() 数据集 =加载_json(os.path.join(PROJECT_ROOT, DATASET_PATH)) 提示词 = { k: 加载_提示词(os.path.join(PROJECT_ROOT, v)) for k, v in PROMPT_FILES.items() } for entry in 数据集中: for 分支, tmpl in 提示词.items(): 运行_单个_分支(entry, 分支, tmpl) time.sleep(RATE_LIMIT_SLEEP) if name ==" main ":__ __ __ __ main()
```

G 附录：大语言模型生成的确定有限自动机中发现的常见问题

本附录总结了不同提示策略下，大语言模型生成的确定有限自动机（DFA）中反复出现的错误模式。每类问题都通过一个具体示例及相应的确定有限自动机图加以说明。本附录旨在进行诊断：刻画系统性的构造失败，而非孤立的错误。

已发现问题类型摘要

问题 1：忽视克林闭包语义

图 5 展示了通过直接构造法为语言 $L = b(a \mid b)^*ab$生成的确定有限自动机。模型正确识别了终止模式 `ab`，但未能考虑克林闭包 $(a \mid b)^*$，因此该确定有限自动机接受以 `ba` 结尾的字符串，而没有遵循所要求的后缀 `ab`。这一错误反映出模型对克林闭包下无界重复的语义理解不完整。

问题 2：违反布佐夫斯基导数语义

构造方法： 基于导数的方法构造。图6 展示了为语言 $L = b(a \mid b)^*ab$生成的基于导数的确定有限自动机。该构造未能将语义等价的 Brzozowski 导数规范化，并错误地重新引入

表 6：常见 DFA 构造问题摘要

问题	已识别的问题
问题 1	忽视克林闭包语义
问题 2	违反布佐夫斯基导数语义
问题 3	最小化之前初始确定有限自动机构造中的错误
问题 4	对目标语言之外字符串的过度接受
问题5	连接下无法保持约束
问题 6	引入冗余或不可达状态

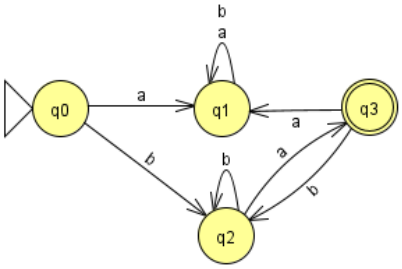


图 5：通过直接构造为$L = b(a \mid b)^*ab$生成的确定有限自动机。

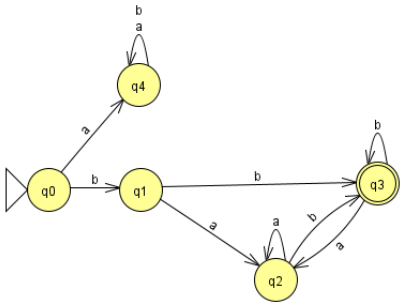


图 6：为 $L = b(a \mid b)^*ab$生成的基于导数的确定有限自动机。

已消耗的前缀。因此，不同的剩余语言被视为独立的状态，违反了正则表达式导数的形式语义，并生成了错误的确定有限自动机。

问题 3：初始确定有限自动机构造中的错误

构造方法：基于最小化的构造方法。图 7展示了在最小化之前为语言 $L = b(a \mid b)^*ab$生成的确定有限自动机。由于状态语义和转移指派有误，初始确定有限自动机构造错误。随后的最小化进一步放大了这些错误：它合并了本应保持区分的状态，最终得到无效的最小化后的确定有限自动机。

问题 4：字符串过度接受

构造方法：基于导数的构造。图8 展示了一个过度接受的确定有限自动机

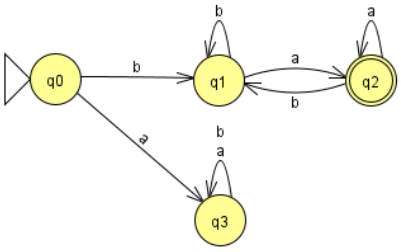


图 7：为 $L = b(a \mid b)^*ab$ 生成的最小化前的确定有限自动机。

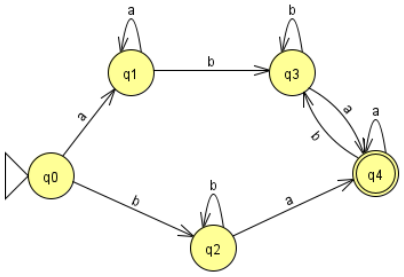


图 8：为目标语言生成的过度接受确定有限自动机。

为该语言生成

$$L = ((a \mid b)^*(aa^*bb^* \mid b(b \mid ab)^*aaa^*bb^*)(aa^*bb^*)^* \mid (a \mid b)^*b(b \mid ab)^*a)aa^*.$$

所构造的确定有限自动机接受了不属于目标语言的字符串：形式上，存在一个字符串 $w \in \Sigma^*$，使得 $w \in L$（确定有限自动机），而 $w \notin L$。这种过度接受现象源于：在未证明语言等价的情况下，将语法上相似但语义上不同的推导合并。在此示例中，形式为 bb^*a 的字符串被错误地接受。

问题5：连接下无法保持约束

构造方法： 直接构造法和汤普森构造法。图9 展示了一个在连接过程中无法保留约束的确定有限自动机，适用于

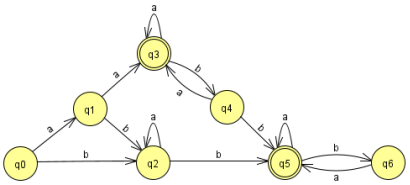


图9：在连接下未能保留约束的确定有限自动机。

语言

$$L = ((a^*ba^*b \mid (a^*ba^*b \mid a^*a)(b \mid a)^*a))a^*.$$

构造出的确定有限自动机能够正确地单独执行各个子约束，例如 a^*ba^*b 和 a^*a 。然而，当这些约束通过连接组合时，全局边界条件无法得到保留。

问题 6：引入冗余或不可达状态

构造方法： 直接构造、汤普森构造、基于导数的方法和基于最小化的方法构造。

图 10 展示了一个包含冗余和不可达状态、用于该语言的确定有限自动机

$$\begin{aligned} L = ((&a^*b \mid (a^*a)a^*b \mid (a^*b)b^*b \\ &\mid ((a)^*ba \mid (a^*a)a^*ba \mid (a^*b)b^*ba) \\ &(aa \mid b)^*a)(a \mid b) \mid (a^*b)b^*)a. \end{aligned}$$

构造出的确定有限自动机包含多个从初始状态不可达的状态，因此这些状态对语言识别没有贡献。在某些示例中，其中一些状态虽然没有任何入边，却仍是接受状态。这表明模型引入了与正则表达式句法成分相对应的状态，却没有进行可达性或有用性分析，因而产生了冗余且在语义上无关的状态。

如需整体查看，图11 展示了语言 L_9 在 ToT 方法下的常见错误。

H 附录： L_9的基于导数的方法确定有限自动机输出

本附录报告了基于导数的确定有限自动机构造为语言 L_9生成的确切输出。内容逐字呈现模型返回的结果，未经规范化、修正或后处理。

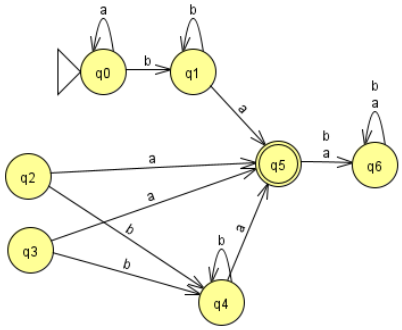


图 10：包含冗余状态和不可达状态的确定有限自动机。

H.0.1 模型输出（ L_9的导数构造）


```
{
  "method": "导数"
  "derivatives": {
    "q0": {
      "正则表达式": "b(a|b)*ab",
      "可空": false
    },
    "q1": {
      "regex": "(a|b)*ab",
      "可空": false
    },
    "q2": {
      "regex": "b(a|b)*ab+(a|b)*b",
      "可空": false
    },
    "q3": {
      "regex": "(a|b)*ab+(a|b)*",
      "可空": 是
    },
    "q4": {
      "正则表达式": "\phi",
      "可空": 否
    }
  },
  "dfa": {
    "状态": [
      "q0",
      "q1",
      "q2",
      "q3",
      "q4"
    ],
    "字母表": [
      "a",
      "b"
    ],
    "start_状态": "q0",
    "接受_状态": [
      "q3"
    ],
    "转移": {
      "q0": {
        "a": "q4",
        "b": "q1"
      },
      "q1": {
        "a": "q2",
        "b": "q3"
      },
      "q2": {
        "a": "q2",
        "b": "q3"
      },
      "q3": {
        "a": "q2",
        "b": "q3"
      },
      "q4": {
        "a": "q4",
        "b": "q4"
      }
```

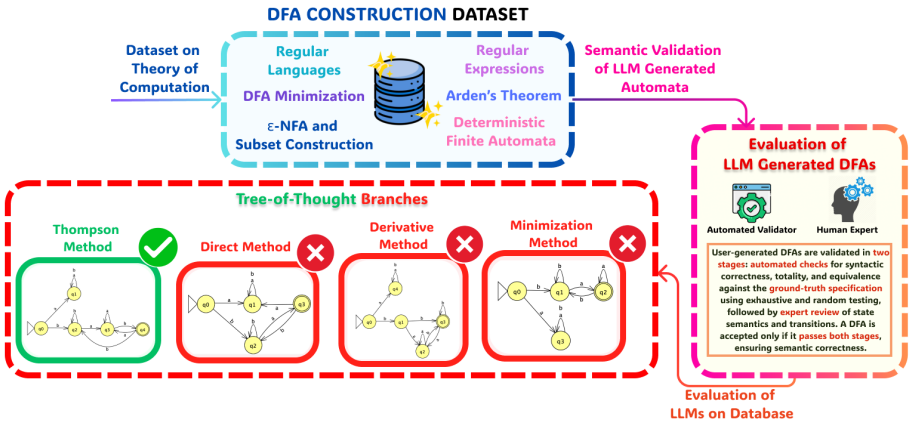


图 11：DFA 构造数据集与评估流程，以及按照针对语言 $L_9 = b(a+b)^*ab$ 的构造策略，通过思维树提示生成的代表性 DFA 输出。

```
        }
    }
}
```

I 附录：确定性与随机性控制

所有实验都力求尽可能具有确定性和可复现性，同时遵守黑箱大语言模型 API 的约束。

解码确定性。 对于所有模型和提示策略，解码温度固定为 0.0，且未修改任何随机采样参数。这确保对于给定的提示词和模型端点，生成过程在服务提供商支持的范围内具有确定性。

提示词确定性。 每个提示模板都在代码仓库中固定并进行版本控制。对于思维树和思维链设置，所有运行均重复使用固定且针对特定策略的提示模板。在单次模型调用期间，不会进行自适应提示词修改或动态提示生成。

重试策略。 为缓解暂时性 API 故障和格式错误，每个查询都会使用完全相同的提示词，最多重试固定次数。仅当输出无效（例如，非 JSON 格式或违反模式）时才会触发重试，且不会引入额外的随机性或替代提示词。

验证随机性控制。行为等价性测试采用混合验证策略，包括：(i) 对所有长度不超过固定上限的字符串进行穷举，以及 (ii) 对更长的字符串进行随机测试。所有随机测试均使用固定的随机种子执行，以确保验证结果完全可复现。

API 调用中的非确定性。 尽管采取了上述控制措施，对同一模型重复进行 API 调用仍可能产生不同的输出，这是因为服务提供商一侧存在未公开的非确定性因素（例如模型更新、推理时优化或分布式服务）。这种变化被视为已部署大语言模型系统的固有属性，而非实验噪声。

报告政策。 所有报告的结果均对应评估期间 API 返回的实际输出，未经过事后筛选。超时、格式错误和无效输出都会被明确记录，并依据正文所述的评估协议进行计数。

J 附录：运行环境

所有实验均使用自定义 Python 脚本在本地工作站上执行。实现采用 Python（版本 3.10+）编写，且仅依赖标准库和官方模型 API。未使用任何专有或未公开的工具。

所有大语言模型交互均通过 API 调用进行，并采用确定性解码设置（温度设为 0）。实验按顺序进行，并实施明确的速率限制和重试逻辑，以确保稳定性和可复现性。每次调用模型都是无状态的：对话

上下文不会在调用之间保留。这包括基于提示的框架，在该框架中，每个阶段都会重新提供完整的问题规格说明。

模型输出、元数据（包括令牌用量和结束原因）、解析后的确定有限自动机表以及验证报告均以结构化 JSON 格式记录到磁盘。运行环境严格执行输出处理规则：每次尝试都会被记录，包括格式错误的输出、超时或生成失败。

验证和评估通过单独的自动化流水线离线进行，该流水线通过穷举和随机测试，将确定有限自动机的行为与真实正则表达式进行比较。验证中使用的所有随机性（例如随机字符串生成）均通过固定的随机种子加以控制。

未进行微调、模型端配置更改或系统级优化。因此，性能差异反映的是模型在相同运行条件下的内在行为，而非环境变异。