arXiv:2602.11202v2 [cs.LO] 17 Mar 2026

interwhen：一种结合测试时监控器的可验证推理可泛化框架

Vishak K Bhat[1], * Prateek Chanda[2], * Vijval Ekbote[1], * Ashmit Khandelwal[1]Maitreyi Swaroop[3] Vineeth N. Balasubramanian[1] SubbaraoKambhampati[4]Nagarajan Natarajan[1], † Amit Sharma[1], †

[1]微软研究院, 印度； [2]印度理工学院孟买分校； [3]卡内基梅隆大学； [4]亚利桑那州立大学

摘要

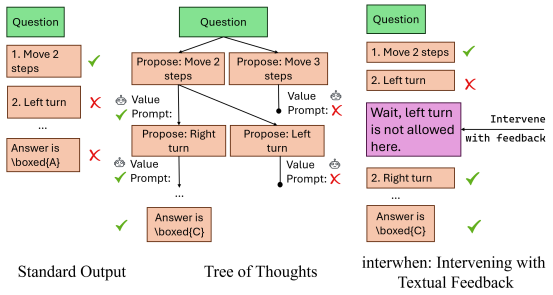
推理模型会生成包含大量中间决策和工具调用的长轨迹，因此，测试时验证对于确保正确性变得日益重要。现有方法要么只验证最终答案，因而会漏掉早期错误；要么依赖分支并验证策略，探索多条轨迹，计算成本则大幅增加。我们提出 interwhen，这是一种单轨迹验证框架，通过针对中间可验证属性提供反馈来引导模型行为。我们的方法应对两项关键挑战。首先，从推理轨迹中提取中间解通常需要提示工程，或将任务外部分解为固定步骤，这可能会限制模型的推理策略。我们采用不同做法，定期轮询推理轨迹和 *fork* 推理，以恢复中间解，而不强加任何预定义结构。其次，频繁调用验证器会增加延迟；为此，我们异步运行验证器，仅在检测到错误时中断主轨迹，否则不影响生成。这种设计提升了可靠性和效率，并自然支持基于近期中间解一致性的 *early stopping* 。在代码生成、算术、逻辑推理和空间推理基准测试中，与标准思维链执行相比，interwhen 的准确率最高提升 15 个百分点，同时令令牌计算成本保持在 1.5× 以内。此外，在每个数据集上，与现有测试时验证方法相比，interwhen 都实现了准确率与效率之间的帕累托最优工作点。代码见 https://github.com/microsoft/interwhen。

1 引言

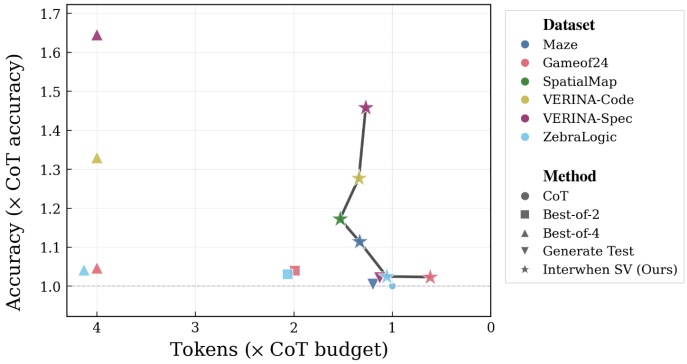
当前人工智能的发展势头显著提升了语言模型（LM）在推理任务上的准确率。然而，在法律、医疗保健或物理世界中的行动等许多高风险现实场景中，语言模型给出的每个答案都必须正确且可信；仅仅具有较高的正确答案生成概率并不足够。因此，验证 语言模型的答案成为一项重要目标。以往的研究探索了通过自一致性等方法进行自我验证 [Wang 等人， 2023], 以及通过现有领域专用验证器进行外部验证 [Cobbe 等人, 2021b]。在典型实现中，会独立生成多个答案轨迹，并选取通过验证器检验的轨迹 [Kambhampati 等人, 2024]。

然而，这些技术侧重于验证最终答案。考虑到近期测试时扩展的发展趋势，最先进的模型在给出最终答案前会输出数千个中间推理词元和/或进行数十次工具调用，因此这种方法效率低下，尤其是因为它无法及时发现早期错误。此外，在模型的中间操作会与外部环境交互的场景中（例如编辑文件或在物理世界中完成任务），在错误不断累积之前对模型进行引导至关重要。更一般地说，问题可验证的层级各不相同。有些解答（如代码）最适合通过最终答案来验证，而另一些问题（如撰写报告）则最好通过生成结果所经历的步骤来验证（因为验证一份长篇报告会带来相当大的认知负荷）。

*同等贡献。
†共同指导。联系邮箱：nagarajn@microsoft.com、amshar@microsoft.com。



(a) interwhen 与分支与验证方法的比较，例如思维树。



(b)

图1：大语言模型过程模约：干预部分输出轨迹，引导模型获得更准确和/或更高效的解答。a) 与思维树等现有方法相比 [Yao 等人， 2023]这类方法将问题拆分为多个步骤并执行多条相互独立的轨迹，interwhen 验证并引导一条单独的推理轨迹，从而提升效率，并让语言模型完全掌控执行过程。b) 使用 Qwen3-30B-A3B-Thinking-2507 获得的结果。图中仅展示各数据集上处于准确率—效率帕累托前沿的方法（完整图表见附录 A.5）。纵轴表示相对于思维链的准确率，横轴表示相对于思维链的输出词元数（坐标轴反转，因此理想点位于右上方）。interwhen 在每个数据集上都处于帕累托前沿，在词元预算相当的情况下，准确率显著高于标准思维链。

因此，要部署推理智能体，中间验证和事后验证都很重要。

因此，我们提出一种新的测试时范式，通过生成经验证的答案，重点验证（并纠正）通往解答的步骤，而不仅仅是最终答案。现有方法会验证部分步骤，并在每一步选择有效步骤 [Yao 等人， 2023] 它们将任务执行过程从外部拆解为若干步骤，从而人为地限制了模型的问题求解策略。相反，为了有效利用推理模型在问题求解能力方面的进展，我们采用以下设计要求：**1)**模型应完全掌控问题的执行过程；不应从外部将问题求解拆解为若干步骤。**2)** 与其在多个相互独立的步骤生成结果之间进行选择（这可能效率低下），不如在测试时向模型提供逐步反馈，并通过验证器反馈引导模型的推理。重要的是，如果没有错误，模型的输出文本应与未经任何测试时验证时完全相同。我们将这一范式称为*LLM-Process-Modulo* （见图1），以区别于 LLM-Modulo 范式 [Kambhampati 等人， 2024], 我们将后者称为大语言模型解答模约。

这引出了两个研究问题：1）给定来自语言模型的推理令牌流和一组可验证属性，如何根据这些属性验证部分输出轨迹；2）如何避免多轮中间验证导致的显著延迟开销。针对第一个问题，我们的核心贡献是一种基于 分叉的方法，避开了从推理轨迹中提取并形式化可验证状态（对应每个属性）时构建解析器这一容易出错的过程 [Wu 等人，2022]。另一种方案是提示模型，使其在求解任务时以所需格式输出中间可验证状态；然而，这需要大量的提示调优，而且可能并不可靠。相反，我们的方法会定期从模型的执行过程中分叉出来，并添加一条简短指令，要求模型以所需格式输出可验证属性。由于无需更改输入提示词或任务规范，我们获得了一种通用方法，只要能够定义可验证属性，就可以轻松迁移到不同域中，从代码到文本报告均可。

分叉策略还有助于通过启用异步 验证来回答第二个问题。模型会定期（例如，在出现 \n k 次之后）在单独的异步分叉执行中输出可验证属性，随后由验证器进行检查。只有当验证器发现错误时，主模型的执行才会中断，从而避免验证不必要地增加模型的推理延迟（见图1）。中断后，模型的输出上下文会回滚到错误发生的位置，并插入验证器的反馈。随后，模型会从该位置重新开始生成

，并以验证器的反馈为条件，这可能有助于模型修正错误。如果模型未能纠正错误，那么下次调用验证器时，验证器会再次标记该错误，从而确保最终的推理轨迹始终与可验证属性一致。

这两个想法——分叉式属性引出和异步验证——共同构成了一个通用框架 interwhen，它能够通过反馈引导模型的输出。给定一组可验证属性及其对应的验证器，interwhen 通过三个函数对引导问题进行抽象：提取、验证和干预——利用反馈更新轨迹。作为一种纯函数式的测试时方法，interwhen 可以轻松地将现有验证器连接到任意推理模型，并确保其符合给定的一组属性。由于将验证器视为黑盒，interwhen 支持各种验证器，包括符号式验证器和基于大语言模型的验证器。值得注意的是，这三个函数具有足够的通用性，可以实现任意测试时验证方法；我们还展示了如何使用 interwhen 实现思维树 [Yao 等人， 2023]等方法。在缺少外部验证器的情况下，这三个函数也能为提前停止 问题提供一种解法 [Mao等，2025，Wang 等， 2025]——避免推理模型过度思考——其中，验证函数会检查引出的中间解是否与先前的 k 解答一致。

我们在五个数据集上评估 interwhen，涵盖算术推理（Game of 24 [nlile， 2025]）、逻辑推理（ZebraLogic [Lin 等人， 2025]）、空间推理（迷宫、SpatialEval [Wang 等人， 2024]）以及代码生成（Verina [Ye 等人， 2025]）。每个数据集中的可验证属性由领域决定。例如，在迷宫中，模型必须遵循二维环境的结构；在 Verina 中，模型必须生成可在 Lean 中验证的代码。在全部五个数据集上，我们发现 interwhen 能够根据验证器的反馈引导模型。与标准思维链执行相比，它最多可将准确率提高 15 个百分点，同时消耗少于 $1.5\times$ 词元。对于每个数据集，与 best-of-n 和思维树 [Yao 等人， 2023] 等现有测试时验证基线方法相比，interwhen 解答均为帕累托最优。具体而言，当中间可验证属性与最终答案的正确性不同时（如迷宫和 SpatialMap），interwhen 最为有效，相较 Best-of-4 等消耗近 $3\times$ 更多词元的事后验证方法，准确率显著提升。最后，在没有外部验证器时，interwhen 也可用于提前停止问题，在不降低准确率的情况下最多实现 30% 令牌减少，显著优于 EAT [Wang 等人，2025] 和 DEER[Yang 等人， 2025] 等最先进方法。

2 相关工作

我们的工作横跨三个研究领域：语言模型输出的验证、测试时扩展和提前停止。

语言模型输出的验证。 在数学、规划和其他域中，测试时验证的主流范式是生成候选解，然后选择其中最佳者，或使用外部验证器 [Cobbe 等人，2021b、Kambhampati 等人， 2024] ，或使用同一个语言模型提供反馈 [Madaan 等人， 2023]。基于部分步骤进行验证和引导的研究，通常利用外部施加的结构将问题分解为多个步骤，例如要求语言模型在多步演绎任务中进行一次推导，或在多步数学问题中完成一步 [Creswell 等人，2022、Yao 等人， 2023]。其他研究则使用带有语言模型提示词的自我验证来验证并改进答案 [Chang 等人，2025、Stechly 等人， 2024]。与这些研究相比，我们在没有任何外部施加、用于控制执行的步骤结构的情况下进行部分步骤验证。相反，我们通过轻量级提示词引导模型输出中间状态。关于过程奖励模型的研究已经十分丰富 [Lightman 等人， 2024] ，但这与我们的工作相互独立。现有的过程奖励模型可以在 interwhen 框架中用作验证器；然而，我们探索的新问题是如何利用此类反馈，在测试时引导推理模型。已有初步研究尝试通过文本反馈在测试时引导推理模型 [Prabhakar 等人， 2025], ，但其与部分输出验证之间的联系尚未明确。

测试时扩展。 测试时扩展是一种新兴范式，它根据任务的复杂程度调整推理时的计算量，以提升语言模型的推理能力 [Snell 等人，2025，Balachandran 等人，2025，Kang 等人， 2025]。一种常用且通常被视为强基线的技术是 best-of-N采样：并行抽取 N 条轨迹 ，再由外部验证器 [Kang 等人， 2025] （例如评判大语言模型）或在允许的情形下采用简单启发式方法（例如多数票）评定并选出最佳最终答案。

这种策略的一种变体是依次 抽取样本，并根据先前答案的反馈，在*LLM-Modulo* 范式下进行 [Kambhampati 等人，2024]。

在分析中间轨迹的技术中，有些技术会鼓励探索 [Muennighoff等人， 2025] 通过启发式方法（例如插入“等等”以继续思考），从而借助额外计算资源解决复杂问题。另一类测试时扩展方法依赖经典的基于树的搜索技术，并使用语言模型引导探索，例如思维树 [Yao 等人， 2023]以及受蒙特卡洛树搜索启发的搜索 [Dainese等，2024年，Wang和Yang，2025，Huang等，2025a，Antoniades等人， 2025]。在这项工作中，我们提出了一种新的测试时扩展维度：通过单次展开，并借助验证器引导推理轨迹。

提前停止。 测试时扩展通过为更难的问题分配更多计算资源来提升性能，而提前停止追求相反的目标：通过缩减不必要的推理来提高效率。测试时扩展可能导致过度思考，即模型在已无显著改进后仍继续生成 [Ghosal等人， 2025], 为此，人们已提出多种缓解策略。

早期方法使用置信度信号来决定何时停止生成。基于熵的方法 [Sharma 和 Chopra， 2025] 将词元对数概率计算出的序列级香农熵作为置信度的替代指标；EAT [Wang 等人， 2025] 则在 </think> 之后计算熵，以触发提前退出；DEER [Yang 等人， 2025] 则监控中间推理块的置信度。这些方法依赖人工设定的阈值，而这些阈值难以解释，也难以在不同任务间校准。其他研究会在置信度较高时抑制反思触发词元（例如“等等”“或者”） [Huang等，2025b], 通过判别器和多臂老虎机将提前停止建模为一个控制问题 [Sun 等人，2025], 或者根据注意力模式，剪除对特殊思考结束词元贡献较低的推理块 [Choi 等人，2025]。Dynasaur [Fu 等人， 2024] 通过在每个推理片段之后执行多次推演，并依据推演答案的稳定性提前退出，来刻画答案稳定。

与我们的工作最相关的是 ES-CoT [Mao 等人 2025], 提出的方法。该方法会在每个推理块之后采样答案，并检查连续答案是否趋于稳定。尽管 ES-CoT 表明答案稳定性是强有力的停止信号，但它仅依赖采样答案之间的表层一致性，并不检查这些答案是否正确。我们的方法在此基础上更进一步：不仅在中间推理节点进行分叉以检查答案稳定性，还验证 候选答案，并将验证信号反馈到推理过程中，从而使模型能够自我纠正。

更广泛地说，interwhen 框架统一了测试时扩展与提前停止这两个不同方向：当中间验证发现不稳定或答案错误时，我们可以通过持续推理分配额外的计算资源。具体而言，在没有外部验证器的情况下，同一框架也可用于实现一种基于中间解稳定性的提前停止方法。

3 LLM-Process-Modulo

正如引言中所述，提高推理能力的多数技术有两个着力点：（1）扩展测试时计算；（2）关注解答的准确率，这通常意味着关注最终答案的正确性。当前用于提升推理能力的技术（见第 2 节）大体可分为两类，划分依据是它们是否以及如何引导推理轨迹，或更广义地说，如何探索解空间。

(1) 标准的 best-of-N采样方法不会引导推理过程。这些技术侧重于最终答案，效率低下，即使是简单问题也往往需要数万个词元（以及数次工具调用）。

(2) 诸如束搜索之类的基于树的搜索技术会利用语言模型引导搜索过程。与(1)相比，这些技术会明确地将解答生成分解为多个中间步骤，然后在树中的每个节点上利用语言模型从若干个允许的操作中选出最佳操作。虽然它们会干预解答的探索过程，并非纯粹专注于最终答案，但也可能受到限制——它们受限于底层搜索算法的启发式方法和选择，而且通常需要手工编写问题特定提示来引导搜索（例如，束搜索中的价值提示 [Yao 等人，2023]）。

为了解决上述两类技术面临的主要批评，我们提出了一个新的方向：干预并引导部分轨迹，我们将其称为“LLM-Process-Modulo”。其核心思想很简单：在推理过程展开时介入其中，使用外部验证机制（若可用）或自我验证（否则），从而确保推理过程具备某种形式的有效性或正确性。我们的提议类似于 LLM-Modulo 框架 [Kambhampati 等人，2024]，后者在最终答案上采用迭代式生成-测试范式；但此处的验证（即验证）发生在解答生成过程中。

3.1 问题规范

令 M_θ 表示一个由 θ 参数化的语言模型（LM）。M_θ 接收描述待解决问题的文本提示词 x 作为输入，并输出一串词元 S_i，其中

$$S_i = [t_1 t_2 ... t_i]$$

并且 t_i 是 M_θ 输出的第 i^{th} 个词元。
我们假设可以访问一组与任务相关的可验证属性 $\mathcal{P}$，从流 S_i 中提取的中间输出必须满足这些属性。每个属性 $p \in \mathcal{P}$ 都是针对某个中间输出的谓词（例如，在迷宫任务中，“到目前为止的所有移动步骤都位于有效的网格单元格上”），并具有一个真值，即其取值为 1（满足）或 0（违反）。

每个属性都关联一个理想验证器 V^*，其定义为一个函数，接收一段文本 T 以及与任务相关的属性 $\mathcal{P}$ 作为输入，并针对每个属性返回一个二元判断。此外，我们假设验证器会为每个未通过的属性提供文本反馈。形式上，我们可以将验证器定义如下：

$$V^*(T, \mathcal{P}) \longrightarrow \big(\mathbf{b}, \mathbf{f}\big), \qquad \mathbf{b} = (b_1, \ldots, b_m), \quad \mathbf{f} = (f_1, \ldots, f_m),$$

其中 $b_k \in \{0, 1\}$ 表示属性 P_k 是否得到满足，f_k 是当 $b_k = 0$ 时用于描述违规情况的自然语言反馈字符串（否则为空），而 m 是 $\mathcal{P}$ 的基数。在实践中，我们需要构建一个尽可能接近 V^* 的验证器 V。验证器可以是基于规则的、工具辅助的或基于大语言模型的，具体取决于领域和特定属性。需要注意的是，我们假设属性与验证决策是确定性的；概率验证则留待未来工作。

给定问题 x、模型 M_θ 和属性 $\mathcal{P}$，我们的目标是生成一个既正确（满足任务目标）又过程有效（$\mathcal{P}$ 中的所有属性在整个生成过程中均成立）的最终答案。在实践中，鉴于许多部署场景固有的计算资源约束，效率也是需要优化的目标。

3.2 LLM-Process-Modulo 的组成部分

给定理想验证器 V^*，求解上述问题的一种朴素方法是在每次词元生成之后（即对每个 S_i）运行验证器，并在每次失败时要求模型根据反馈重新生成。然而在实践中，构建验证器是最重要的子任务之一。此外，对于实用的解答而言，保持执行效率也同样重要。因此，求解该问题涉及以下组成部分：（1）**何时干预：** 推理模型会输出一串难以验证的词元，因此我们需要一种方法来识别推理过程中可验证的“状态”；（2）**如何验证**：在某些场景中我们可能需要形式化验证器，而在其他场景中则需要基于大语言模型的验证器。选择取决于具体领域和当前任务。（3）**如何干预：** 如果某个中间状态被验证器拒绝，我们应如何干预或引导 推理模型。这三个组成部分定义了 LLM-Process-Modulo 范式，我们将在下文加以描述。

何时进行干预。 要理解推理语言模型输出的连续“思考”词元往往颇具挑战，更不用说验证了。因此，我们需要一种系统的方法来 eliciting 推理过程中的可验证部分。首先面临的挑战是，如何在语言模型中引出对过程的验证。在数学等域中，我们可以将每个推导或公式视为一个“步骤”，而验证过程就是验证

给定前置条件后各个步骤的正确性。在这种情况下，只需解析轨迹中的步骤并对其进行验证或许就足够了。

但对于涉及逻辑和规划的一般推理问题，可能很难识别 可验证的中间步骤。这是因为在许多情况下，推理过程可能十分复杂，可能涉及提出假设、遇到矛盾时回溯等。因此，我们的目标不是识别可验证状态，而是生成 验证器检查某项性质所需的任意输入。也就是说，我们从推理轨迹中分叉出来，并指示模型输出性质检查验证器所需的输入。例如，图1展示了一个在二维迷宫中沿路径前进的问题。可验证状态是迷宫中的移动步骤，但它们可能与一般文本推理交错出现，因此难以解析。在下一节中，我们将展示如何通过从推理轨迹中分叉并指示模型以所需格式输出迄今为止的移动步骤来实现这种引导。

分叉缓解了识别或解析待验证确切步骤这一难题。它可以在推理轨迹中的任意时刻启动，例如在出现 k 次'\n'之后，或在任何其他周期性标记之后。尽管如此，更准确地识别应启动验证的词元（例如依据模型不确定性或概念识别）可能有助于提升效率，这留待未来研究。

如何验证。 我们的目标是验证部分输出轨迹。在规划等领域，可以使用同一个最终答案验证器来验证部分轨迹中的步骤。在其他领域，我们可能需要开发新的验证器。在许多推理场景中，验证器可能比求解问题更容易实现（例如，用于检查游戏对局有效性的基于规则的检查器）。例如，对于图1中的迷宫问题，我们通过在 Python 中将迷宫表示为二维数组，并在其上重放运算，来实现一个简单的验证器。在规划、逻辑和代码领域，我们还可以利用形式化验证工具，并对其进行扩展，使其能够适应部分轨迹中的不完整信息。最后，对于一般推理问题（例如文档生成、反事实分析），我们可以使用基于大语言模型的验证器。

在这项工作中，我们关注已有验证器的领域。为一般推理任务构建验证器是未来一项重要挑战 [Zhang 等人，2024]。

如何进行干预。 当验证器提供的反馈指出部分推理轨迹中存在错误时，我们需要确定如何干预推理过程。可以有多种选择：1）直接回退模型迄今为止生成的解答；2）提供验证器（或更一般地说，工具乃至人类专家）的文本反馈，然后继续推理过程；3）提供验证器给出的二元正确/错误反馈，等等。更激进的做法是将反馈作为用户消息呈现，假设模型可能比起自身的内部词元，更倾向于遵循用户指令。一般而言，恰当的干预方式取决于推理模型的能力、验证器的保真度以及当前任务的复杂度。例如，对于图1中的迷宫问题，我们可以将验证器反馈作为模型推理的一部分，通过在推理轨迹中用“等等”之类的反思词元引出反馈；也可以通过添加 $< /think >< user >$ 序列并在用户标签内提供反馈，以用户身份进行干预。

虽然本文无法回答所有问题，但我们提出了一个可泛化框架，以便在下一节中对这些问题进行研究。我们还提出了一种候选方法，相较于现有技术有显著改进。

4 interwhen：一个可泛化框架

人们已经提出了许多测试时验证和扩展技术。然而，对于给定任务，要全面评估和比较这些技术在准确率与效率之间的权衡并不容易。因此，我们构建了一个名为 interwhen 的通用测试时验证框架，用于探索上述研究问题，同时也为实现和比较现有的测试时扩展策略提供了一种简便的方法。

4.1 实现 LLM-Process-Modulo 的组成部分

第 3.2 节介绍的三个组成部分启发了三种基本运算，用于构建任何基于部分轨迹运行的测试时验证方法：1）从部分语言模型输出中**提取**可验证步骤（何时进行干预）；2）**验证**候选可验证步骤（如何验证）；3）通过提供反馈来改进某个候选步骤，或从候选步骤中选择一个，以此进行**干预**（如何干预）。

1. **提取**。给定部分输出流 S_i，extract 会返回 $J_i \in \{0,1\}$，用于指示是否应在输出流的此处通过分叉提取中间状态。对于程序化解法，即语言模型按顺序列出解答步骤的情况（例如简单的小学数学问题 [Cobbe 等人，2021a]），使用换行符或其他词元级分隔符，便足以确定输出流中进行分叉的位置。对于更复杂的问题，可以选择每隔 k 次出现分隔符（如“\n”）时进行分叉，或在反思词元之前进行分叉。这一步可以形式化表示如下：

$$J_i = \mathsf{extract}(S_i),$$

2. **验证**。如果 $J_i = 1$，我们需要在模型推理轨迹的此阶段获取中间可验证属性。我们通过在此处分叉 输出流，并通过一条简短指令以所需格式生成中间属性来实现这一点。此操作可定义为：

$$Z_i = \mathsf{fork\_and\_generate}(S_i)$$

其中 Z_i 是包含中间属性的文本。下一节将更详细地介绍我们的分叉流程。给定该 Z_i和相关属性 $\mathcal{P}$，我们使用验证器 V进行验证。验证器以中间输出 Z_i 和与任务相关的属性 $\mathcal{P}$为输入，并针对每项属性返回二元判断及相关文本反馈。形式上，我们可以将验证器表示为：

$$V(Z, \mathcal{P}) \longrightarrow \big(\mathbf{b}, \mathbf{f}\big), \qquad \mathbf{b} = (b_1, \ldots, b_m), \quad \mathbf{f} = (f_1, \ldots, f_m),$$

其中，$b_k \in \{0,1\}$ 表示属性 P_k 是否得到满足；f_k 是在 $b_k = 0$ 时描述违规情况的自然语言反馈字符串（否则为空）；m 是 $\mathcal{P}$的基数。

3. **干预**。如果验证器判定中间输出不满足某项属性，则处理所提供的反馈并更新输出流 S_i：

$$S_i^{updated} = \mathsf{intervene}(S_i, \mathbf{b}, \mathbf{f}).$$

干预可以要求模型继续生成，也可以决定停止生成，还可以选择将文本反馈添加到轨迹中。

可以使用上述运算实现不同的测试时扩展算法。例如，best-of-N 是一种特定情况：唯一需要验证的属性是最终答案；提取只在终点调用一次；干预则是停止生成，并在循环中重复这一过程。思维树 [Yao 等人，2023] 可以通过在外部强制模型遵循指定步骤来实现，然后对每一步的提取—验证—干预过程执行束搜索；其中，对于验证器判定排名靠前的步骤，干预步骤仅执行恒等操作，而对所有其他步骤则停止生成。

4.2 提议的算法：单轨迹引导

现在，我们介绍一种基于验证的算法，它结合了上述定义的三个函数。该算法不探索多条轨迹，而是维护一条输出轨迹。对于每个识别出的中间状态，它都会应用验证器；如果验证通过，就允许继续生成。否则，它会将文本反馈添加到模型的输出轨迹中，然后继续生成。其直觉是，模型应根据针对先前状态的文本反馈，改变其解题轨迹。图 2 展示了可视化示意，算法 1 则列出了具体步骤。

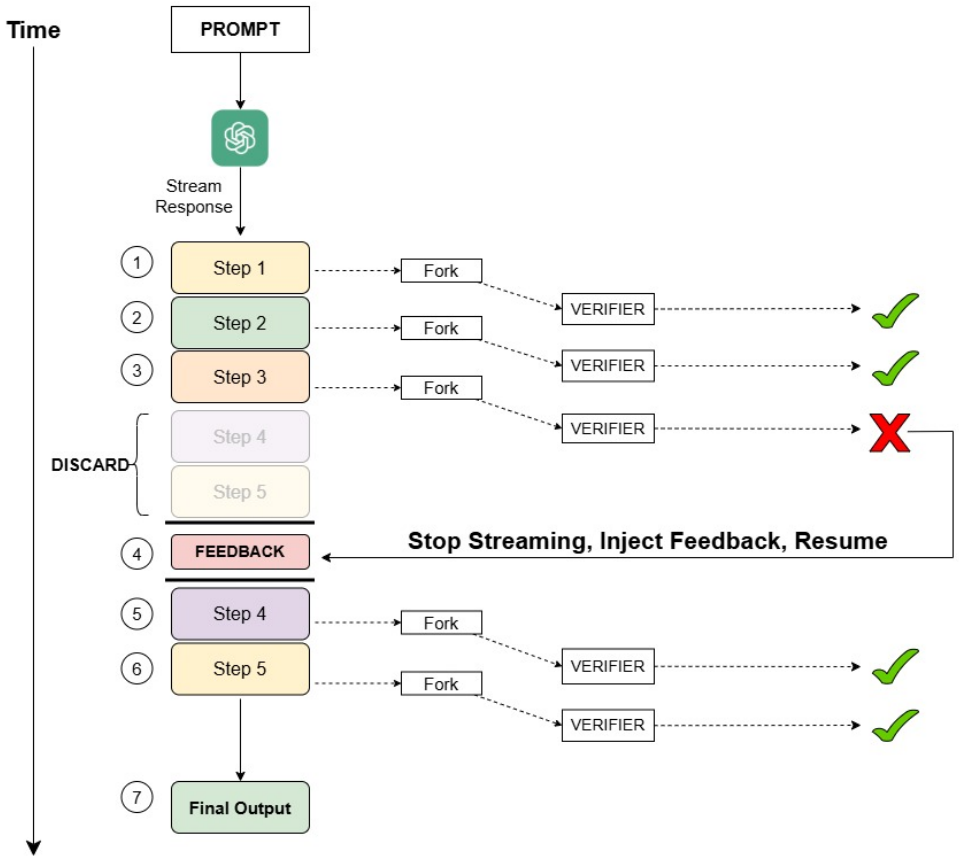


图 2：interwhen 流程图。为验证某项属性，interwhen 会分叉出一条独立轨迹，并指示模型输出可验证的属性。如果任何属性未能通过验证，模型的输出就会回滚，并在错误发生的确切位置插入反馈，从而让模型纠正错误。

在这一策略中，每经过 k 个段落，我们就通过复制提示词及迄今生成的文本来分叉生成过程。从这个分叉状态出发，我们会附加一个特殊标记（例如 </think>），接着添加一条指令，提示模型以结构化格式呈现其输出（例如：“让我按这种格式给出输出”）。根据目标可验证属性，这一输出可以包含中间推理步骤，甚至可以是模型在响应流特定阶段对答案的猜测。模型按照指定格式提供输出 Z 后，系统会解析该输出，并由 interwhen 验证器进行验证。如果验证器确认中间输出满足可验证属性，则无需干预，主生成过程会不间断地继续。由于验证异步运行（图1），主要的流式处理过程能够保持高效。不过，如果验证器检测到错误并生成反馈，框架就会丢弃分叉点之后生成的所有词元，并从分叉点恢复生成，同时纳入验证器的反馈（纳入反馈的方式是将其附加到推理轨迹中，可以采用模型自身的口吻，也可以作为用户消息）。为防止无限重试，我们引入了一个名为 maxretries 的参数（设为 5），用于限制反馈驱动的修正次数。如果超过这一限制，系统就会输出“无解”，确保不会返回错误答案。相对于验证器而言，这一设计提高了框架的可靠性，因为系统只会返回满足验证器要求的解答。

顺序（状态）验证器的优点：对于给定的一组验证器，我们的方法具有三项关键优势，即**（1）推理可靠性**，因为我们的方法能够保证中间步骤和最终输出符合给定验证器的要求。**（2）灵活的模型驱动推理**。不同于

算法 1 单轨迹引导（INTERWHEN）

输入： 大语言模型 M_θ，提示词 x，属性 $\mathcal{P}=\{P_1,\ldots,P_m\}$，验证器 V，分叉间隔 k，最大重试次数 R
输出： 已验证的输出或无解 –
$S \leftarrow [\,]$；$r \leftarrow 0$；$c \leftarrow 0$ ▷ 流、重试数量、分隔符数量
当 M_θ 将词元生成到 S 时，条件为 x **执行**
  将下一个词元 t追加到 S
  $J \leftarrow \text{EXTRACT}(S)$
  如果 $J == 1$ **，则**
    $S_{\text{fork}} \leftarrow S$ ▷ 保存分叉点，异步
    $S' \leftarrow S_{\text{fork}} +$ </think> + 格式指令 ▷ 侧流提示，异步
    $Z \leftarrow \text{GENERATE}(M_\theta,\ p,\ S')$ **直至完成** ▷相对于主流异步
    $(\mathbf{b}, \mathbf{f}) \leftarrow V(Z, \mathcal{P})$ ▷ $b_j \in \{0,1\}$, $f_j = P_j$的反馈
    如果 $\forall j:\ b_j = 1$ **则** ▷所有属性均已满足
      如果– IS COMPLETE（V）**则** ▷验证器是完备的验证器
        停止主流
        $S \leftarrow S_{\text{fork}} +$ 成功消息 + </think>
        返回 GENERATE（$M_\theta,\ p,\ S$） ▷ 强制给出最终答案
      else
        跳转到 4
      如果结束
    else
      $r \leftarrow r + 1$
      如果 $r > R$ **那么**
        返回无解 –
      如果结束
      停止主流
      $S \leftarrow S_{\text{fork}} + \left(\sum_{j:\, b_j=0} f_j\right)$ ▷ 回滚生成，并在分叉处追加违规反馈
      跳转到 4 ▷ 从更新后的 S恢复生成
    终点条件
  结束 if
结束 while
返回 S

会从外部限制推理过程的经典搜索启发式方法，我们的分叉方法能够让推理模型保有搜索的自主权。**（3）最少的提示工程**。我们无需通过提示工程引导推理模型生成中间状态，只需在分叉过程中使用简单提示词来引出中间可验证属性。

4.3 提前停止的应用

为了展示这三个功能的通用性，我们提出了一种简单的提前停止策略：k-***StableAnswer***，并使用 interwhen 实现；在没有外部验证器可用时，该策略也适用。思考模型在推理时，会自然而然地在推理轨迹中生成候选最终解；作为推理过程的一部分，模型往往会多次回顾并重述其最终解。我们的方法利用了这一行为，且无需分叉。具体而言，extract 使用正则表达式模式，在候选最终解出现时实时从推理轨迹中识别并提取这些解。verify 会将每个新提取的解与先前提取的解进行比较，检查同一解是否已连续稳定存在 k 个推理步骤。一旦满足这一稳定性准则，intervene 就会注入一个 </think> token，终止推理并提示模型给出最终答案。其直觉是，如果最终解已在推理轨迹中趋于稳定，继续审 deliberation 不太可能改变它。这一策略减少了不必要的推理，同时保留了模型最终预测的准确率。实际应用中，我们发现 k=2 在各种任务上都表现良好。

表 1：interwhen 中使用的数据集、任务特定可验证属性、中间表示（Z）、反馈和验证器概览。

数据集	问题	可验证属性 (P)	中间输出（Z）	验证器 (V)	验证器反馈
MAZE	给定一个网格迷宫，其中有起点（S）和终点（E），例如："需要多少次右转才能到达从 S 到 E？"	(1) 每个提议的移动都是有效转移（不会撞墙或越界）。(2) 转弯分类（左转-/右转）和转向次数是正确的。	当前位置、提议的移动方向、更新后的向左／向右转向次数/右转次数 at 每一步。	基于规则的（Python 矩阵）	表示是否一步导致一个无效位置 or 是否 the turn 方向/数量 is 不正确。
SPATIALMAP	给定物体位置，例如："什么位于西北 of the 公园？"或"有多少个物体位于学校的东侧学校的？"	生成的空间陈述在逻辑上与中的约束一致问题描述。	中间断言以及关于.....的推理空间关系在物体之间。	形式化 (Z3 SMT 求解器)	表示生成的主张与之矛盾规定了问题中的空间约束题设中的约束-lem.
GAMEOF24	给定四个整数，构造一个表达式使用 +、−、×、/ 来得到 24，例如："用 1, 5、5、5 算出 24。"	(1) 该表达式使用恰好四个输入数字，各自恰好一次。(2) 该表达式的值恰好为 24。	目前推导出的候选算术表达式基于迄今为止的主推理轨迹。	基于规则的的（Python）	表示不正确算术结果，使用多余/缺失数字。
ZEBRALOGIC	给定房屋、特征，以及自然语言线索，例如，"养猫的人住在蓝色房屋旁边。" De-得出唯一的分配方案——将所有特征分配给房屋。	当前属性分配至房屋分配满足所有谜题约束。	JSON 表示 of 当前分配属性给房屋。	形式化 (Z3 SMT 求解器)	表示当前赋值与谜题中的约束相矛盾。
VERINA	给定 a 问题描述，生成以下任一项：(a) 函数-tion body 给定 a 函数签名 and 前置条件/-后置条件，或 (b) 前置条件/后置条件给定辅助函数-项和参考实现。	生成的 Lean4 代码或规约能够成功编译。	Lean4 代码或规约-对应到目前为止的推理轨迹。	Lean4 编译器	编译错误消息以及基本 Lean4 语法指导。

5 实验设置

5.1 数据集与模型

我们在空间推理任务上评估 interwhen，包括 MAZE 和 SPATIALMAP （各有 1,500 个实例） [Wang 等人， 2024]; 通过 GAMEOF24 进行算术推理（1,362 个实例） [nlile， 2025], ; 通过 ZEBRALOGIC 进行逻辑推理（1,000 个实例） [Lin 等人， 2025] ; 以及使用 VERINA 进行代码和规约生成（189 个实例） [Ye 等人， 2025]。下面介绍数据集详情：

- **迷宫**：每个示例都提供一个包含起点和终点的网格；问题要求围绕路径进行推理，例如计算右转次数、总转弯次数，或起点相对于终点的位置。
- **SpatialMap**：示例会指定物体位置；问题会询问某个物体的相对位置、指定方向上的物体，或沿某个方向的物品数量。
- **24点游戏**：每个示例都提供四个整数，任务是使用每个数字恰好一次构造一个表达式，并通过 +， −， ×， / 得到 24。
- **ZebraLogic**：每个示例都包含一组按顺序编号的房屋，以及一组特征（例如颜色、

汽车型号、宠物），每个特征都有一个有限域值集合，还有一份以自然语言线索编码约束的列表，例如同屋共现、位置安排、相对顺序和相邻关系。任务是推导出满足每条线索的所有房屋的所有特征值的唯一分配。我们还报告了 ZebraLogic 的 X-Large 子集上的结果，该子集按照原论文的定义，包含数据集中最困难的问题。

- **Verina**：我们对两个子任务进行评估：代码生成（VERINACODE）和规约生成（VERINASPEC）。这两项任务都要求根据问题描述和补充细节生成正确的 Lean4 代码。

模型与解码。我们的主要实验使用两种不同的开源推理模型：Qwen3-30B-A3B-Thinking-2507[1] [Team, 2025a] 和 QwQ-32B[2] [Team, 2025b]。所有实验均使用 vLLM 运行。对于这两种 Qwen模型，我们将最大生成长度设为 32,768 个词元，温度设为 0.6，top-p 0.95，top-k 20。我们还使用另一个模型 Phi-4-Reasoning[3] [Abdin 等人， 2025], 进行了一些消融实验，其最大生成长度为 16,384 个词元，温度为 0.8，top-p0.95，top-k 50；这些设置遵循其官方文档中的推荐配置。表 1 展示了我们所用数据集的概览。

5.2 特定任务的可验证属性

对于每项任务，在特定片段后创建分支时，我们会指示模型以指定格式输出中间输出 Z，以便使用可验证属性 $\mathcal{P}$ 专用验证器 V 检查任务特定的属性是否成立，并在出现违规时提供纠正反馈。除 Verina 外，对于所有任务，我们都以模型自己的口吻提供反馈（即在 $< /think >$ 和 $< /think >$ 标签内）。对于 Verina，我们发现，将反馈作为用户消息传递时，模型对反馈的遵循程度更高。根据任务结构，我们的所有验证器均通过基于规则的检查或形式化求解器实现。

对于**24点游戏**，根据当前推理状态，我们会在分支中提示模型输出它根据主执行轨迹目前推导出的算术表达式。具体而言，侧流会将 </think> 我目前找到的表达式是 { 追加到当前思考轨迹，并流式生成一段简短的完成内容（∼20 个词元），以提取候选表达式。关键的可验证属性 包括：(i) 提取出的表达式是否恰好使用了四个输入数字，且每个数字仅使用一次；(ii) 表达式的计算结果是否为 24。这些检查由基于规则的验证器实现，该验证器通过程序计算表达式的值。如果表达式正确且完整，验证器便会触发提前停止，并让模型转而输出最终答案。如果表达式不正确，则会向思考轨迹注入第一人称反馈，指出具体错误（例如，算术结果错误、缺少或多出数字），并指示模型不要再次使用失败的表达式。接在自然的</think> 标签之后，第二个验证阶段会强制模型输出一个 \boxed{} 表达式，并对其进行相同的检查；如果检查失败，则会追加一个结构化的 [验证器反馈] 块，提示模型采用完全不同的方法。

对于**迷宫**导航，我们要求模型输出一系列步骤，其中包括迷宫中的当前位置、拟议移动，以及更新后的左转和右转次数。首要的可验证属性是拟议移动是否会在迷宫中产生有效转移。为验证这一点，我们会解析迷宫并将其存储为 Python 矩阵，然后检查每个拟议移动，确保其不会进入墙内或超出迷宫边界。另一个可验证属性是连续移动之间的方向变化是否对应正确的转弯分类（例如左转或右转），以及转向次数是否已正确更新。出现违规时，反馈会指出拟议步骤是否导致无效位置，或对转弯方向的判断是否有误。

对于 **SpatialMap** 推理，模型需要在生成最终答案之前，针对物体之间的空间关系提出中间断言并进行推理。此任务中的可验证属性 是生成的断言在逻辑上是否与问题中指定的约束一致。为进行检查，系统使用 z3 SMT 求解器对问题描述中的约束进行编码，并将每个生成的

[1] https://huggingface.co/Qwen/Qwen3-30B-A3B-Thinking-2507
[2] https://huggingface.co/Qwen/QwQ-32B
[3] https://huggingface.co/microsoft/Phi-4-reasoning

断言转换为相同的表示形式。随后，验证器会检查新断言是否与现有约束产生矛盾。如果检测到矛盾，反馈会指出生成的断言违反了问题中提供的约束。

对于 **ZebraLogic** 问题，在分叉指令中，模型被要求在推理过程中生成当前属性到房屋的分配的 JSON 表示。此处的可验证属性 是这些分配是否满足谜题中指定的约束。与 SpatialMap 验证器类似，这些约束使用 z3 进行编码，并检查所生成分配的逻辑一致性。如果分配违反任何约束，反馈会指出当前分配与问题中给出的条件相矛盾。

最后，对于 **Verina** （代码与规约生成），模型被要求生成与迄今推理轨迹相对应的 Lean4 代码。在此设定中，可验证属性 是所生成的代码在语法上是否有效并能成功编译。为验证该属性，所生成的代码会被提取并使用 Lean4 进行编译。如果编译失败，监控器会将编译错误消息作为反馈返回。由于很大一部分错误源于语法问题，反馈中还会包含基本的 Lean4 语法指导，以帮助模型修正所生成的代码。

5.3 基线与评估指标

5.3.1 基线方法

Best-of-K (评判器)：对于每个输入实例，我们从基线模型中采样 K 条相互独立的推理轨迹。一个辅助评判器模型（大语言模型评判器，与基线模型相同）会评估每个候选项，并为其分配一个表示其正确性的标签。我们选择评判器评分最高的第一个候选项作为最终预测。如果评判器未能识别出有效候选项，则回退到第一条生成轨迹所给出的答案。我们报告 $K \in \{2, 4\}$ 的结果，遵循 [Brown 等人， 2024]

Best-of-K：对于每个输入实例，我们从基线模型中采样 K 条相互独立的推理轨迹。如果 K 个候选项中任意一个与标准答案匹配，则该答案正确。这为完美选择机制所能达到的性能提供了上限。我们报告 $K \in \{2, 4\}$ 的结果。

生成-测试（评判器反馈）：我们采用先生成后验证循环：模型先生成完整解答，然后由评判器评估生成的解答，并就其有效性提供反馈。如果解答通过验证，流程便终止并返回答案。否则，模型会根据评判器的反馈生成修订后的解答，并重复这一过程，直到生成有效解答或耗尽生成预算，见 [Kambhampati 等人， 2024]。我们将迭代次数限制为四次。

思维树： 我们按照 Yao 等人 [2023] 的方法实现思维树方法。提议者提示词生成候选后继状态，而带有评分少样本示例的价值提示词则通过分类判断（确定、很可能、可能、不太可能或不可能）评估中间状态。这些分数通过束宽为 2 的束搜索引导搜索，从而最多产生四个活动分支。所有数据集上的搜索深度上限为 8。

5.3.2 评估指标

准确率。 准确率衡量模型生成的最终答案的正确性。对于非代码任务，正确性要求与标准答案精确匹配；而对于代码生成任务，则通过对生成的代码运行所提供的测试用例来确定。

%词元。 我们报告相对于思维链（CoT）基线所生成的词元数量，并将该基线的词元数量作为参考，设为 100%。对于大多数基线方法，这对应于推理轨迹和最终答案中生成的词元总数。对于使用验证器或辅助推理流的方法，我们将这些额外流中生成的词元计入总数量。具体而言，当使用基于验证器的分支时，我们会统计主推理流和侧流分叉中的词元。由于非大语言模型验证器的执行速度很快，我们不考虑其执行时间带来的计算开销。

由于我们的设定本质上是一个多目标优化问题，因此可以说，理想方法在准确率和词元数量方面应当是帕累托最优的；也就是说，对于给定的数据集，不存在其他方法能在准确率上严格优于理想方法，或使用严格少于理想方法的词元。

表 2：不同基准测试和模型的准确率与词元用量对比。CoT：思维链，Bo2：二选优，Bo2-c：二选优（评判器），Bo4：四选优，Bo4-c：四选优（评判器），GT-c：生成与测试（评判器），ToT：思维树，IW：interwhen（我们的方法）。词元用量以相对于思维链的百分比表示。对于每个准确率行，整体准确率最高的方法以<u>下划线</u>标出；在词元用量不超过 Bo2 的方法中，准确率**最高** 的方法以粗体标出。

		Qwen3-30B								QwQ-32B							
数据集	**指标**	CoT	Bo2	Bo2-c	Bo4	Bo4-c	GT-c	ToT	**IW**	CoT	Bo2	Bo2-c	Bo4	Bo4-c	GT-c	ToT	**IW**
Game24	准确率（%）↑	95.0	**98.8**	97.4	<u>99.5</u>	97.4	96.1	95.0	97.2	83.7	**93.9**	84.4	<u>97.7</u>	84.4	84.3	74.1	89.1
	词元百分比 ↓	100	199	199	400	400	116.3	936.5	61.6	100	196	196	389	389	120.7	812	93.7
Maze	准确率（%）↑	88.5	90.4	88.5	91.2	88.6	88.9	90.3	**<u>98.6</u>**	85.8	**91.5**	87.0	<u>94.7</u>	87.1	80.7	69.3	87.8
	词元百分比 ↓	100	201	201	399	399	119.7	746.2	133.1	100	199	199	401	401	107.4	727	132.8
SpatialMap	准确率（%）↑	74.7	79.1	75.1	81.6	74.9	73.9	75.6	**<u>87.5</u>**	74.9	78.7	74.8	81.4	74.7	63.2	67.1	**<u>85.0</u>**
	词元百分比 ↓	100	201	201	399	399	111.8	672.5	152.9	100	198	198	395	395	127.2	563.5	156.4
ZebraLogic	准确率（%）↑	95.5	**98.4**	93.6	<u>99.5</u>	93.8	94.4	17.2	97.9	76.1	**82.9**	77.0	<u>86.6</u>	77.9	77.6	16.2	77.9
	词元百分比 ↓	100	206	206	413	413	102.7	861.1	105.6	100	190	190	380	380	94.3	740.1	92.2
ZebraLogic X-Large	准确率（%）↑	79.2	**94.3**	70.3	<u>97.4</u>	70.3	72.9	1.2	89.6	19.3	24.5	17.2	<u>38.5</u>	20.8	18.8	0.5	**26.0**
	词元百分比 ↓	100	203	203	406	406	91.9	880.3	103.4	100	192	192	384	384	97.7	780.2	88.2
Verina-Code	准确率（%）↑	56.1	65.6	57.1	<u>74.6</u>	56.6	53.4	51.9	**71.6**	56.1	66.7	53.4	<u>79.9</u>	53.4	57.1	49.2	**75.7**
	词元百分比 ↓	100	202	202	400	400	111.4	1681.5	133.9	100	196	196	399	399	97.8	1203	114.1
Verina-Spec	准确率（%）↑	25.4	33.9	27.0	<u>41.8</u>	27.0	25.9	29.6	**37.0**	37.6	40.2	33.3	<u>48.7</u>	33.3	34.9	33.9	**44.4**
	词元百分比 ↓	100	203	203	400	400	112.8	1825.4	126.9	100	197	197	400	400	99.2	1049.7	187.1

更形式化地说，如果我们将每种方法表示为一个元组（a_i，t_i），那么当不存在其他方法 j 使得 $a_j \geq a_i$ 且 $t_j \leq t_i$时，该方法就是帕累托最优的，其中至少有一个不等式是严格的。

6 结果

6.1 interw hen 在准确率与效率的权衡中实现帕累托最优性 off

表2展示了 Qwen3-30B 和 QwQ-32B 在七个基准测试上的准确率和 %tokens。我们将 interwhen 与六种基线方法进行比较：标准思维链（CoT）、Best-of-2 及其评判器变体（Bo2、Bo2-c）、Best-of-4 及其评判器变体（Bo4、Bo4-c）、采用评判器反馈的生成与测试（GT-c），以及思维树（ToT）。令牌用量以 CoT（100%）为参照进行报告。我们还在图1b中直观展示了这些结果。

摘要。 在全部七项任务中，interwhen 始终能达到与最佳基线方法相当或更高的准确率，同时只需消耗其一小部分令牌预算。在结构化任务（MAZE、 SPATIALMAP、GAMEOF24）上，其准确率往往最高，成本仅为 $1\times$–$1.5\times$ CoT，与采样和搜索方法的 $4\times$–$9\times$ 相比，成本更低。基于评判器的选择（Bo2-c、Bo4-c）表现往往不如 CoT，这表明 LLM-as-judge 在方案排序方面并不可靠。思维树同样依赖语言模型进行验证，它在 MAZE 上略有提升，但在逻辑和验证任务上的准确率则大幅下降。有趣的是，采用神谕验证器的 best-of-n 是最强的基线方法：在七项任务中，有三项任务上 best-of-2 的准确率高于 interwhen。不过，interwhen 的效率显著更高，适用于可能需要关注延迟的部署场景：其相对于基线的 %tokens 为 62%–152%，而 best-of-2 为 200%。

interwhen 在每个数据集上均达到帕累托最优。 我们观察到，对于每个数据集，我们的方法都位于方法的帕累托前沿上，即对于给定的数据集，不存在在准确率和词元数量这<style id='5'>两个方面都严格优于或等于 interwhen 的方法。每个数据集的帕累托前沿图见附录 A.5。

准确率提升，效率与思维链相当。如表 2 所示，在所有基线方法中，只有生成-测试（GT-c）的词元开销与 i nterwhen 相当。在所有数据集和模型上，interwhen 的表现均优于生成-测试。例如，使用 Qwen3-30B 模型完成24点游戏时，interwhen 达到

97.2% 的准确率，高于 GT-c 的 96.1%；其词元用量仅为思维链的 61.6%，而 GT-c 所需词元用量则为思维链的 116.3%。在 SpatialMap 中，我们的方法准确率达到 87.5%，高于 GT-c 的 73.9%。即使 interwhen 需要略多的词元，例如在 Verina-Code 和 Verina-Spec 中，我们观察到其性能比 GT-c 提升了最高 18 个百分点，凸显了我们的方法仅需相对少量额外词元所带来的益处。在词元用量更高的方法中，best-of-n 方法的准确率最高。然而，即使增加了计算量，best-of-2 方法在 7 项任务中的 4 项上准确率仍低于 interwhen，这表明中间验证具有优势。

在更难的任务上提升幅度更大。基线思维链表现较差的难任务中，准确率提升最为显著，例如 Verina 任务和 ZebraLogic X-Large。在 VERINA-CODE 上，interwhen 在 Qwen3-30B 上接近 Bo4 的上限（71.6% 对比 Bo4 的 74.6%），在 QwQ-32B 上也接近 Bo4 的上限（75.7% 对比 Bo4 的 79.9%）；而其思维链词元用量仅为 Bo4 的 114%–134%，Bo4 的用量为 ∼400%。与思维链相比，这使 Qwen3-30B 的准确率提高了 15.5 个百分点（思维链为 56.1%），QwQ-32B 则提高了 19.6 个百分点。基于评判器的选择在此任务上始终表现不佳：Bo2-c（57.1%、53.4%）、Bo4-c（56.6%、53.4%）和 GT-c（53.4%、57.1%）的准确率都徘徊在思维链准确率附近或低于它，这表明 LLM-as-judge 难以可靠地评估代码正确性。思维树表现更差，在两个模型上的准确率都低于思维链（51.9%、49.2%）。在 VERINA-SPEC 上，interwhen 在 Qwen3-30B 上达到 37.0%（比思维链高 +11.6），在 QwQ-32B 上达到 44.4%（比思维链高 +6.8），词元成本分别为 127% 和 187%。虽然 Bo4 仍是准确率上限（41.8% 和 48.7%），但它需要 ∼400% 的词元。Bo2（33.9%、40.2%）提供了更高效的替代方案，但在两个模型上的准确率仍不及 StepVerifier。与 VERINA-CODE一样，所有基于评判器的方法都会使准确率低于思维链，这进一步表明，针对形式化验证任务的评判器选择校准不佳。
在 ZEBRALOGIC X-LARGE 上，interwhen 在 Qwen3-30B 上达到 **89.6%**，比思维链（79.2%）高出 10 个百分点，词元仅增加至 103%。同样，Bo2（94.3%）和 Bo4（97.4%）的准确率高于 interwhen，但分别需要两倍和四倍的词元；Bo4-c 的准确率则降至 70.3%，而思维树完全失效（1.2%）。请注意，虽然在 ZebraLogic X-Large 上准确率提高了 10 个百分点且词元没有显著增加，但在 ZebraLogic 上仅提高了 2 个百分点，这再次表明，我们的方法在较难任务上收益更大。

interwhen 有利于中间检查与最终检查不同的任务。除 Maze 和 SpatialMap 外，在所有其他任务中，所检查的中间属性与最终答案所检查的属性相同。例如，在 Maze 中，中间可验证属性包括转向分配是否正确、移动步骤是否会导致任何碰撞等，而最终答案只检查转向次数。在 SpatialMap 中，会根据问题的约束验证中间推理，而对于最终答案，我们只检查位置或物体数量（取决于具体问题）是否正确。在此类任务中，我们观察到使用 interwhen 能带来更大的性能提升，甚至优于 Best-of-4 等方法，这表明验证过程而不仅仅是最终答案确实有益。相比之下，在 24点游戏、Verina 和 ZebraLogic 等任务中，最终验证与对中间解所执行的验证相同。在这类任务中，我们观察到，best-of-n 等简单的基于采样的方法能够胜过 interwhen。

完整的**验证器使测试时扩展与提前停止得以结合**。在 GAMEOF24上，interwhen 在 Qwen3-30B 上达到 97.2%，而 CoT 令牌用量仅为**61.6%** ，是唯一一种相较于标准 CoT 同时提高准确率并减少令牌用量的方法。这一独特结果源于 GAMEOF24 验证器的特性：它是完备的。对于给定的候选算术表达式，验证器能够明确确认该表达式是否恰好使用全部四个输入数字并计算出 24。当模型在任何中间推理步骤提出有效的完整表达式时，验证器会立即确认其正确性，步骤级反馈循环则通过发出成功信号（例如，附加反馈说明表达式正确，促使模型结束推理）提前终止生成。这一结果表明，当存在完备验证器时，interwhen 能够同时提升准确率和效率。

6.2 消融实验

我们现在通过一种基于提示的替代方案，对 interwhen 中的分叉策略进行消融：在输入提示中加入指令，要求模型在推理轨迹中输出结构化中间解。我们还比较了 interwhen 对推理模型与非推理模型的效果。

元提示。 标准提示只提供任务描述，而元提示还会对输出轨迹施加结构约束，使中间步骤清晰标示并可供验证。形式上，给定一条非结构化输出轨迹，元提示旨在将其重塑为半结构化序列，其中中间状态与自由形式推理交替出现："$\ldots, z_i, \ldots, z_{i+1}, \ldots$"，其中$\ldots$表示任意文本。实践中，我们通过提供少样本示例来实现这一点，这些示例展示了如何按照指定格式表达中间状态。例如，对于二维迷宫任务（见附录 A.3），元提示只需指示模型以固定、可解析的格式输出每一步移动。实践中我们发现，这种策略会导致模型仅在推理接近末尾时才输出中间解，尤其是在 </think> 标签之后。

不过，由于 interwhen 作用于中间状态，一个自然的问题是：它的有效性是否依赖于特定的提示策略，还是说，即使在 </think> tag 之后输出部分答案，它仍能作为验证层实现泛化。为检验这一点，我们采用上述修改后的提示下生成的 Qwen3-30B-A3B 输出，并对提取出的中间输出应用 interwhen 的验证。如表 3 所示，interwhen 在三个基准测试上均带来了稳定的准确率提升：Maze（$95.46 \rightarrow 98.74$%）、Game24（$96.84 \rightarrow 96.99$%）和 SpatialMap（$83.33 \rightarrow 86.13$%）；同时仅产生适度的令牌开销增加（相较于元提示基线最多增加 $\sim$14%）。这些结果证实，interwhen 与提示无关：它可以与任何上游提示策略组合，作为额外的验证层，捕捉残余错误并进一步提升正确性。

表 3：消融实验：在采用元提示的 Qwen3-30B-A3B 输出上应用 interwhen。*% Tokens* 表示相对于元提示基线的词元用量。元推理提示见附录。

Task	**方法**	**准确率（%）** ↑	**% 词元** ↓
Maze	元提示	95.46	100.00
	+ interwhen	**98.74** (+3.28)	113.87
Game24	元提示	96.84	100.00
	+ interwhen	**96.99** (+0.15)	100.72
SpatialMap	元提示	83.33	100.00
	+ interwhen	**86.13** (+2.80)	103.87

推理与非推理模型。此前所有实验均使用推理模型进行，这些模型会在生成最终答案之前生成显式的思维轨迹。为考察我们的方法能否推广到此类模型之外，我们还在一个非推理模型上对其进行了评估。具体而言，我们在 ZebraLogic 基准测试上测试了 Qwen/Qwen3-30B-A3B-Instruct-2507，并将其性能与启用了推理功能的对应模型 Qwen/Qwen3-30B-A3B-Thinking-2507 进行了比较。我们可以从表 4 中清楚地看到，与思维链相比，interwhen 的准确率更高，而词元数量仅略有增加。

表 4：interwhen 与标准思维链（CoT）提示在 Qwen3-30B-A3B 的推理和非推理变体上的表现比较。

		Qwen3-30B-A3B-Instruct-2507		**Qwen3-30B-A3B-Thinking-2507**	
数据集	**指标**	CoT	IW	CoT	IW
ZebraLogic	准确率（%）	93.0	**97.4**	95.5	**97.9**
	词元百分比	100	104.9	100	105.6
ZebraLogic X-Large	准确率（%）	69.8	**90.6**	79.2	**89.6**
	词元百分比	100	103.1	100	103.4

模型	方法	Maze		SpatialMap		GameOf24		Verina-Code		Verina-Spec	
		准确率	减少百分比	准确率	减少百分比	准确率	减少百分比	准确率	%Red	Acc.	%Red
Qwen3-30B-A3B Thinking-2507	Base	88.53	0.0	74.67	0.0	95.01	0.0	56.08	0.0	25.40	0.0
	k-Stable（我们的方法）	88.53	**32.24**	75.00	**10.83**	95.45	**31.65**	56.61	**3.54**	25.40	**55.09**
	DEER	88.53	0.61	74.87	8.23	95.01	3.82	56.08	2.00	25.40	6.88
	EAT	88.53	0.00	74.60	0.41	95.01	0.00	56.08	0.00	25.40	1.70
QwQ-32B	Base	85.80	0.0	74.93	0.0	83.70	0.0	56.08	0.0	37.57	0.0
	k-Stable（我们的方法）	85.80	**17.45**	74.93	4.69	**84.51**	**19.35**	56.08	**29.42**	38.10	**24.33**
	DEER	85.53	3.64	75.00	**6.42**	83.77	1.37	58.20	14.31	37.57	6.46
	EAT	85.73	0.14	74.93	0.02	83.70	0.08	56.08	0.00	37.57	8.09
Phi-4 推理	Base	86.47	0.0	69.73	0.0	77.97	0.0	33.86	0.0	20.63	0.0
	k-Stable（我们的方法）	86.53	**4.87**	69.80	16.05	77.97	**9.12**	33.86	17.49	20.63	2.40
	DEER	87.07	2.76	70.13	**35.36**	77.97	1.2	34.92	**50.73**	22.22	**8.07**
	EAT	87.07	2.88	69.80	1.68	77.97	0.78	33.86	1.43	20.63	0.00

表 5：不同模型和数据集上的提前停止消融实验。我们报告了准确率，以及在相对于基线保持或提高准确率的同时实现的最大令牌减少比例（减少百分比）。每个模型–数据集组合中最佳的减少百分比以**粗体**显示。

7 应用 interwhen 与早停

在没有外部验证器的情况下，我们现在展示如何仍能利用 interwhen 的基础功能来提升推理模型的效率。该方法基于第 4.3 节所述的 k-稳定答案 方法。

为不同任务实现 提取**和**验证。值得注意的是， k-稳定答案方法无需在任何时候进行分叉；它完全在单条推理轨迹内运行，利用模型在延长思考过程中自然倾向于重新审视并重述最终解答这一特点。对于多项选择推理任务（Maze 和 SpatialMap），提取函数使用正则表达式模式检测由“答案是”等短语及其相关变体所标示的答案提议。随后，验证函数会检查提取出的选项是否连续 k 次保持不变。对于 Gameof24 数据集，提取函数会识别模型提出的方程，并要求输入中的每个数字在提取出的表达式中恰好出现一次。验证函数会测试同一方程是否连续出现 k 次。对于 VERINA 代码生成和规范生成任务，提取函数首先使用正则表达式模式（例如“代码是”等短语及其相关变体）找出响应中表明存在代码或规约的部分。由于在此情形下，简单的字符串匹配并不足够，我们使用一个小型语言模型（Qwen/Qwen2-1.5B-Instruct）实现验证函数，以判断当前提取的产物在语义上是否等同于前一个产物。对于规范生成，我们分别跟踪前置条件和后置条件的稳定性，使用两个独立的计数器，并且仅在两者连续 k 次保持不变时触发终止。在所有数据集中，一旦验证函数确认满足稳定性准则，干预函数便会注入 </think> 标记来终止推理，并提示模型输出最终答案。

基线方法。 我们将 k-*Stable Answer* 与另外两种近期的早停方法进行比较，这两种方法也使用 interwhen 实现。我们使用与第 5.1 节中所述相同的模型及其对应的解码参数。

EAT （Entropy After </think>） [Wang 等人 2025] 在每个推理块之后附加一个 </think> 词元（定义推理块的一种方式是每当生成 \n\n 时就停止），并计算下一个词元分布的熵。对该熵进行指数移动平均；当其低于预设阈值时，推理便会终止。

DEER （推理模型中的动态提前退出） [Yang 等人 2025] 通过提示模型“</think> 答案是。”，在每个推理步骤后估计答案置信度。如果置信度超过阈值，就会停止推理，并由模型生成最终答案。

结果。 表 5 在五个基准测试和三个模型上评估了三种早停方法： *k-Stable Answer* （本文方法）、DEER 和 EAT。对于每种方法，我们遍历其关键超参数，并选出在相较于基线思维链准确率不下降的前提下令词元减少（减少百分比）最大的设置： k 用于 *k-Stable Answer* 的稳定窗口、用于 DEER 的置信度阈值，以及λ 用于 EAT 的阈值。附录 A.4.1 列出了所评估的全部超参数取值。 *k-Stable Answer*始终能在保持或提高准确率的同时，大幅节省词元。在 Qwen3-30B 上，它在 MAZE减少了 **32.2%** ，在 GAMEOF24减少了 **31.7%** ，并在 VERINA-SPEC 减少了 **55.1%** 。在 QwQ-32B 上，它在 GAMEOF24 减少了 **19.4%**，在 VERINA-CODE减少了 **29.4%**，并在 VERINA-SPEC减少了 **24.3%**。重要的是，即使在减少幅度较小的设置中，例如 QwQ-32B 上的 SPATIALMAP （4.7%）或 Phi-4-Reasoning 上的VERINA-SPEC （2.4%）， *k-Stable* 也始终能实现实质性的减少，且不会损害准确率。

DEER 和 EAT 在某些设置下具有竞争力；例如，DEER 在 Phi-4-Reasoning 的 SPATIALMAP 上实现了显著减少（35.4%），在 Phi-4-Reasoning 的 VERINA-CODE 上也实现了显著减少（50.7%）。然而，这些方法的表现不够稳定：EAT 经常几乎无法节省令牌，而 DEER 偶尔会降低准确率（例如，在 QwQ-32B 上的 MAZE）。相比之下， *k-Stable Answer* 保证在我们评估的每种模型与数据集组合中都能减少令牌，因此是最可靠的早停策略。完整的超参数扫描结果见附录表 6–8。

8 结论与未来工作

我们提出了一种测试时验证框架，通过引导单条输出轨迹，同时提升准确率和效率。该框架支持内部验证和外部验证，并能轻松接入验证器，以便在测试时强制执行所需属性。更广泛地说，interwhen 框架提供了一个试验平台，可将验证作为测试时扩展的新维度进行探索。

我们已经从效率的角度展示了如何在 interwhen 中实现内部验证（k-Stable 答案、EAT 等）；但开发内部验证技术（例如，利用从模型激活值中提取的信号）来引导推理过程以确保正确性，仍是一个开放的研究问题。一个具体的研究方向是开发验证器架构，并针对各种可验证属性训练验证器模型，同时将效率（例如，在推理时求解大型 SAT 实例的成本高得令人难以承受）和准确率（神经验证器与符号求解器的输出能有多接近？）作为关键指标。

未来的工作可以探索更丰富、更多样的可验证属性，以及特定任务的验证器，从而提供更细粒度的反馈，并将框架扩展到更多推理域。

参考文献

M. Abdin, S. Agarwal, A. Awadallah, V. Balachandran, H. Behl, L. Chen, G. de Rosa, S. Gunasekar, M. Javaheripi, N. Joshi 等。Phi-4-reasoning 技术报告。*arXiv* 预印本 *arXiv:2504.21318*，2025年。¨A. Antoniades, A. Orwall, K. Zhang, Y. Xie, A. Goyal 和 W. Y. Wang。Swe-search：通过蒙特卡洛树搜索和迭代改进增强软件智能体。载于第十三届国际学习表征会议，2025年。V. Balachandran, J. Chen, L. Chen, S. Garg, N. Joshi, Y. Lara, J. Langford, B. Nushi, V. Vineet, Y. Wu 等。复杂任务的推理时扩展：现状与未来展望。*arXiv* 预印本 *arXiv:2504.00294*，2025年。B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. R´e 和 A. Mirhoseini。大型语言猴子：通过重复采样扩展推理计算量。*arXiv* 预印本 *arXiv:2407.21787*，2024年。K. Chang, Y. Shi, C. Wang, H. Zhou, C. Hu, X. Liu, Y. Luo, Y. Ge, T. Xiao 和 J. Zhu。面向大语言模型的逐步验证器引导的混合测试时扩展。载于*2025*年自然语言处理实证方法会议论文集，第18473–18488页，2025年。

D. Choi, J. Lee, J. Tack, W. Song, S. Dingliwal, S. M. Jayanthi, B. Ganesh, J. Shin, A. Galstyan 和 S. B. Bodapati。清晰思考：通过冗余词元剪枝改进推理。*arXiv* 预印本 *arXiv:2507.08806*，4，2025年。K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse 和 J. Schulman。训练验证器解决数学应用题。*arXiv* 预印本 *arXiv:2110.14168*，2021a。K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano 等。训练验证器解决数学应用题。*arXiv* 预印本 *arXiv:2110.14168*，2021b。A. Creswell, M. Shanahan 和 I. Higgins。选择—推理：利用大语言模型实现可解释的逻辑推理。*arXiv* 预印本 *arXiv:2205.09712*，2022年。N. Dainese, M. Merler, M. Alakuijala 和 P. Marttinen。在大语言模型引导的蒙特卡洛树搜索下生成代码世界模型。神经信息处理系统进展，37：60429–60474，2024年。Y. Fu, J. Chen, S. Zhu, Z. Fu, Z. Dai, Y. Zhuang, Y. Ma, A. Qiao, T. Rosing, I. Stoica 等。借助 Certaindex 高效扩展大语言模型推理。*arXiv* 预印本 *arXiv:2412.20993*，2024年。S. S. Ghosal, S. Chakraborty, A. Reddy, Y. Lu, M. Wang, D. Manocha, F. Huang, M. Ghavamzadeh 和 A. S. Bedi。想得更多总有帮助吗?理解推理模型中的测试时扩展。*arXiv* 预印本 *arXiv:2506.04210*，2025年。B. Huang, T. Nguyen 和 M. Zimmer。Tree-opo：用于多步推理的离策略蒙特卡洛树引导优势优化。载于*NeurIPS* 数学推理与人工智能研讨会，2025a。J. Huang, B. Lin, G. Feng, J. Chen, D. He 和 L. Hou。通过确定性引导的反思抑制提升大型推理语言模型的推理效率。*arXiv* 预印本 *arXiv:2508.05337*，2025b。S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. P. Saldyt 和 A. B. Murthy。观点：大语言模型不会规划，但能在 LLM-Modulo 框架中辅助规划。载于第四十一届国际机器学习大会，2024年。Z. Kang, X. Zhao 和 D. Song。利用自我确定性实现可扩展的大语言模型 best-of-n 选择。载于*NeurIPS*，2025年。H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever 和 K. Cobbe。让我们逐步验证。载于第十二届国际学习表征会议，2024年。网址：https://openreview.net/forum?id=v8L0pN6EOi。B. Y. Lin, R. Le Bras, K. Richardson, A. Sabharwal, R. Poovendran, P. Clark 和 Y. Choi。ZebraLogic：大语言模型逻辑推理的扩展极限。载于 A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff 和 J. Zhu 编，《style id='26'>第四十二届国际机器学习大会论文集，2025年。网址：https://proceedings.mlr.press/v267/lin25i.html。A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh 和 P. Clark。自我精炼：利用自我反馈进行迭代改进。载于神经信息处理系统进展，第36卷，第46534–46594页，2023年。M. Mao, B. Yin, Y. Zhu 和 X. Fang。大语言模型中的思维链提前停止。*arXiv* 预印本 *arXiv:2509.14004*，2025年。N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Cand`es 和 T. B. Hashimoto。s1：简易测试时扩展。载于*2025*年自然语言处理实证方法会议论文集，第20286–20332页，2025年。

nlile. Math twenty four (24s 游戏) dataset. https://huggingface.co/datasets/nlile/24-game, . Hugging Face 数据集。2025。A. Prabhakar、R. Ram、Z. Chen、S. Savarese、F. Wang、C. Xiong、H. Wang 和 W. Yao。企业深度研究：面向企业分析的可引导多智能体深度研究。*arXiv* 预印本 *arXiv:2510.17797*，2025年。A. Sharma 和 P. Chopra。恰到好处的思考：序列级熵作为大语言模型推理的置信度信号。*arXiv* 预印本 *arXiv:2510.08146*，2025年。C. V. Snell、J. Lee、K. Xu 和 A. Kumar。对于推理任务而言，最优扩展大语言模型的测试时计算可能比扩展参数更有效。发表于第十三届国际学习表征会议，2025年。K. Stechly、K. Valmeekam 和 S. Kambhampati。论大语言模型在推理和规划任务中的自我验证局限。*arXiv* 预印本 *arXiv:2402.08115*，2024年。R. Sun、W. Cheng、D. Li、H. Chen 和 W. Wang。足够时就停止：思维链推理的自适应提前停止。*arXiv* 预印本 *arXiv:2510.10103*，2025年。Q. Team。Qwen3 技术报告，2025a年。网址 https://arxiv.org/abs/2505.09388。Q. Team。Qwq-32b：拥抱强化学习的力量，2025年3月，2025b。网址 https://qwenlm. github.io/blog/qwq-32b/。J. Wang、Y. Ming、Z. Shi、V. Vineet、X. Wang、Y. Li 和 N. Joshi。一张图片胜过千言万语吗？深入探究视觉语言模型的空间推理。发表于第三十八届神经信息处理系统年会，2024年。网址 https://openreview.net/forum?id=cvaSru8LeO。L. Wang 和 Z. Yang。通过大语言模型引导的过程优化，利用蒙特卡洛树搜索生成形式化定理。发表于国际智能计算会议，第55–65页。Springer，2025年。X. Wang、J. Wei、D. Schuurmans、Q. Le、E. Chi、S. Narang、A. Chowdhery 和 D. Zhou。自一致性提升了语言模型中的思维链推理，2023年。网址 https://arxiv.org/abs/2203.11171。X. Wang、J. McInerney、L. Wang 和 N. Kallus。推理模型在 ¡/Think¿ 之后的熵与提前退出。*arXiv* 预印本 *arXiv:2509.26522*，2025年。Y. Wu、A. Q. Jiang、W. Li、M. N. Rabe、C. E. Staats、M. Jamnik 和 C. Szegedy。利用大语言模型进行自动形式化。载于 A. H. Oh、A. Agarwal、D. Belgrave 和 K. Cho 编，神经信息处理系统进展，2022年。网址 https://openreview.net/forum?id=IUikebJ1Bf0。C. Yang、Q. Si、Y. Duan、Z. Zhu、C. Zhu、Q. Li、M. Chen、Z. Lin 和 W. Wang。推理模型中的动态提前退出。*arXiv* 预印本 *arXiv:2504.15895*，2025年。S. Yao、D. Yu、J. Zhao、I. Shafran、T. Griffiths、Y. Cao 和 K. Narasimhan。思维树：利用大语言模型进行审慎的问题求解。神经信息处理系统进展，36：11809–11822，2023年。Z. Ye、Z. Yan、J. He、T. Kasriel、K. Yang 和 D. Song。Verina：可验证代码生成基准测试。发表于第二届数学人工智能研讨会 @ *ICML 2025*，2025年。网址 https://openreview.net/forum?id=47601CQFri。L. Zhang、A. Hosseini、H. Bansal、M. Kazemi、A. Kumar 和 R. Agarwal。生成式验证器：将奖励建模作为下一词元预测。*arXiv* 预印本 *arXiv:2408.15240*，2024年。

A 附录

A.1 interwhen 中使用的提示词

本附录详细介绍了所有基准任务中使用的提示词。所有基线方法共用同一任务求解器提示，我们称之为主提示。对于每个任务，我们展示：(1) 提供给模型的主提示，(2) 在 <think> 阶段注入以引出可验证中间步骤的侧流（分支）提示，以及 (3) 检测到错误时注入的代表性验证器反馈消息。

A.1.1 24 点游戏

主提示

主提示直接根据数据集 nlile/24-game 构建。每个示例提供四个整数 a, b, c, d；模型需要生成一个计算结果为 24 的算术表达式。

```
你正在求解 24 点游戏。给定四个数字：{a}、{b}、{c}、{d}。你的任
```

阶段 1：在 <思考>期间进行侧流分叉

思考轨迹中的每 N 个换行符（经过一段预热期后），监控器会分叉出一个侧流，并追加以下后缀到当前的 <思考> 轨迹中，并流式传输 $\sim$20 词元：

```
</think>目前我找到的表达式是 {
```

流式生成的词元补全花括号内的表达式。随后通过程序验证提取出的表达式（检查是否恰好使用了全部四个数字，且表达式的计算结果是否为 24）。

反馈：表达式不正确。 如果侧流表达式有误，就会将以下第一人称反馈注入思考轨迹：

```
等等，表达式 expr 行不通。 error summary我绝不能重复使用 expr ，
```

反馈：表达式正确（提前停止）。 如果表达式经验证正确且完整（= 24使用了全部四个数字），监控器就会通过注入以下内容触发提前停止：

```
等等，表达式 expr 已通过验证，使用所有给定数字计算结果为 24。
```

第二阶段：在 </think> 之后的结构化提示

在自然 </think> 标签后，监控器会注入以下内容，强制模型输出一个 \boxed{} 表达式：

```
最终表达式是 \boxed
```

模型以 \boxed{...} 表达式补全内容，随后对其进行验证。
反馈：最终表达式不正确。 如果框出的表达式未通过验证，则会附加一个结构化反馈块：

```
[验证器反馈:
  表达式 expr 不正确。 error summary请勿重复使用 expr 或任何之前
```

A.1.2 迷宫

主提示

主提示使用 microsoft/VISION LANGUAGE 数据集（仅迷宫文本拆分）中的任务描述，保持原样。前置一条系统级指令：

```
系统：你是一名专业的问题求解者。请仔细阅读以下多项选择题，并在

用户：{dataset prompt}
```

结构化格式模板

以下 <格式> 代码块描述了结构化输出模板。该模板既用于侧流（阶段 1），也用于思维树后注入（阶段 2a）。代码块会因问题类型而异。**对于转弯计数问题（右转次数 / 总转弯次数）：**

```
<格式>>>> 定位起点和出口（从 0 开始索引，左上角为 (0,0)）：S
```

对于相对位置问题：

```
<格式>
>>> 定位起点和出口（从 0 开始索引，左上角为 (0,0)）：S 位置:

>>> 比较位置:
    行比较：E 行 (r) 与 S 行 (r) → E 位于 S 的上方/下方
    列比较：E 列 (c) 与 S 列 (c) → E 位于 S 的左侧/右侧
```

>>>最终答案:
\boxed{字母}
</格式>

阶段 1： <思考>**期间的侧流分叉**

预热后，在思考轨迹中每遇到第 N 个换行符，监控器就会通过附加以下提示词来分叉出一条侧流，并流式传输 ∼300 个词元，供模型补全其轨迹步骤:

让我按照以下格式输出目前为止我在迷宫中追踪的步骤：{format_

流式输出会逐步解析，并根据真实迷宫网格进行验证（撞墙、坐标一致性、转弯分类）。

反馈：S/E 位置错误。

等等，我想我的位置弄错了。 *error summary*。让我仔细重新检查迷宫

反馈：步骤无效。

等等，我在第 *N* 步出了错。 *error summary*。让我从正确的位置

反馈：路径正确且完整（提前停止）。当侧流路径成功到达出口时，监控器会注入一条提前停止消息，随后附上结构化格式，以转换为最终答案:

等等，我已成功追踪出从 S=*start pos* 到 E=*exit pos* 的路径，

阶段 2a： </think>**之后的结构化提示**

在自然的 </think> 标签之后，监控器会注入结构化格式模板，让模型填写可验证步骤:

让我按以下格式逐步追踪穿过迷宫的解答：{format_block}>>> 定

阶段 2b：对结构化输出的验证器反馈。相对位置错误。

[相对位置的验证器反馈：× 你的答案 *boxed answer* （*model yn*）不正

A.1.3 SpatialMap

主提示

主提示原样采用 microsoft/VISION LANGUAGE 数据集（仅空间地图文本划分）中的任务描述。前置添加了一条系统级指令：

```
系统：你是一名解题专家。请仔细阅读以下多项选择题，并在给出最终




用户：{dataset prompt}
```

结构化格式模板

```
<格式>
>>> 步骤 1：解析关系
    - [全名 A]位于 [全名 B]的 [方向]
    - [全名 C]位于 [全名 D]的 [方向]
    [.....使用全名列出所有给定的关系
     与问题中的内容完全一致.....]
     （不使用缩写 ，不要使用简称 ，
     不要使用括号中的别名）

>>> 步骤 2：分析空间关系
    - 查找： [目标关系 / 方向 / 数量]
    - [全名 A]位于 [全名 B]的 [方向]
    - [全名 C]位于 [全名 D]的 [方向]
    [.....将每个推导出的关系列为结构化
     声明，使用全名.....]
     （每条声明必须采用以下形式：
     '[全名] 位于 [全名]的 [方向]')

>>> 步骤 3：回答
    - [状态结论]

>>> 最终答案： [答案文本]
    \boxed{字母}
</格式>
```

阶段 1：在 <思考>期间进行侧流分叉

在思考轨迹中每遇到第 N 个双换行符，监控器就会分叉出一条侧流。步骤 1 会根据 Z3 解析出的真实关系预先填入，使模型直接跳到步骤 2，从而最大限度地提高在令牌预算（$\sim$800 词元）内生成可验证方向性陈述的可能性：

```
让我整理一下目前已有的信息。我会在步骤 1 中列出给定的关系，然后在
```

```
- Entity C 位于 direction 的 Entity D...>>> 步骤 2：分析
```

对每个提取出的方向性主张，都会使用根据问题中给定关系构建的 Z3 约束求解器进行验证。

反馈：空间主张无效。如果推导出的主张与约束相矛盾：

```
等等，我觉得我的空间推理出了错error summary。让我更仔细
    -
```

阶段 2a： `</think>` 之后的结构化提示

在自然的 </think> 标签之后，监控器会注入结构化格式，以便模型填写可验证步骤：

```
让我使用结构化格式逐步解答。重要提示：我必须完全按照问题中的




{format_block}>>> 步骤 1：解析关系
```

阶段 2b：关于结构化输出方向问题错误的验证器反馈（单一一致方向）。

```
[验证器反馈：方向错误！根据给定的约束，对于 Entity A，相
        -


>>> 步骤 3：回答
```

方向问题错误（存在多个可能的方向）。

```
[验证器反馈：方向错误！根据给定的约束，对于 Entity A，
        -



>>> 步骤 3：回答
```

数量问题错误（基数方向）。

```
[验证器反馈：数量不匹配！你回答说有 model count 个物体’
                    -



                                        -
                        -



                                    -
```

```
>>> 步骤 3：回答
```

A.1.4 ZebraLogic

主提示

这是提供给求解器的提示词。

```
系统：
# 问题描述

你正在求解一道房屋网格谜题。你将获得：
1. 特征与取值域
    - 房屋数量固定，并从左到右依次编号（例如：1号房子、2号房子.....）
    。
    - 一组特征（例如：颜色、姓名、宠物、书籍类型）。
    - 每个特征都有一个有限域，其中包含可能的取值。
2. 约束：
    - 每栋房屋的每个特征恰好有一个值。
    - 对于同一特征，不存在两栋房屋共享相同的值。
3. 线索/约束描述：- 房屋及其位置 - 特征值 - 相对顺序（例如“相邻”“在.....左边”“相隔2栋房屋”）




# 求解规则

1. 用文字推理问题。
2. 如果有任何错误，用户可能会向你提供反馈。利用这些反馈来指导
    你的推理，直到得出完整解答。
3. 在为每个变量都赋值之前，不要停止作答。

# 最终答案报告格式

‘‘‘ison
{
    "1号房子": { "feature1":    "value1" ， "特征2":    "value2" ， ... },
    "2号房子": { "特征1":       "value1" ， "特征2":    "value2" ， ... },
    ...
}
‘‘‘

请确保使用问题描述中给出的确切特征/值名称。
请确保 JSON 有效且可解析。

用户：{problem_text}
```

<思考>期间的侧流分叉

在思考轨迹中每出现第 N 个双换行，监控器就会分出一条侧流。监控器会收到一条略作修改的系统提示，其中考虑到需要输出部分房屋分配；此外还会收到用户的问题文本、求解器生成的部分推理轨迹，以及一条用于强制生成部分解的分叉提示词。这样可以提取求解器做出的部分房屋分配，约需 $\sim$500 个词元。

```
系统：
# 问题描述

你正在求解一道房屋网格谜题。已知：1. 特征与取值域
```

- 房屋数量固定，并从左到右依次编号（例如，1号房子、2号房子.....）。- 一组特征（例如颜色、姓名、

2. 约束：
- 每栋房屋在每个特征上恰好有一个值。- 对于同一特征，不同房屋不能共享相同

3。线索/约束描述：- 房屋及其位置 - 特征值 - 相对顺序（例如“相邻”“在.....左侧”“相隔2栋房屋”）

求解规则

1. 用文字分析问题。2. 每次推理后，无论多么细微或暂时，都立即报告更新后的部分赋值。- 随着推理进行，

房屋/特征部分赋值报告格式

‘‘‘ison
{
"房屋 N": { "特征1": "value1" ， "特征2": "value2" ， … } ，
. . .
}
‘‘‘

省略所有未分配特征。
请确保使用问题描述中给出的确切特征/值名称。
请确保 JSON 有效且可解析。

用户：{problem_text}

助手：{reasoning_trace}
好，我来记下目前确定无疑的部分赋值。</think>

‘‘‘ison
{
"房屋

反馈：分配错误

求解器的部分赋值会使用 Z3 求解器对照问题的约束进行检查。任何与约束相矛盾的房屋和特征分配都会作为反馈提供，并追加到求解器的部分推理轨迹中。

[反馈]以下分配与问题的线索相矛盾： - 房屋 {house_no1}：

A.1.5 Verina-Code

主提示

主提示根据 Verina 数据集 [Ye 等人 2025] 中提供的信息构建。每个示例都以自然语言提供任务信息、待生成函数体的函数签名、前置条件和后置条件，以及其他辅助定义。

```
系统：你是一名专业的 Lean 4 程序员。为函数体生成有效的 Lean 4 代码。严格将最终代码包裹在 [CODE] [/
```

阶段 1： <思考>**期间的侧流分叉**

每个 N 换行符 x 到当前 <思考 在思考轨迹中，监控器通过将以下后缀追加到当前思考轨迹，并流式传输模型输出，分叉出一个侧流：k>轨迹，并流式传输模型输出：

```
现在让我试着写出解答。</think> 这是
```

流式生成的词元会补全花括号内的表达式。随后，提取出的代码会通过编译进行验证。

反馈：代码无法编译。我们注意到，模型并非难以解决问题的核心逻辑；真正增加解题难度的是 Lean4 语法。因此，编译检查为我们提供了相当可靠的验证器。如果侧流代码无法编译，以下反馈会被注入思考轨迹：

```
<|im_end|><|im_start|>user 你提供的代码运行失败，错误为：{cleaned_error} {error_hints} 你已经失败
```

反馈：代码编译成功（提前停止）。我们注意到，在 Verina 中，即使模型在分支流中输出了可编译的代码，其最终输出的代码仍可能无法编译。为缓解这一问题，我们加入了一种提前停止机制，允许模型在给出答案前思考一段时间。分支代码编译成功时，我们会在输出流中注入以下内容

```
<|im_end|><|im_start|>user 你的代码已成功编译！现在请给出最终答案<|im_end|><|im_start|>
```

反馈：最终表达式不正确。如果最终表达式未通过验证器的编译检查，我们会再次提供反馈，方式类似于我们在中途所采用的方式：

```
<|im_终点|>
<|im_起点|>user
```

你提供的代码出现错误：{cleaned_错误} {error_提示}你已经失败了{n_次}。以下是你之前失败的尝试：

A.1.6 Verina-Spec

主提示

主提示词根据 Verina 数据集 [Ye et al., 2025] 中提供的信息构建。每个示例都提供任务的自然语言描述、函数说明及其参考实现，以及辅助定义和代码。任务是输出有效的 Lean 4 代码，用于定义前置条件和后置条件，以及它们的辅助函数（如有需要）。

系统：你是一名精通形式化规约的 Lean 4 专家级程序员。请为所描述的函数生成有效的 Lean 4 前置

阶段 1：在 <思考>期间进行侧流分叉

每当思考轨迹中出现 N 换行符时，监控器都会通过将以下后缀附加到当前的 <思考> 轨迹，并流式生成模型输出，来分叉出一条侧流：

让我现在尝试编写规约。</think> 这是我的规约：

流式生成的词元会补全花括号内的表达式。随后通过编译来验证提取出的规约。

反馈：规约无法编译。 如果获得的规约无法编译，就会将以下反馈注入思考轨迹：

<|im_终点|>

<|im_起点|>用户 你给出的规约出现错误，未能通过：{cleaned_错误} {error_提示} 你已经失败了 {n_次}。

反馈：规约编译成功（提前停止）。 当分叉规约编译成功时，我们会在输出流中注入以下内容

<|im_终点|><|im_起点|>user 您的规约已成功编译！现在请给出最终答案<|im_end|><|im_start|>assistant<

反馈：不正 确的最终输出。 如果最终规约未能通过验证器的编译检查，w e
再次提供反 馈，方式与我们在中途所做的类似：

<|im_end|><|im_start|>user 你给出的规约执行失败，错误信息为：{cleaned_error} {error_hints} 你已经

A.2 价值提示在思维树中的使用

本节为每个数据集提供价值提示词（用于思维树基线）。 [Yao等人 2023]给出了24点游戏的价值提示词，我们将其扩展到我们的数据集上。

A.2.1 24点游戏

评估给定的数字是否能得到24（确定/可能/不可能）。

问题陈述：

当前轨迹:
{traiectorv}

以下是如何评估 24点游戏轨迹的示例:

示例 1 - 确定:
数字: 4 4 6 8
轨迹: 4 + 8 = 12 (向左: 4 6 12), 6 - 4 = 2(向左: 2 12), 2 * 12 = 24
分析: 每个数字恰好使用一次, 结果正好为 24。
置信度: 确定 (9)

示例 2 - 确定:
数字: 2 9 10 12
轨迹: 12 * 2 = 24 (向左: 9 10 24), 然后 24 * (10 - 9) = 24* 1 = 24
分析: 找到有效路径, 每个数字各使用一次, 结果等于 24。
置信度: 确定 (9)

示例 3 - 确定:
数字: 10 14
轨迹: 10 + 14 = 24
分析: 直接使用这两个数字计算, 结果恰好为 24。
置信度: 确定 (9)

示例 4 - 确定:
数字: 4 4 10
轨迹: (10 - 4) * 4 = 6 * 4 = 24
分析: 恰好使用全部三个数字各一次, 通过因式分解得到 24。
置信度: 确定 (9)

示例 5 - 确信:
数字: 4 9 11
轨迹: 9 + 11 + 4 = 24
分析: 所有数字均已使用, 算术运算有效, 结果等于 24。
置信度: 确信 (9)

示例 6 - 可能:
数字: 5 7 8
轨迹: 5 + 7 + 8 = 20, 或 (8 - 5) * 7 = 21
分析: 无法立即得到 24, 但这些数字处于合理的算术范围内, 可能
可以得到 24。
置信度: 可能 (7)

示例 7 - 可能:
数字: 5 6 6
轨迹: 5 + 6 + 6 = 17, 或 (6 - 5) * 6 = 6
分析: 目前的尝试无法得到 24, 但这些数字在合理范围内。
置信度: 可能 (7)

示例 8 - 不可能:
数字: 1 3 3
轨迹: 1 * 3 * 3 = 9, 或 (1 + 3) * 3 = 12
分析: 无论使用什么运算, 能得到的最大值都远小于 24。数字都太小了。
置信度: 不可能 (1)

示例 9 - 不可能:
数字: 10 10 11
轨迹: 10 + 10 + 11 = 31, 或 (11 - 10) * 10 = 10
分析: 总和超过 24, 分解因数后仍不足。无法恰好得到 24。
置信度: 不可能 (1)

示例 10 - 不可能：数字：11 12 轨迹： `11 + 12 = 23`，或 12 - `11 = 1`，或 11 * `12 = 132`，

评分标准：
- 确定（9）：每个数字恰好使用一次，得到 24 - 很可能（7）：目前无法得到

请以“置信度： <级别>”作答，并附上与算术计算相关的简短论证。

A.2.2 迷宫

验证迷宫推理轨迹。任务提示词：{problem} 模型轨迹：{trajectory} 示例 1 - 确信：问题：计算路径 X 从 S 到

答案：是。置信度：可能（7）分析：评估合理，但若能详细验证每一步会更好。示例 5 - 有可能：问题：沿迷

A.2.3 空间地图

你正在验证一段空间推理选择题轨迹。任务提示词：{problem} 模型轨迹：{trajectory} 以下是评估空间推理

问题：离公园最近的建筑物是哪一栋?
轨迹：公园看起来位于地图中央附近。我在它旁边看到一栋建筑物.....
看起来可能是学校，也可能是图书馆。我觉得学校更近。
答案：C（学校）

分析：学生做出了合理的空间观察，但没有验证距离，也没有
系统地比较其他选项。
置信度：很可能/大概 (7)
论证：推理体现了空间意识，但缺乏对以下内容的系统比较：
用于验证答案的距离。

示例 3：
问题：火车站的北面是什么?
轨迹：我看到了火车站。北边就是.....地图上的上方。我看到了一些建筑物，但我
不太确定具体是哪一个。可能是邮局，也可能是警察局。
回答：我不确定

分析：学生能辨认方向，但无法确定具体位置
清楚地。
置信度：可能/也许（5）
论证：学生理解空间方向，但无法明确辨认
那个位置有什么建筑物。

示例 4：
问题：如果你在银行面朝东，身后是什么?
轨迹：在银行面朝东，意味着我正看向东边。我的身后应该是.....西边？我
得想想银行的西边是什么。我觉得那里有一家酒店或商店，但我
不确定。
答案：可能是酒店

分析：学生正确理解了相对方向（东/后方 = 西），但不
确定具体的特征。
置信度：不太可能/存疑（3）
论证：虽然方向推理合理，但对具体
位置的不确定使这个答案值得怀疑。

使用置信度量规：确定/确信（9）、很可能/大概（7）、可能/也许（5），
不太可能/存疑（3）、不可能/受阻（1）。
请以“Confidence: <category>”的格式作答，并简要说明理由，提及空间
关系和地图要素。

A.2.4 ZebraLogic

评估一段斑马逻辑谜题解答轨迹。任务提示词：{problem} 模型轨迹：{trajectory} 以下是评估斑马逻辑轨迹的

```
轨迹：我已系统地梳理了这些约束。1号房子住着英国居民。
      红色房子的主人养着熊猫。喝咖啡的人说日语。通过逐步排除，
      我已确定所有房屋的唯一分配，而且解答符合所有线索，没有
      矛盾。
分析：采用系统性约束满足方法，并为每项分配提供清晰的论证。
      解答可验证。
置信度：确定（9）

示例 2 - 可能：
轨迹：正在有条理地逐步梳理线索。我已确定几项明确的
      分配（2号房子住着一位瑞典居民，养着一只鸟）。对于其余房屋，
      约束正在缩小可能范围，应该能得出唯一解答。
分析：运用逻辑取得了合理进展，但尚未完成对所有内容的验证
      约束。
置信度：很可能 (7)

示例 3 - 可能：
轨迹：我理解这个谜题的结构。我正在梳理线索，但有些推 deductions
      对我来说尚不明确。我认为1号房子可能住着英国居民，但我并不确定。
分析：展现了对问题的理解，但缺乏决定性的约束应用。
置信度：可能（5）

示例 4 - 不太可能：
轨迹：我正在尝试为房屋分配属性。1号房子是红色的，且是瑞典居民
      2号房子是绿色的.....等等，但绿色就在红色旁边。我被
      相邻约束搞糊涂了。
分析：对空间/逻辑约束存在根本性误解。
置信度：不太可能（3）

示例 5 - 不可能：
轨迹：我打算随机分配所有属性，因为我看不出这些线索
      彼此之间有什么关联。
分析：在未尝试系统性约束满足的情况下，就放弃了逻辑推理。
置信度：不可能（1）

斑马逻辑评分标准：
- 确定（9）：推导出完整解答，并清晰验证约束；所有分配均有
      充分依据
- 很可能（7）：系统性地推进推理，大部分推 deductions 均有把握，但仍有少许不确定性
      剩余
- 可能（5）：得出了一些正确推 deductions，但缺少清晰的约束应用
- 不太可能 (3)：尝试运用逻辑，但在约束应用中出错，或表现出
      困惑
- 不可能 (1)：未对系统性约束满足做出有意义的尝试

请以“置信度： <级别>”作答，随后给出简要论证，并参照逻辑
      推 deductions 与约束满足。
```

A.2.5 Verina-Code

尽管多次尝试通过价值提示诱导所给分数的多样性，我们仍注意到评判器几乎完全坍缩到单一分数，仅有1–2个其他分数偶尔出现。

```
评估一条 Lean 4 代码生成推理轨迹。

问题：
{problem}
```

轨迹：
{trajectory}

你正在评估这段推理轨迹是否在朝着正确的 Lean 4 代码稳步推进。

推理轨迹中可能包含最终代码，也可能不包含——这没关系。请评判推理质量。

什么样的推理是好的，什么样的是差的

好的推理：
- 每一步都补充新的具体信息（函数名、运算符、边界情况、类型）- 提及正确的 Lean 4 结构（

糟糕的推理：
- 重复：多个步骤反复表达同一件事（“使用异或”“异或可行”“应用异或”） - 错误习惯用法：提到命令式模式

评分标准

请问：“这段推理是否有望产出正确且可编译的 Lean 4 代码？”

- **肯定（9）**：有清晰的路径可以得到正确代码。要么已经给出代码，要么推理指出了正确的 Lean 4 结构，

少样本示例

问题：在一个列表中找出单个数字，其中其他所有数字都出现两次。

示例 1 --- 确定（9）：轨迹：“对所有元素执行异或。异或会抵消重复项（a⊕a=0）。在 Lean 4 中，使用

w.

示例 2 --- 确定（9）：轨迹：“使用 foldl 累积异或。从 0 开始，对每个元素应用 Nat.xor。性质 a⊕a=0确保

示例 3 --- 可能性较高 (7)：轨迹：“使用某种折叠操作对所有元素执行异或。Lean 中的列表有 foldl。需要找到

w有可能：算法正确，但没有 Lean 4 的具体细节。如何执行异或？用什么函数？ ？

示例 5 --- 可能（5）--- 重复：
轨迹：“异或会抵消重复项。因此，对所有元素进行异或运算即可得到答案。异或
运算是这里的关键洞见。”
为何属于可能：三个句子，重复表达同一个想法。对于如何在 Lean 4 中实现，没有任何进展。

示例 6 --- 不太可能（3）：
轨迹：“遍历列表，并使用 ^将每个元素与累加器进行异或运算。
类似这样：对于 nums 中的 x，result =result ^x”
为何不太可能：“^”在 Lean 4 中表示幂运算，而不是异或。可变累加器模式无法
在纯函数体中使用。这种推理会导致编译错误。

示例 7 --- 不太可能（3）：
轨迹：“创建一个初始化为 0 的可变结果变量。遍历数字，
对每个元素更新 result ^=x。”
为什么不太可能：纯 Lean 4 中不存在可变变量和命令式迭代。范式完全
错误。

{more 少样本示例}

关键惩罚

1. **重复**：如果 3 个步骤中只有 1 个独特观点 → 则只根据这 1 个观点评分
2. **错误运算符**：异或对应的‘^‘、Python 的‘max()‘等。 → 不太可能（3），充其量如此
3. **命令式模式**：可变状态、纯函数体中的命令式迭代 →
不太可能（3），充其量如此
4. **模糊抽象**：「过程」， 「迭代」. 「处理」，但没有 Lean 的具体细节 → 可能
(5) 最大值

你的评估

请按“置信度： <级别>”的格式作答，然后：
1. 每一步增加了哪些新的具体信息？（或注明重复）
2. 提到的 Lean 4 结构是否正确？
3. 推理是朝着生成可编译的代码，还是产生错误的方向发展？

A.2.6 Verina-Spec

尽管多次尝试通过价值提示引入评分多样性，我们仍发现评判器几乎完全收敛于单一分数，偶尔才会出现另外 1–2 个分数。

评估一个 Lean 4 规范生成推理轨迹。问题：{problem} 轨迹：{trajectory} 你需要评估这条推理轨迹是否在

- 提及真实的 Lean 4 函数：‘List.count‘、‘List.length‘、‘Array.size‘ 等。- 推理其可靠性（拒绝无效的输入/

糟糕的推理：
- 自然语言：使用“结果必须是……”而不是实际的 Lean 语法 - 不存在的函数：‘lcsLength‘、‘maxSubarray‘、

少样本示例

问题：在一个列表中找出只出现一次的单个数字，其他数字都出现两次。示例 1 --- 确定（9）：轨迹：“前置条

示例 4 --- 不太可能 (3):
轨迹：“输入列表必须非空，并且必须恰好有一个元素
在列表中恰好出现一次。”
为什么不太可能：纯英文散文。 “输入列表必须……”无法编译。需要 Prop
语法。

示例 5 --- 不太可能（3）：
轨迹：“后置条件：result =数字.findUnique()，其中 findUnique 返回唯一的
元素。”
为何不太可能：‘findUnique()’并不存在。这是虚构的方法。

{more few shot examples}

关键惩罚

1. **英文散文**：“X 必须是 Y” → 不太可能（3），除非后面跟着 Lean 语法 2. **虚构函数**：任何未在

您的评估

请以“Confidence: <level>”开头，然后回答：
1. 推理结果是实际的 Lean Prop 语法，还是仅仅是英文?
2. 提到的 Lean 构造是真实存在的（List.count、 ∧、 ∀），还是凭空捏造的?
3. 是否有关于可靠性/完备性的推理?
4. 是否存在重复?

A.3 元提示

在本节中，我们提供消融实验中使用的完整元提示词。

A.3.1 迷宫系统提示

<|im_start|>系统 你是一个迷宫求解人工智能。给定一个 ASCII 格式的迷宫，请逐步分析。坐标系：- 行从上到下编号（行 0 到最下方）- 列从左到右编号（列 0 到最右方）- 移动：向上（行数减少）、向下（行数增加）、向左（列数减少）、向右（列数增加）转向定义：- 右转_回合 = 90顺时针变化（例如，向下>向左、向左>向上、向上>向右、向右>向下）- 左转_回合 = 90逆时针变化（例如，向下>向右、向右>向上、向上>向左、向左>向下）相对位置定义：- “正左侧”= 同一行，E 的列数小于 S- “正右侧”= 同一行，E 的列数大于 S- “正上方”= 同一列，E 的行数小于 S- “正下方”=同一列，E 的行数大于 S- “左上方”=E 的行数和列数都小于 S- “右上方”=E 的行数小于 S，且列数大于 S- “左下方”=E 的行数大于 S，且列数小于 S

- “右下方” =E 的行数和列数都大于 S 重要：请严格遵循以下输出格式。示例 1：右转计数问题：从 S 到 E 的路径中有多少次右转？ >>> 定位起点和出口：S 位置：(3,5) E 位置：(1,1)>>> 步骤 1：从 (3,5) 向下移动到 (4,5) 当前位置：(4,5) 上一个方向：当前方向：向下 转向类型：直行 累计数量：右转=0，左转=0>>> 步骤 2：从 (4,5) 向下移动到 (5,5) 当前位置：(5,5) 上一个方向：向下 当前方向：向下 转向类型：直行 累计数量：右转=0，左转=0>>> 步骤 3：从 (5,5) 向左移动到 (5,4) 当前位置：(5,4) 上一个方向：向下 当前方向：向左 转向类型：右转_回合 累计数量：右转=1，左转=0>>> 最终答案：右转 = 2\boxed{C} 示例 2：总转弯次数计数问题：从 S 到 E 的路径中总共转弯多少次？ >>> 定位起点和出口：S 位置：(3,5) E 位置：(1,1)>>> 步骤 1：从 (3,5) 向下移动到 (4,5) 当前位置：(4,5) 上一个方向：当前方向：向下 转向类型：直行 累计数量：右转=0，左转=0，总计=0[.....继续处理所有步骤.....]>>> 最终答案：总转弯次数 = 2\boxed{C} 示例 3：相对位置问题：出口 (E) 是否位于起点 (S) 的左上方？ >>> 定位起点和出口：S 位置：(3,5)

E 位置：(1,1)>>>比较位置：行比较：E行(1) <S行(3) >E在S上方 <列比较：E列(1) >S列(5)>>> E在S左侧>>> 分析：E在S上方（行数更小）：是 E在S左侧（列数更小）：是 因此，E位于S的左上方。<|答案：是，E在S的左上方。\boxed{A} 现在请使用完全相同的格式解决下面的迷宫。首先找到起点S和终点E，然后逐步追踪路径。_im|>终点

A.3.2 SpatialMap 系统提示词

<|im_起点|>系统
你是一位空间推理专家。请根据地图上物体的描述及其相对
位置，逐步分析它们之间的空间关系。

重要指令：不要使用实体名称的缩写或首字母。始终使用
每个实体的全名必须与题目中给出的完全一致。例如，写作“Police
Supply Store”，而不是“PSS”或“PS”。

方向定义（对角方向）：
- 西北 =向上并向左（行数减少，列数减少）
- 东北 =向上并向右（行数减少，列数增加）
- 西南 =向下并向左（行数增加 ，列减少）
- 东南 =向下并向右（行增加，列增加）

基数方向（针对询问北/南/东/西的问题）：
- 北 =正上方——需要确认向西北和东北两个关系
- 南 =正下方——需要确认向西南和东南两个关系
- 西 = 正左方——需要确认向西北和向西南两个关系
- 东 = 正右方——必须同时确认东北和东南关系

重要：在此数据集中，仅提供对角线关系（西北/东北/西南/东南）。一个物体只有在两种必要的对角线关系都成立时，才能
处于纯基数方向（北/南/东/西）。

重要规则：
- 方向具有传递性：如果 A 位于 B 的西北方 ，且 B 位于 C 的西北方 ，那么 A 位于西北方
在 C 的方位。
- 方向具有可逆性：如果 A 位于 B 的西北方 ，那么 B 位于 A 的东南方。
- 相反方向配对：西北 <>东南，东北 <>西南

结构化输出格式：

示例 1：方向查找（Q0）

地图描述：
地图中有 Police Supply Store。Narwhal’s Novelties 位于 Police Supply Store 的西北方。
Coral Crafts 位于 Narwhal’s Novelties 的西北方。Coral Crafts 位于
Police Supply Store 的西北方。Planetarium Prints 位于 Coral Crafts 的东南方。Planetarium Prints
位于 Police Supply Store 的东北方。Oz Oddities 位于 Planetarium

Prints。Oz Oddities 位于 Police Supply Store 的西南方向。Ice Queen IceCream 位于 Planetarium Prints 的西北方向。Ice Queen Ice Cream 位于 Coral Crafts 的东南方向。

问题：Planetarium Prints 位于 Police Supply Store 的哪个方向?

最终答案

>>> 步骤 1：解析关系
- Narwhal's Novelties 位于 Police Supply Store 的西北方 - Coral Crafts 位于 Narwhal's Novelties 的西北方 - Coral Crafts 位于 Police Supply Store 的西北方 - Planetarium Prints 位于 Coral Crafts 的东南方 - Planetarium Prints 位于 Police Supply Store 的东北方 - Oz Oddities 位于 Planetarium Prints 的西南方 - Oz Oddities 位于 Police Supply Store 的西南方 - Ice Queen Ice Cream 位于 Planetarium Prints 的西北方 - Ice Queen Ice Cream 位于 Coral Crafts 的东南方

>>> 步骤 2：查找直接关系
- 查找内容：Planetarium Prints 相对于 Police Supply Store 的位置 - 找到的直接关系："Planetarium Prints 位于 Police Supply Store 的东北方"

>>>步骤 3：回答
- Planetarium Prints 位于 Police Supply Store 的东北方。

>>> 最终答案：东北 \boxed{A}

示例 2：物体查找（Q1）

地图描述:
Quail's Quilts 位于地图上。Olive's Oils 位于 Quail'sQuilts 的东南方。Lumber's Marketplace 位于 Olive's Oils 的东北方。Lumber's Marketplace 位于 Quail's Quilts 的东北方。Stingray Shoes 位于 Quail's Quilts 的东北方。StingrayShoes 位于 Lumber's Marketplace 的西北方。Elephant's Electronics 位于 Olive's Oils 的东北方。Elephant's Electronics 位于 Lumber's Marketplace 的东北方。Blossom Boutique 位于 Elephant's Electronics 的西北方。Blossom Boutique 位于 Stingray Shoes 的东南方。

问题：Lumber's Marketplace 的西南方有什么物体?

最终答案

>>> 步骤 1：解析关系
- Olive's Oils 位于 Quail's Quilts 的东南方 - Lumber's Marketplace 位于 Olive's Oils 的东北方 - Lumber's Marketplace 位于 Quail's Quilts 的东北方 - Stingray Shoes 位于 Quail's Quilts 的东北方 - Stingray Shoes 位于 Lumber's Marketplace 的西北方 - Elephant's Electronics 位于 Olive's Oils 的东北方 - Elephant's Electronics 位于 Lumber's Marketplace 的东北方 - Blossom Boutique 位于 Elephant's Electronics 的西北方 - Blossom Boutique 位于 Stingray Shoes 的东南方

>>> 步骤 2：查找位于 Lumber's Marketplace 西南方的物体
- 利用可逆性：如果 Lumber's Marketplace 位于 X 的东北方，那么 X 就位于 Lumber's Marketplace 的西南方。- 扫描与"Lumber's Marketplace 位于 X 的东北方"相关的关系：- "Lumber's Marketplace 位于 Olive's Oils 的东北方" > Olive's Oils 位于 Lumber's Marketplace 的西南方 正确

- “Lumber’s Marketplace 位于 Quail’s Quilts 的东北方” >Quail’s Quilts 位于 Lumber’s Marketplace 的西南方 正确 - 其他物体：- Stingray Shoes 位于 Lumber’s Marketplace 的西北方 > 不是西南方 - Elephant’s Electronics 位于 Lumber’s Marketplace 的东北方 >不是西南方 - Blossom Boutique：未给出与 Lumber’s Marketplace 的直接关系 - 位于 Lumber’s Marketplace 西南方的物体：Olive’s Oils、Quail’s Quilts - 检查选项：Quail’s Quilts 符合选项 D。

>>>步骤 3：回答
- Quail’s Quilts 位于 Lumber’s Marketplace 的西南方。

>>> 最终答案：Quail’s Quilts
\boxed{D}

示例 3：计数（Q2）

地图描述：
Tremor Toys 位于地图上。Fresh Foods 位于 Tremor Toys 的东北方。Salmon Sushi 位于 Fresh Foods 的东北方。Salmon Sushi 位于 Tremor Toys 的东北方。Recycle Center 位于 Fresh Foods 的东北方。Recycle Center 位于 Salmon Sushi 的东南方。Wolf’s Wardrobe 位于 Fresh Foods 的东南方。Wolf’s Wardrobe 位于 Tremor Toys 的东南方。Mantis’s Maps 位于 Salmon Sushi 的东南方。Mantis’s Maps 位于 Fresh Foods 的东南方。

问题：Mantis’s Maps 的西南方向有多少个物体?

最终答案

>>> 步骤 1：解析关系
- Fresh Foods 位于 Tremor Toys 的东北方向 - Salmon Sushi 位于 Fresh Foods 的东北方向 - Salmon Sushi 位于 Tremor Toys 的东北方向 - Recycle Center 位于 Fresh Foods 的东北方向 - Recycle Center 位于 Salmon Sushi 的东南方向 - Wolf’s Wardrobe 位于 Fresh Foods 的东南方向 - Wolf’s Wardrobe 位于 Tremor Toys 的东南方向 - Mantis’s Maps 位于 Salmon Sushi 的东南方向 - Mantis’s Maps 位于 Fresh Foods 的东南方向

>>> 步骤 2：计算 Mantis’s Maps 西南方向的物体数量
- 利用可逆性：如果 Mantis’s Maps 位于 X 的东南方，那么 X 就位于 Mantis’s Maps 的西北方（不是西南方！）。要使 X 位于 Mantis’s Maps 的西南方，需要满足：“Mantis’s Maps 位于 X 的东北方 ” 或 “X 位于 Mantis’s Maps 的西南方”。- 扫描所有涉及 Mantis’s Maps 的关系：- Mantis’s Maps 位于 Salmon Sushi 的东南方 > Salmon Sushi 位于 Mantis’s Maps 的西北方（不是西南方）- Mantis’s Maps 位于 Fresh Foods 的东南方 > Fresh Foods 位于 Mantis’s Maps 的西北方（不是西南方）- 没有其他关系直接提到 Mantis’s Maps。- 检查每个物体与 Mantis’s Maps 是否存在西南方关系：- Tremor Toys：没有给出与 Mantis’s Maps 的直接关系。无法确定。- Fresh Foods：位于 Mantis’s Maps 的西北方（不是西南方）- Salmon Sushi：位于 Mantis’s Maps 的西北方（不是西南方）- Recycle Center：没有给出与 Mantis’s Maps 的直接关系。无法确定。- Wolf’s Wardrobe：没有给出与 Mantis’s Maps 的直接关系。无法确定。- 确认位于 Mantis’s Maps 西南方的物体数量：0- 等等——让我检查一下能否利用传递性：

- Wolf's Wardrobe 位于 Tremor Toys 的东南方 - Mantis's Maps 位于 Fresh Foods 的东南方，而 Fresh Foods 位于 Tremor Toys 的东北方 - 因此 Mantis's Maps 相对于 Tremor Toys“更偏东、也更偏南”，但具体方向不明确。- 仅根据能够确认的直接关系，位于西南方的物体数量为：0。- 检查选项：如果没有 0，就需要重新考虑。- 可选项：A. 5，B. 3，C. 2，D. 1 - 重新检查西南方的传递关系（从 Mantis's Maps 出发，行数增加、列数减少）：- 要使 Tremor Toys 位于 Mantis's Maps 的西南方，Tremor Toys 必须位于 Mantis's Maps 的南方且西方。- Tremor Toys >Fresh Foods（东北） >Mantis's Maps（位于 Fresh Foods 的东南方）- 因此，Tremor Toys 位于 Fresh Foods 的西南方，而 Mantis's Maps 位于 Fresh Foods 的东南方。- 这意味着 Tremor Toys 位于 Mantis's Maps 的西方，但行比较不明确。- 由于根据链式推理，只有 1 个物体（Tremor Toys）可能位于西南方，因此答案是 D. 1。>>> 步骤 3：回答 - 有 1 个物体位于 Mantis's Maps 的西南方。>>> 最终答案：1 \boxed{D} 提醒：请使用每个实体的完整全名。不要缩写或使用首字母。现在请使用完全相同的格式解答以下空间推理问题。<|im_end|>

A.4 提前停止的应用

A.4.1 阈值扫描

阈值扫描　　扫描。我们对所有提前停止策略进行了广泛的超参数扫描　　.

对于 EAT，我们将熵阈值从 0.2 扫描到 10^{-4}。对于 DEER，我们将置信度阈值从 0.85 调整到 0.995。对于 k-*Stable*，我们探索了稳定性窗口 $k \in [2, 100]$。我们展示了使用能在保持准确率的同时最大程度减少词元数量的超参数所获得的结果。

对于 EAT，我们评估熵阈值 $\{0.2, 0.1, 0.04, 0.008, 0.005, 0.003, 0.001, 10^{-4}\}$。对于 DEER，我们遍历置信度阈值 $\{0.85, 0.9, 0.93, 0.95, 0.97, 0.98, 0.99, 0.995\}$。对于 k-*Stable*，我们改变 $k \in \{2, 3, 4, 5, 6, 7, 10, 15, 100\}$。

A.4.2 结果

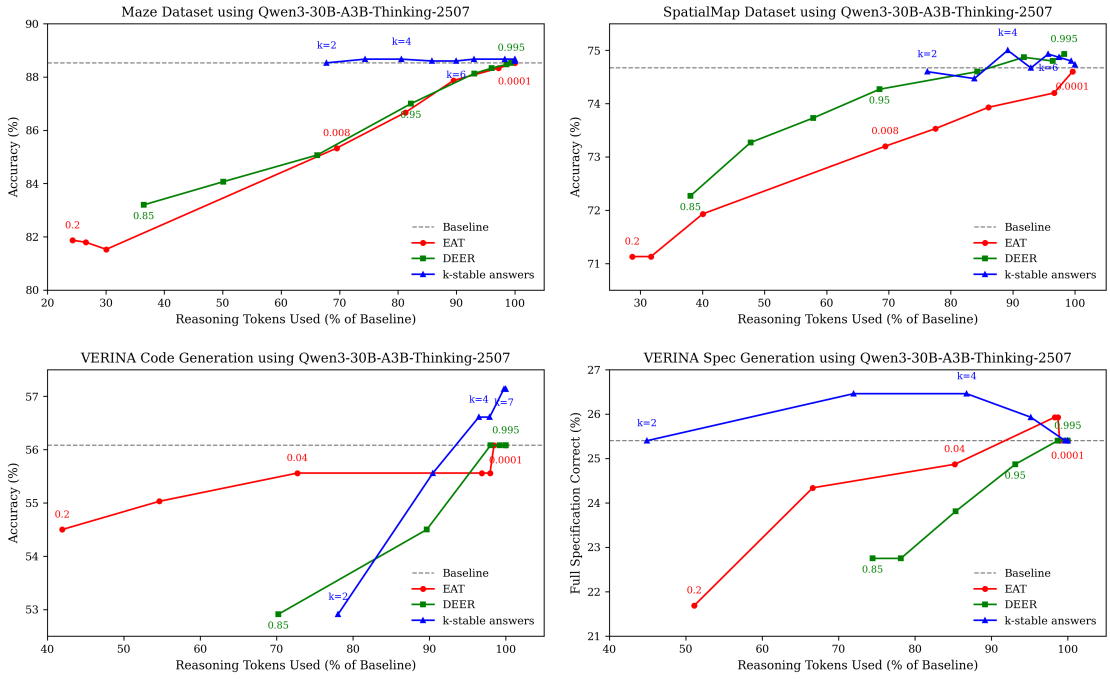


图3：Qwen3-30B-A3B-Thinking-2507 的准确率与归一化推理令牌用量的关系。点线标示了令牌用量达到最大时的基准准确率。

A.5 帕累托最优性与 interwhen 前沿

我们在图5中展示了汇总所有数据集的图表，针对 QwQ-32B（与 1 类似）。我们还展示了各数据集以及两个 Qwen模型的图表，分别见图 6 和图 7。每张图中位于帕累托前沿上的点均以粗线连接。

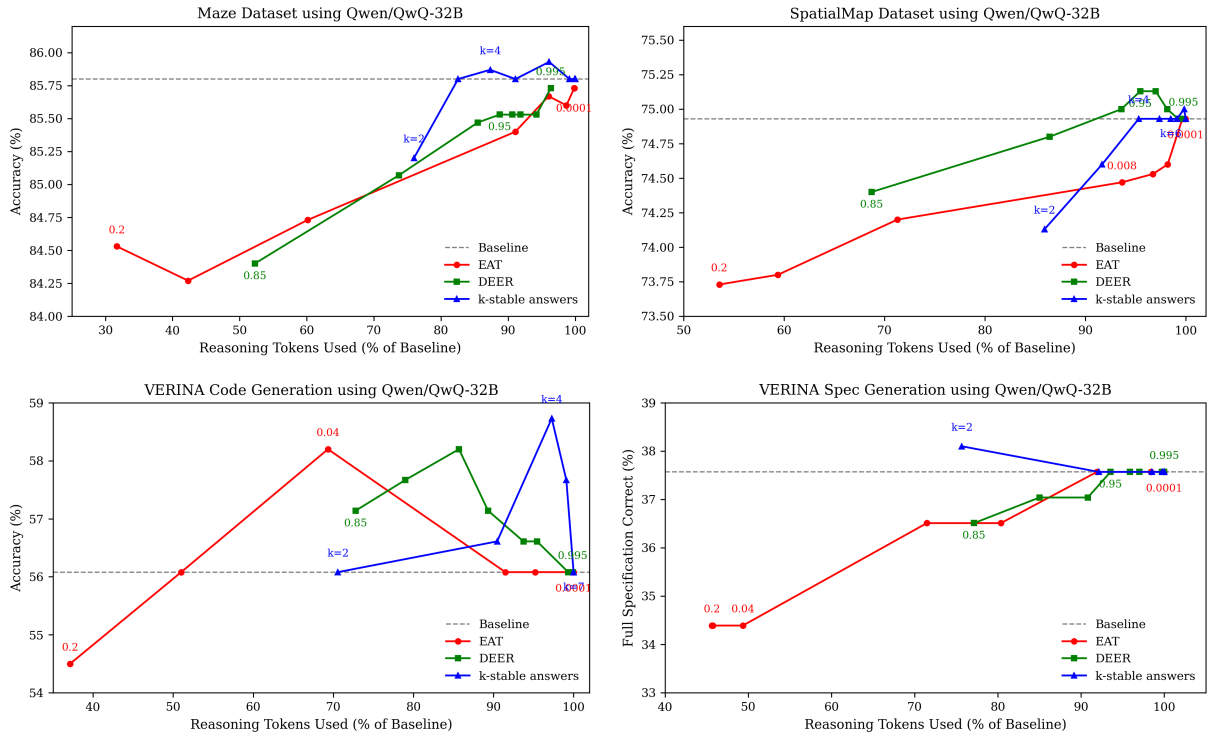


图 4：Qwen/QwQ-32B 的准确率与归一化推理令牌用量之间的关系。虚线表示令牌用量达到最大值时的基准准确率。

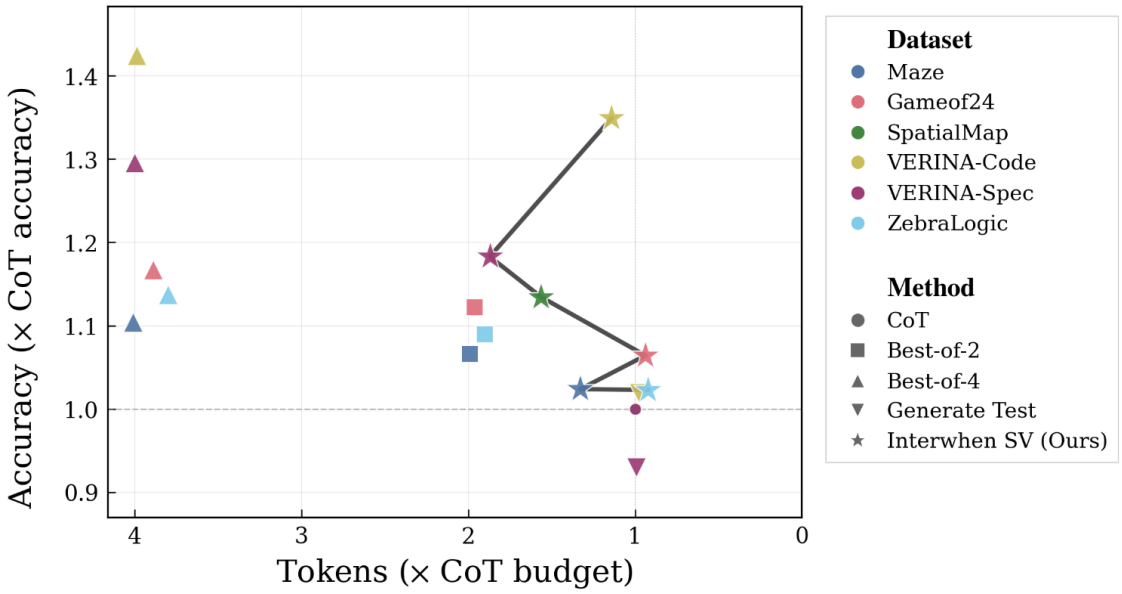


图 5：使用 QwQ-32B 模型绘制的所有数据集的综合结果图。我们绘制由 interwhen 点构成的前沿，其中每个点也位于相应数据集的帕累托前沿上。为清晰起见，我们仅绘制位于相应数据集帕累托前沿上的点。

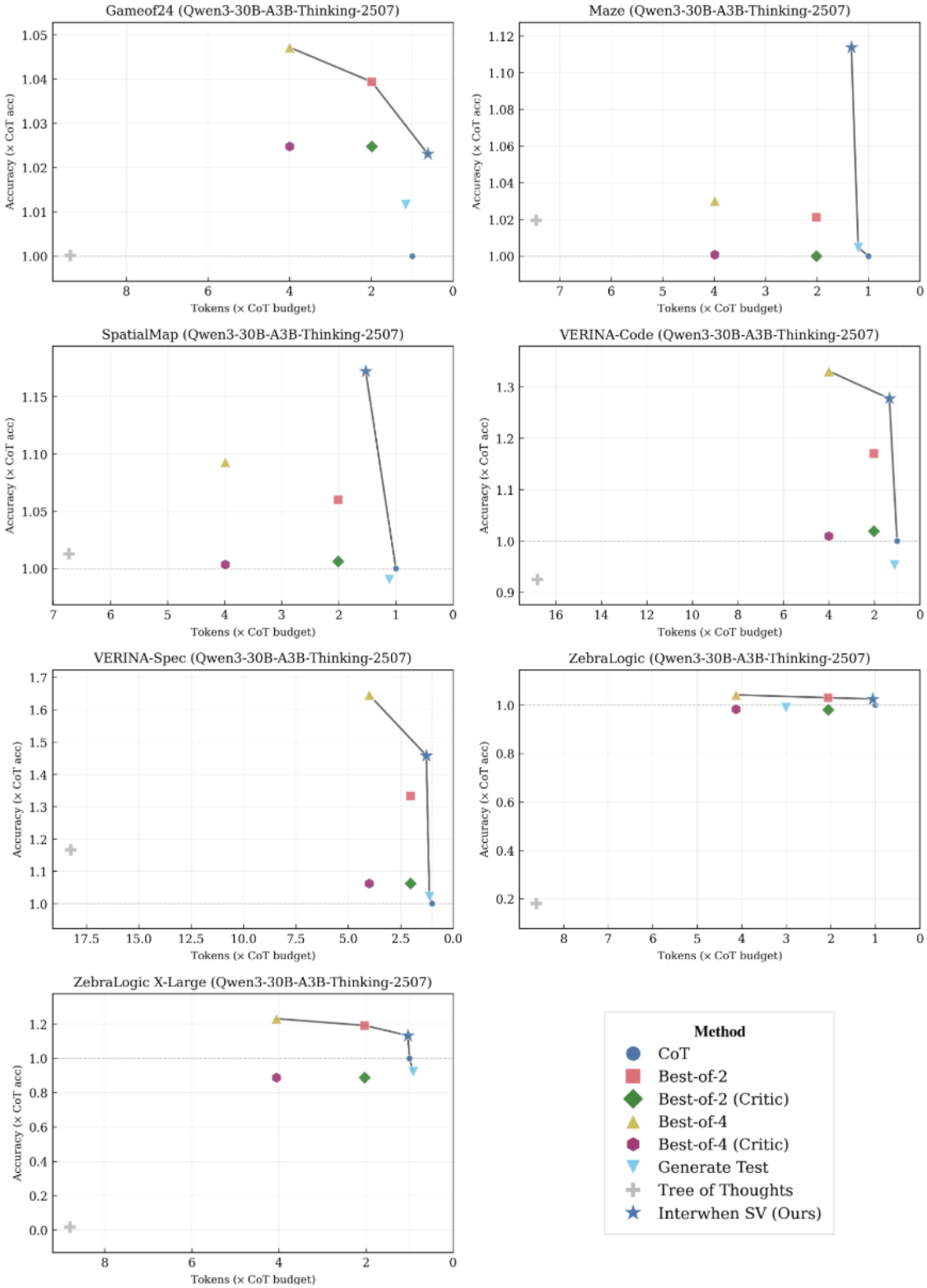


图6：展示使用 Qwen3-30B-A3B-Thinking-2507 模型时各数据集帕累托前沿的图表。值得注意的是，interwhen 在每张图中都位于帕累托前沿上。

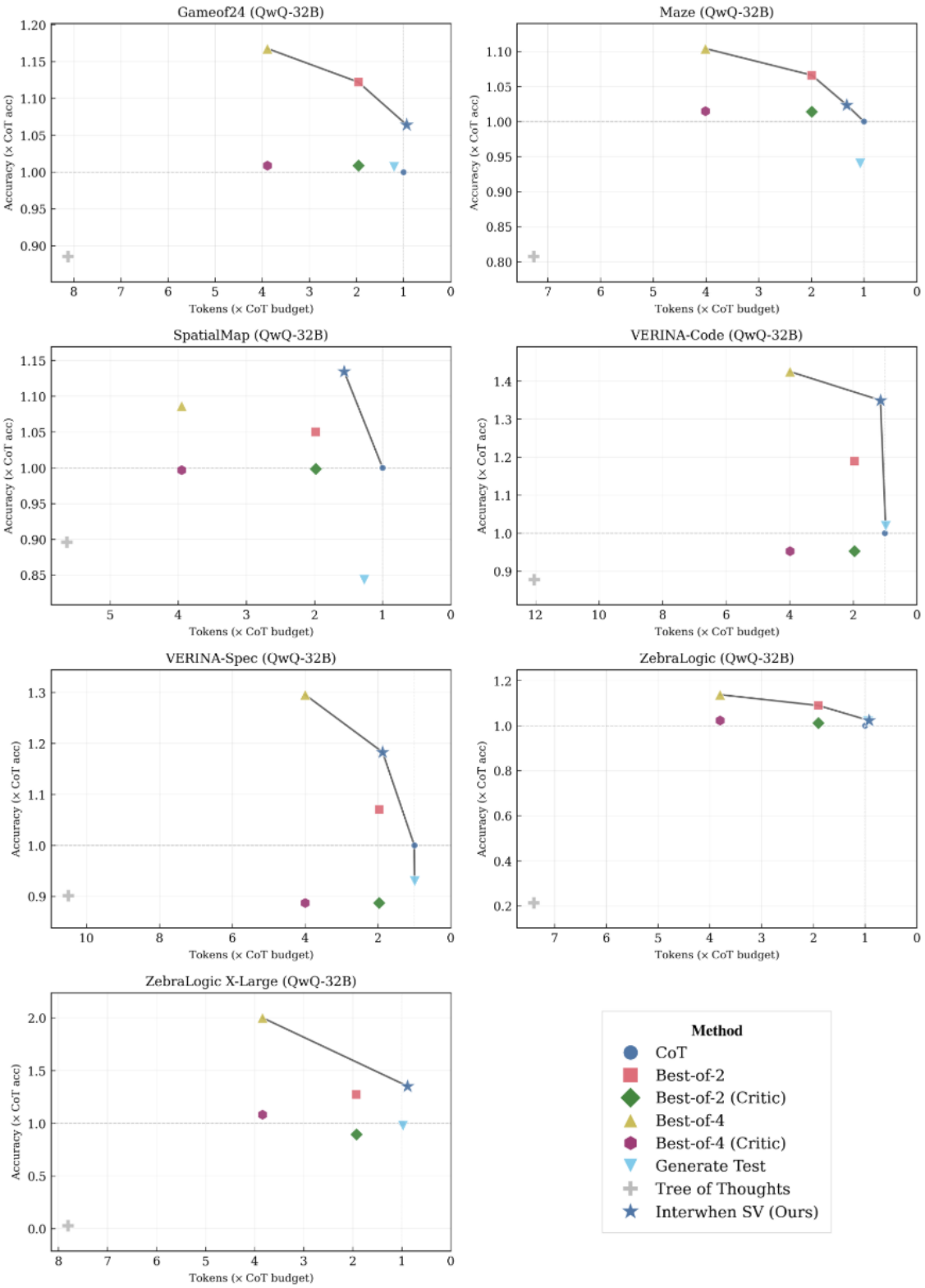


图 7：展示使用 QwQ-32B 模型时各数据集的帕累托前沿的图表。值得注意的是，interwhen 在每张图中都位于帕累托前沿上。

方法值		Maze			SpatialMap			VERINA 代码			VERINA 规范		
		准确率	词元	减少百分比	准确率	词元	减少百分比	准确率	词元	减少百分比	完整规范	词元	减少百分比
基线	0	88.5	4486.3	0.0	74.7	7112.1	0.0	56.1	2881.2	0.0	25.4	3529.1	0.0
EAT	0.2	81.9	1090.0	75.7	71.1	2041.0	71.3	54.5	1208.3	58.1	21.7	1804.1	48.9
	0.1	81.8	1190.5	73.5	71.1	2255.0	68.3	55.0	1574.7	45.4	24.3	2351.0	33.4
	0.04	81.5	1349.0	69.9	71.9	2847.5	60.0	55.6	2095.1	27.3	24.9	3007.3	14.8
	0.008	85.3	3118.5	30.5	73.2	4937.2	30.6	55.6	2790.3	3.2	25.9	3468.9	1.7
	0.005	86.7	3646.5	18.7	73.5	5512.1	22.5	55.6	2821.6	2.1	25.9	3483.7	1.3
	0.003	87.9	4015.6	10.5	73.9	6119.2	14.0	56.1	2836.4	1.6	25.4	3491.4	1.1
	0.001	88.3	4360.4	2.8	74.2	6872.2	3.4	56.1	2881.2	0.0	25.4	3529.1	0.0
	0.0001	88.5	4486.3	0.0	74.6	7083.2	0.4	56.1	2881.2	0.0	25.4	3529.1	0.0
DEER	0.85	83.2	1636.9	63.5	72.3	2708.1	61.9	52.9	2023.4	29.8	22.8	2627.4	25.6
	0.9	84.1	2247.9	49.9	73.3	3395.6	52.3	54.5	2583.2	10.3	22.8	2758.0	21.9
	0.93	85.1	2968.2	33.8	73.7	4112.7	42.2	56.1	2823.4	2.0	23.8	3010.9	14.7
	0.95	87.0	3689.2	17.8	74.3	4873.7	31.5	56.1	2857.4	0.8	24.9	3286.4	6.9
	0.97	88.1	4174.2	7.0	74.6	5992.4	15.7	56.1	2881.2	0.0	25.4	3481.7	1.3
	0.98	88.3	4307.1	4.0	74.9	6526.6	8.2	56.1	2881.2	0.0	25.4	3520.6	0.2
	0.99	88.5	4422.3	1.4	74.8	6856.4	3.6	56.1	2881.2	0.0	25.4	3529.1	0.0
	0.995	88.5	4458.8	0.6	74.9	6987.9	1.7	56.1	2881.2	0.0	25.4	3529.1	0.0
k Stable Answers	2	88.5	3040.1	**32.2**	74.6	5419.2	23.8	52.9	2248.6	22.0	25.4	1584.7	**55.1**
	3	88.7	3337.3	25.6	74.5	5959.8	16.2	55.6	2606.2	9.5	26.5	2539.6	28.0
	4	88.7	3615.6	19.4	75.0	6341.7	**10.8**	56.6	2779.2	**3.5**	26.5	3061.7	13.2
	5	88.6	3850.3	14.2	74.7	6607.0	7.1	56.6	2820.2	2.1	25.9	3358.0	4.9
	7	88.7	4173.0	7.0	74.9	6928.4	2.6	57.1	2873.9	0.3	25.4	3516.9	0.3
	10	88.7	4409.1	1.7	74.8	7068.4	0.6	57.1	2879.6	0.1	25.4	3528.8	0.0
	15	88.7	4486.3	0.0	74.7	7112.1	0.0	57.1	2881.2	0.0	25.4	3529.1	0.0
	100	88.6	4486.3	0.0	74.7	7112.1	0.0	57.1	2881.2	0.0	25.4	3529.1	0.0

表 6：使用 Qwen3-30B-A3B-Thinking-2507 在迷宫、SpatialMap、VERINA 代码和 VERINA 规范上的性能与效率对比。在迷宫数据集上，在保持准确率接近基线的情况下，EAT 和 DEER 仅略微减少了词元（0.6%），而 k-稳定答案在 k=2实现了大幅提升，减少了 32.2%。类似地，在 SpatialMap 上，DEER 在保持性能的同时将词元减少了约 8%，而 k-稳定答案的减少幅度约为 10.8%。对于 VERINA 代码生成，在保持准确率的前提下，EAT（1.6%）和 DEER（2.0%）的减少幅度仍然有限，而 k-稳定答案实现了更高的减少幅度，达到 3.5%。在 VERINA 规范生成方面，EAT 和 DEER 分别将词元减少了 1.7% 和 6.9%，而 k-稳定答案在k=2实现了显著更大的减少幅度，达到 55.1%。

方法值		Maze			SpatialMap			VERINA 代码			VERINA 规范		
		准确率	词元	减少百分比	准确率	词元	减少百分比	准确率	词元	减少百分比	完整规范	词元	减少百分比
基线	0	85.8	5411.4	0.0	74.9	6246.0	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0
EAT	0.2	84.5	1715.5	68.3	73.7	3344.8	46.5	54.5	1070.2	62.9	34.4	1220.3	54.4
	0.1	84.3	2290.0	57.7	73.8	3708.3	40.6	56.1	1469.8	**49.0**	34.4	1223.2	54.3
	0.04	84.7	3253.5	39.9	74.2	4452.2	28.7	58.2	1997.8	30.7	34.4	1319.4	50.7
	0.008	85.4	4928.6	8.9	74.5	5849.2	6.4	56.1	2635.9	8.5	36.5	1910.8	28.5
	0.005	85.7	5196.6	4.0	74.5	6041.4	3.3	56.1	2743.6	4.8	36.5	2148.7	19.7
	0.003	85.6	5339.8	1.3	74.6	6133.3	1.8	56.1	2881.4	0.0	37.6	2457.7	**8.1**
	0.001	85.7	5403.2	0.2	74.9	6224.8	0.3	56.1	2881.4	0.0	37.6	2631.2	1.6
	0.0001	85.7	5403.8	0.1	74.9	6244.8	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0
DEER	0.85	84.4	2829.0	47.7	74.4	4292.3	31.3	57.1	2097.4	**27.2**	36.5	2062.6	22.9
	0.9	85.1	3990.2	26.3	74.8	5398.8	13.6	57.7	2276.2	21.0	37.0	2272.8	15.0
	0.93	85.5	4625.4	14.5	75.0	5845.1	**6.4**	58.2	2469.1	14.3	37.0	2428.1	9.2
	0.95	85.5	4802.2	11.3	75.1	5962.6	4.5	57.1	2573.8	10.7	37.6	2501.3	**6.5**
	0.97	85.5	4905.1	9.4	75.1	6058.8	3.0	56.6	2701.0	6.3	37.6	2563.4	4.1
	0.98	85.5	4970.3	8.2	75.0	6130.0	1.9	56.6	2749.9	4.6	37.6	2593.8	3.0
	0.99	85.5	5098.3	5.8	74.9	6198.9	0.8	56.1	2862.0	0.7	37.6	2666.5	0.3
	0.995	85.7	5214.5	3.6	74.9	6230.9	0.2	56.1	2878.5	0.1	37.6	2674.0	0.0
k Stable Answers	2	85.2	4110.8	24.0	74.1	5367.1	14.1	56.1	2033.6	**29.4**	38.1	2023.3	**24.3**
	3	85.8	4467.3	**17.5**	74.6	5725.6	8.3	56.6	2606.3	9.6	37.6	2463.8	7.9
	4	85.9	4725.6	12.7	74.9	5953.3	**4.7**	58.7	2802.9	2.7	37.6	2633.8	1.5
	5	85.8	4929.7	8.9	74.9	6082.4	2.6	57.7	2855.3	0.9	37.6	2667.1	0.3
	7	85.9	5201.2	3.9	74.9	6190.6	0.9	56.1	2881.4	0.0	37.6	2674.0	0.0
	10	85.8	5365.1	0.9	75.0	6235.8	0.2	56.1	2881.4	0.0	37.6	2674.0	0.0
	15	85.8	5406.9	0.1	74.9	6245.2	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0
	100	85.8	5411.4	0.0	74.9	6246.0	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0

表 7：使用 Qwen/QwQ-32B 在迷宫、SpatialMap、VERINA 代码和 VERINA 规范上的性能–效率比较。在迷宫数据集上，当保持接近基准准确率时，EAT 和 DEER 仅带来有限的词元节省（约 1–4%），而 k-稳定答案在 k=3实现了显著更大的 17.5% 降幅，同时保持了性能。在 SpatialMap 上，EAT 在接近基准准确率时再次仅提供有限节省（低于 1%），而 DEER 实现约 6% 的中等降幅，k-稳定答案则在 k=4以 4.7% 的降幅进一步提升效率。对于 VERINA 代码生成，EAT 在保持准确率的同时实现高达 49% 的大幅降幅，优于最大降幅约为 27% 的 DEER，而 k-稳定答案在 k=2取得了相当的 29.4% 效率提升。最后，在 VERINA 规范生成上，EAT 和 DEER 分别仅带来约 8% 和 6.5% 的相对有限降幅，而 k-稳定答案再次带来最强改进，在 k=2 将推理令牌减少 24.3%，同时保持相当的完整规范正确性。

方法值		Maze			SpatialMap			VERINA 代码			VERINA 规范		
		准确率	词元	减少百分比	准确率	词元	减少百分比	准确率	词元	减少百分比	完整规范	词元	减少百分比
基线	0	86.5	4098.4	0.0	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
EAT	0.2	80.3	387.1	90.6	66.7	964.4	81.0	31.8	323.6	84.3	11.1	384.1	85.8
	0.1	79.7	464.3	88.7	68.3	1643.0	67.6	32.3	358.5	82.7	13.2	549.8	79.7
	0.04	80.4	1380.7	66.3	70.0	3005.9	**40.7**	30.7	639.4	69.1	15.9	1058.8	61.0
	0.008	86.3	3825.9	6.6	69.7	4873.3	3.8	33.3	1509.6	27.0	20.1	2272.7	16.2
	0.005	86.7	3980.5	**2.9**	69.8	4981.8	1.7	33.3	1632.0	21.0	20.1	2439.1	10.1
	0.003	87.0	4053.4	1.1	69.8	5036.6	0.6	32.8	1837.0	11.1	20.6	2567.6	5.3
	0.001	87.1	4084.5	0.3	69.7	5067.1	0.0	33.9	2037.3	**1.4**	20.6	2712.4	0.0
	0.0001	87.1	4092.3	0.2	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
DEER	0.85	86.9	3985.4	**2.8**	70.1	3275.3	**35.4**	32.3	521.1	74.8	20.1	2246.9	17.2
	0.9	87.1	4082.4	0.4	69.9	4733.9	6.6	32.3	727.8	64.8	22.2	2493.4	**8.1**
	0.93	87.0	4090.5	0.2	69.9	4977.3	1.8	34.9	1018.3	**50.7**	22.2	2596.6	4.3
	0.95	87.0	4092.4	0.1	69.7	5029.9	0.7	32.8	1232.4	40.4	20.6	2658.1	2.0
	0.97	86.3	4093.1	0.1	69.7	5059.5	0.1	33.9	1532.4	25.9	20.6	2685.6	1.0
	0.98	86.5	4093.1	0.1	69.7	5066.4	0.0	33.3	1724.8	16.6	20.6	2692.1	0.7
	0.99	86.7	4093.1	0.1	69.7	5067.1	0.0	33.3	1995.4	3.5	20.6	2692.1	0.7
	0.995	86.6	4093.1	0.1	69.7	5067.1	0.0	33.9	2055.8	0.5	20.6	2712.4	0.0
k Stable Answers	2	84.3	3508.9	14.4	69.8	4254.0	**16.1**	32.3	1164.0	43.7	20.6	2647.2	**2.4**
	3	84.7	3762.7	8.2	69.8	4630.5	8.6	33.9	1705.3	**17.5**	20.6	2711.7	0.0
	4	86.5	3898.7	**4.9**	69.8	4848.1	4.3	33.9	1847.5	10.6	20.6	2712.4	0.0
	5	86.7	3987.9	2.7	70.2	4973.0	1.9	33.9	1885.8	8.8	20.6	2712.4	0.0
	7	86.7	4073.1	0.6	69.9	5053.2	0.3	33.9	2066.8	0.0	20.6	2712.4	0.0
	10	86.5	4095.7	0.1	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
	15	86.5	4098.4	0.0	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
	100	86.5	4098.4	0.0	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0

表 8：使用 microsoft/phi-4-reasoning 在 Maze、SpatialMap、VERINA 代码和 VERINA 规范上的性能—效率对比。在 Maze 数据集上，在保持接近基准准确率的情况下，三种方法的令牌节省均较为有限，约为 2–5%；其中，k-稳定答案在 $k{=}4$ 时实现了最大的 4.9% 减少。在 SpatialMap 上，EAT 在接近基准准确率的情况下实现了显著的 40.7% 减少，DEER 同样实现了较大的 35.4% 节省，而 k-稳定答案在 $k{=}2$ 时的减少幅度较为适中，为 16.1%。对于 VERINA 代码生成，DEER 在准确率略有提升的同时实现了最显著的 50.7% 减少，k-稳定答案在 $k{=}3$ 时实现了 17.5% 的减少，而 EAT 在接近基准准确率的情况下仅实现了 1.4% 的边际减少。最后，在 VERINA 规范生成上，DEER 在完整规范正确性提升的同时实现了最强的 8.1 % 改进，EAT 实现了约 5% 的节省，而 k-稳定答案在 $k{=}2$ 时实现了适度的 2.4% 减少。