Test-Time Scaling 相关论文解读

2025年2月19日 4 分钟阅读

#AI #Test-Time Scaling #论文 #技术

本文介绍了Test-Time Scaling（测试时扩展）的概念，并对其技术原理、主要贡献、论文方法、评估结果和局限性进行了详细解读。

Test-Time Scaling介绍

2024年OpenAI发布的新模型Orion，因为性能提升相对有限，从而引发了对Scaling Laws进入边际效应递减阶段的讨论。一部分人认为Scaling Laws的极限尚未到来，而另一些人则认为其已进入边际递减阶段，需要寻找新的方法提升模型性能，例如关注“Scaling What”（扩大正确的规模）而非单纯增加模型参数。

当前预训练扩展法则因高质量数据的稀缺性和计算成本的增加而面临瓶颈，而推理时间计算通过让模型在回答前“思考”更长时间，显著提升复杂推理任务的性能，尤其适用于训练成本高但推理成本较低的场景。同时，合成数据在后训练中的有效性已被验证，不仅能弥补高质量人类数据的不足，还能针对特定任务优化模型表现，从而推动模型性能的持续提升。我认为推理时间计算和合成数据将成为未来进步的主要驱动力，而实现最有效的推理方法将是关键，为强化学习将是实现这一目标的重要途径。

Scaling通常包括：

- 增加模型参数 (Model Scaling)
- 增加训练数据 (Data Scaling)
- 增加训练时间 (Time Scaling)
- 增加测试时计算 (Test-Time Scaling)
- 增加测试时训练 (Test-Time Training)

本文主要关注"Test-Time Scaling"这一方向，并稍微介绍"Test-Time Training"。

目录

文章信息

字数

阅读时间

发布时间

更新时间

标签

#AI #Test-Time Scaling #论

什么是 Test-Time Scaling (TTS)?

定义：TTS 指的是在模型完成训练后，在**推理阶段**（也就是模型实际应用、生成结果的时候）**增加计算资源投入**，从而提高模型性能的一种方法。换句话说，TTS 不是在训练模型时投入更多算力，而是在使用模型的时候，让模型"多思考一会儿"。

目的：TTS 的核心目标是在不改变模型本身参数的情况下，**通过额外的计算来提升模型在特定任务上的表现。**

TTS可以被分为两类:

内部TTS：训练LLM模型，使其通过较长的思考链来缓慢思考。

外部TTS：通过具有固定llm的基于采样或者搜索的方法来提高推理性能。

为什么需要 Test-Time Scaling?

提升推理能力：TTS 能够**增强**大型语言模型（LLM）在复杂任务中的**推理能力**，例如数学问题求解和代码生成。通过让模型进行更深入的思考和探索，可以获得更准确和可靠的结果。

突破模型限制：研究表明，**较小的模型通过 TTS 可以在特定任务上超越更大的模型。**这意味着 TTS 有潜力**挖掘小型模型的潜力**，使其在计算效率上更具优势。例如，一个 10 亿参数的 LLM 在 MATH-500 数据集上可以超过一个 4050 亿参数的 LLM。

与人类思考方式的相似性：TTS 的理念与**人类解决问题的方式**类似。面对难题时，人们通常会投入更多的时间和精力进行思考，而不是立即给出答案。

Test-Time Scaling 的方法

Self-Refinement（自我完善）：模型迭代地改进自身的输出或想法，识别并纠正错误。通过**对语言模型进行微调**，模型能够学会**自我纠正**，从而在推理时**提升准确性。**

Search against a Verifier（基于验证器的搜索）：生成多个候选答案，然后使用一个独立的验证器来选择最佳答案。这种方法依赖于生成多个答案，并使用奖励模型来选择最佳答案，其生成方式可以从**Best-of-N 采样**到 **Monte Carlo 搜索**不等。具体实现方法包括：

Best-of-N Sampling：独立生成 N 个候选答案，然后选择奖励模型评分最高的那个。通过大量的生成，并选择奖励最高的答案来提升准确性。

Beam Search（集束搜索）：逐步生成答案，每一步保留多个最有可能的候选方案（也就是"集束"），并使用奖励模型进行评估，最终选择最佳的完整答案。集束搜索可以更好地**指导复杂推理任务**，通过奖励模型评估中间步骤的正确性，从而优先选择有希望的路径。

Diverse Verifier Tree Search (DVTS)：对集束搜索进行扩展，将搜索过程分成多个独立的子树，每个子树独立进行集束搜索，从而增加搜索的多样性。DVTS 旨在解决集束搜索中可能出现的**树坍塌问题**，通过**探索多个不同的路径**来**提高性能。**

Adaptive Graph of Thoughts (AGoT)：一种动态的、基于图的推理框架，它将复杂的查询分解为结构化的子问题，形成一个动态的有向无环图。AGoT 选择性地扩展那些需要进一步分析的子问题，将链、树和图的优势统一到一个有凝聚力的框架中，并将计算分配到最需要的地方。

修改提议分布：这种方法主要受到**强化学习**技术的启发，模型逐步完善自身的输出。类似于 **STAR**（自学推理器）或**强化自训练**，模型使用额外的计算来纠正或改进先前的输出，耐心地选择要生成的 token，并在答案不佳时回溯。

Search-o1：模型在检测到知识缺失时暂停推理，构建搜索查询，获取文档，并整合检索到的信息后再继续。

预算强制：通过使用"Wait"和"Final Answer"等标签来调整模型的推理时间，从而控制模型思考的时间。

此外，还有一些其他的 TTS 实现方式，例如：

NVIDIA 的 SANA-1.5：通过**增加** SANA-1.5 生成的**图像数量**，然后使用 AI"法官"（NVILA-2B 模型）基于排名进行选择，从而创建更准确的图像。

集体蒙特卡洛树搜索 (CoMCTS)：通过使用高级测试时间验证器来进行多模态模型的测试时间缩放。

迭代细化：对模型进行微调，使其能够迭代地改进答案。这意味着将问题和模型给出的答案输入到模型中，让模型给出改进后的答案，重复此过程以获得更好的答案。

总结来说，TTS 的实现方式多种多样，选择哪种方式取决于具体的任务和模型。一般来说，需要考虑以下因素：问题的难度、计算预算、以及对奖励模型质量的信任程度。

奖励模型 (Reward Model) 在 Test-Time Scaling 中的作用

评估候选答案：**奖励模型**（或称作"验证器"）在 TTS 中扮演着至关重要的角色。它**评估**模型生成的**候选答案**，并给出相应的**分数**，用于**指导搜索过程。**

类型：奖励模型可以分为两种主要类型：

Outcome Reward Model（结果奖励模型）：**评估最终答案的正确性。**

Process Reward Model（过程奖励模型）：**评估解决问题的中间步骤的质量**，例如每一步推理是否合理。

Test-Time Scaling 的优势

提高小模型的竞争力：TTS 使**较小的模型**能够**在特定任务上与更大的模型竞争甚至超越它们。**

更有效地利用计算资源：通过在**推理时**投入**更多算力**，而不是一味地扩大模型规模，可以**更有效地利用计算资源**，降低成本。

潜在的应用前景：TTS 为构建**通用自提升智能体**开辟了新的途径，这些智能体可以在开放式的自然语言环境中运行。

与预训练的互补性：TTS 可以与预训练相结合，在某些情况下，**对小模型使用额外的测试时计算**可能**比使用标准推理计算的大模型更有效。**

Test-Time Scaling 的局限性

对奖励模型的依赖：TTS 的性能高度依赖于奖励模型的质量。如果奖励模型存在偏差或漏洞，可能会导致模型在搜索过程中做出错误的选择。

计算成本：虽然 TTS 可以在一定程度上降低对超大模型的依赖，但**额外的推理计算仍然会带来成本。**尤其是在需要进行大量采样和验证的情况下，计算成本可能会非常高昂。

泛化能力：在某个特定数据集或任务上表现良好的 TTS 策略，可能**无法**很好地**泛化到其他领域。**

其他限制:

可能在不同的想法之间跳跃太快，过早放弃有希望的想法。

简单查询的速度很快，而复杂查询可能需要更长的时间，这在实时应用中可能会出现问题。

某些查询可能会获得比需要的更多的计算，从而导致效率低下，而另一些查询可能会获得的计算更少，从而导致次优答案。

由于系统负载或模型启发式等外部因素，同一查询在不同情况下可能会获得不同级别的计算，这可能会导致不一致的输出。

每个查询的成本各不相同，这使得难以对具有高度可变查询的用户进行预算。

总而言之，Test-Time Scaling 是一种很有前途的技术，它通过在推理阶段投入更多算力来提高语言模型的性能。虽然 TTS 存在一些局限性，但随着研究的不断深入和计算成本的降低，它有望在未来发挥更大的作用，推动人工智能技术的发展。

Self-Refinement（自我完善）

Self-Refinement（自我完善）是一种**测试时计算**（Test-Time Compute, TTC）的策略，旨在**通过迭代的方式改进模型自身的输出。**这种方法让模型在生成初始答案后，**不是直接给出最终结果，**

而是对自身的答案进行评估和改进，重复这个过程直到满足某个停止条件。

以下是 Self-Refinement 的关键步骤和特点：

生成初始答案：

模型首先**根据输入的问题或任务**，生成一个初始的答案或解决方案。

自我评估：

模型**对生成的答案进行评估**，判断其**正确性、完整性、流畅性**等方面。

这个评估过程可以**由模型自身完成**，也可以**借助一个单独的验证模型**（Verifier Model）。

在自我评估中，模型可能会**识别出答案中的错误、不完整或不清晰之处。**

迭代改进：

基于自我评估的结果，模型**对初始答案进行改进**，尝试**修正错误、补充信息或优化表达。**

这个改进过程可以**重复多次**，每次迭代都**基于上一次的结果进行优化。**

在每次迭代中，模型可能会**生成多个候选的改进方案**，并**选择其中最佳的一个。**

停止条件：

Self-Refinement 的过程**需要一个停止条件**，以**避免无限迭代。**

停止条件可以是**达到最大迭代次数**、**答案的评估分数超过某个阈值**或**答案在多次迭代后没有明显改进**等。

具体实现方法：

Prompt工程：通过精心设计的 Prompt，引导模型进行自我评估和改进。 例如，可以在 Prompt 中加入 “Take a deep breath and think through it step by step” 等指令。

多轮对话：将单轮问题转化为多轮对话，让模型在对话中逐步完善答案。

强化学习：使用强化学习（Reinforcement Learning, RL）训练模型，使其学会自我完善的策略。

例如，可以**定义一个奖励函数**，**奖励模型的改进行为**，**惩罚模型的错误行为。**

通过 RL，模型可以**学习到如何有效地进行自我评估和改进**，从而提高最终答案的质量。

监督式微调：使用**包含自我完善过程的数据集**，对模型进行微调。

例如，可以**收集人类专家解决问题的过程，将其转化为模型可以学习的示例。**

通过监督式微调，模型可以**模仿人类专家的自我完善行为**，从而提高自身的推理能力.

与其他技术的结合：

Chain of Thought (CoT)：Self-Refinement 可以与 CoT 结合使用，让模型在**生成答案的同时，输出思考过程**，从而**更好地进行自我评估和改进。**

Search：Self-Refinement 可以与**搜索算法**结合使用，让模型**在多个候选答案中进行选择和改进**，从而**提高找到最佳答案的概率。**

优点：

提高准确性：通过迭代改进，可以**减少错误**，**提高答案的准确性。**

增强鲁棒性：通过自我评估，可以**发现并纠正答案中的不足**，**增强模型的鲁棒性。**

提高可解释性：通过输出思考过程，可以**提高模型的可解释性**，**让人们更好地理解模型的决策过程。**

缺点：

计算成本高：Self-Refinement 需要进行多次迭代，因此**计算成本相对较高。**

可能陷入局部最优：如果自我评估不准确，模型可能会陷入局部最优，无法找到全局最佳答案。

总之，Self-Refinement 是一种**有效的测试时计算策略**，它通过**模拟人类的思考和改进过程**，**提高模型的推理能力和答案质量。**尽管存在一些挑战，但随着计算资源的不断提升和算法的不断优化，Self-Refinement 有望在未来得到更广泛的应用。

Search against a Verifier（基于验证器的搜索

“Search against a Verifier (基于验证器的搜索)” 是一种**测试时计算（Test-Time Compute, TTC）策略，它通过生成多个候选答案**，并**使用一个独立的验证器（Verifier）来选择最佳答案**，以此来提高模型性能。

以下是 “Search against a Verifier” 的关键步骤和特点：

生成候选答案 (Generating Candidate Responses)：

Proposer 模型：使用一个大型语言模型（LLM）作为 “Proposer”，负责**生成多个**可能的**候选答案。**

抽样方法：可以使用不同的抽样方法来生成这些候选答案，例如 **Best-of-N sampling (最佳N选一抽样)**，**Monte Carlo 树搜索** 或其他搜索算法。

验证与评分 (Verification and Scoring)：

Verifier 模型：使用一个独立的模型作为 “Verifier”，负责**评估每个候选答案的质量。**

奖励模型 (Reward Model)：Verifier 通常是一个**奖励模型**，它**根据一定的标准**（例如，正确性、简洁性、相关性、流畅性等） **对每个候选答案进行评分。**奖励模型可以基于不同的监督方式进行训练，例如结果监督（Outcome-Supervised）或过程监督（Process-Supervised）。

结果监督奖励模型 (Outcome-Supervised Reward Model, ORM)：**评估 LLM 的完整生成结果。**

过程监督奖励模型 (Process-Supervised Reward Model, PRM)：**逐步评估 LLM 生成的每个步骤。**

自我验证 (Self-Verification)：在某些情况下，Proposer 模型本身也可以承担 Verifier 的角色，**对自身生成的候选答案进行评估。**

选择最佳答案 (Selecting the Best Answer)：

基于 Verifier 的评分，选择得分最高的候选答案作为最终输出。

加权：对多个答案进行抽样，然后**使用验证器来决定哪个答案是最好的**，对其进行加权聚合。

关键组成部分：

Proposer：是一个 LLM，可生成多个备选完成项。

Verifier：是另一个模型，用于对每个完成项进行评分并选择最佳完成项。

优点：

提高准确性：通过在多个候选答案中选择，可以**提高最终答案的准确性。**

利用现有知识：**主要提取现有知识并完善它，而不是获得新知识。**

可以减少模型大小：可以使用较小的模型**获得与较大模型相当的准确度。**例如，通过使用 ORM 方法，6B 模型可以胜过 175B 非 ORM 模型。

缺点：

计算成本高：需要生成和评估多个候选答案，因此**计算成本相对较高。**

依赖 Verifier 的质量：最终答案的质量**很大程度上取决于 Verifier 的准确性和可靠性。**

过度检查会适得其反。

与其他技术的结合：

Chain of Thought (CoT)：可以与 CoT 结合使用，**对 CoT 的推理过程进行验证**，选择最佳的推理路径。

Beam Search (束搜索)：一种**优化 PRM 的方法**，通过搜索其每步预测。该过程是**抽样初始预测，然后对生成的步骤进行评分，然后过滤得分最高的步骤，然后为每个候选步骤设计后续步骤，然后重复此过程。**

Lookahead Search (前瞻搜索)：**修改了束搜索评估各个步骤的方式。**它使用前瞻性展开来提高 PRM 在搜索过程的每个步骤中的价值评估的准确性。

多样性验证树搜索（Diverse Verifier Tree Search, DVTS）

通过将搜索过程划分为 N/M 个子树来扩展 Beam Search，其中每个子树都使用 Beam Search 独立探索，从而提高多样性。

总之，“Search against a Verifier” 是一种**有效的测试时计算策略**，它通过**引入一个独立的 Verifier 来评估和选择候选答案，提高模型的准确性和可靠性。** 尽管计算成本较高，但通过**优化抽样方法**和**提高 Verifier 的质量**，可以有效地**降低成本**并**提高性能。**

测试时训练（TTT: Test-Time Training）

根据麻省理工学院（MIT）的论文，**Test-Time Training (TTT)** 是一种**参数模型在推理过程中通过动态参数更新进行适应的方法。**这种技术是一种**转导学习**的形式，模型利用测试数据的结构来改进其预测。TTT 仍然是大型语言模型时代中相对未被探索的方法。 这种技术是 24 年 11 月份由 MIT 提出的另一条实现大模型Scaling Law的路线。它不同于标准的微调，因为它在极低数据的情况下运行，通常对单个输入或一两个上下文中的标记示例使用无监督或监督目标。相当于对推理过程中的数据进行调整后合成测试时训练数据用

来更新模型的参数，这种方法对抽象推理的问题效果较好，MIT 团队在 Llama38B 模型上使用这种方法后，相比于 1B 的基础微调模型，准确率提高了 6 倍；在 8B 参数的语言模型上应用 TTT，在 ARC 公共验证集上实现了 45% 的准确率，比 8B 基础模型提高了近 157%。但是该方法仍在初期试验阶段，对计算资源要求也很高，所以论文的评估主要在 ARC 公共验证集的一个子集上进行，并没有提交到官方排行榜。

TTS, TTC, TTT的关系

Test-Time Scaling (TTS)、**Test-Time Compute (TTC)** 和 **Test-Time Training (TTT)** 之间的关系可以概括如下：

> **TTS (测试时缩放)** 是一种**在模型推理阶段**，通过**增加计算资源或优化推理策略**来提升模型性能的方法。TTS 的目标是在不改变模型本身参数的情况下，尽可能地**挖掘模型的潜力**，提高其在特定任务上的准确性和可靠性。
>
> **TTC (测试时计算)** 是 TTS 的**核心资源**，指的是在推理过程中**用于生成输出的计算量**。TTS 通过**调整 TTC 的分配方式**，例如增加采样次数、进行更复杂的搜索或进行迭代优化，来实现性能的提升。
>
> **TTT (测试时训练)** 是一种**更高级的技术**，它不仅利用测试时的计算资源，还**允许模型在测试阶段进行参数更新和学习**，从而更好地适应新的数据和环境。TTT 可以看作是 TTS 的一种**未来发展方向**。

具体来说，它们之间的关系可以类比为：

> **TTC 是基础**：指的是推理时使用的计算资源，类似于“燃料”。
>
> **TTS 是策略**：利用 TTC 来提升模型性能，例如通过让模型进行更长时间的思考或进行更复杂的搜索，类似于“驾驶技术”。
>
> **TTT 是展望**：是一种在测试阶段继续训练模型的新兴技术，可以看作是 TTS 的一种高级形式，类似于“自动驾驶系统”。

从这个角度来看，TTC 是 TTS 和 TTT 的共同基础，而 TTS 和 TTT 则是在 TTC 的基础上，通过不同的方法来实现模型性能的提升。TTS 侧重于在推理过程中利用额外的计算资源进行搜索和验证，而 TTT 则侧重于在测试阶段通过持续学习来适应新的数据和环境。

总结：

> **TTS 是一种利用额外计算资源提升推理性能的策略。**
>
> **TTT 是一种在测试阶段持续训练模型以适应新数据和环境的技术。**

TTC 是 TTS 和 TTT 的共同基础，是推理过程中使用的计算资源。

TTT 可以看作是 TTS 的一种**高级形式和未来发展方向。**

相关论文和博客

Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach (2502.05171)

Generating Symbolic World Models via Test-time Scaling of Large Language Models (2502.04728)

Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling (2502.06703)

Llasa: Scaling Train-Time and Inference-Time Compute for Llama-based Speech Synthesis (2502.04128)

Rethinking Fine-Tuning when Scaling Test-Time Compute: Limiting Confidence Improves Mathematical Reasoning (2502.07154)

Adaptive Graph of Thoughts: Test-Time Adaptive Reasoning Unifying Chain, Tree, and Graph Structures (2502.05078)

Sample, Scrutinize and Scale: Effective Inference-Time Search by Scaling Verification (2502.01839)

CodeMonkeys: Scaling Test-Time Compute for Software Engineering (2501.14723)

OpenAI: Scaling Laws for Neural Language Models

DeepMind: Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

MIT: The Surprising Effectiveness of Test-Time Training for Abstract Reasoning

OpenAI o1 System Card by OpenAI

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning by DeepSeek

Virgo: A Preliminary Exploration on Reproducing o1-like MLLM by [BAAI](Baichuan AI.), Gaoling School of Artificial Intelligence, Renmin University of China and Baichuan AI

Mulberry: Empowering MLLM with o1-like Reasoning and Reflection via Collective Monte Carlo Tree Search

Can We Generate Images with CoT? Let's Verify and Reinforce Image Generation Step by Step by CUHK MiuLar Lab & 2MMLab, Peking University, Shanghai AI Lab,

SANA 1.5: Efficient Scaling of Training-Time and Inference-Time Compute in Linear Diffusion Transformer by NVIDIA

s1: Simple test-time scaling by Stanford, University of Washington, Allen AI, and Contextual AI

Thoughts Are All Over the Place: On the Underthinking of o1-Like LLMs

o1-Coder: an o1 Replication for Coding

Test-time Computing: from System-1 Thinking to System-2 Thinking

O1 Replication Journey: A Strategic Progress Report – Part 1 by NYU, Shanghai Jiao Tong University, MBZUAI and GAIR

O1 Replication Journey – Part 2: Surpassing O1-preview through Simple Distillation, Big Progress or Bitter Lesson?

O1 Replication Journey – Part 3: Inference-time Scaling for Medical Reasoning

What is test-time compute and how to scale it?

LLM Test-Time Compute

> **DeepSeek 微调**

本文介绍了如何使用合成推理...

> **DeepSeek R1 论文解读**

本文介绍了深度求 ...