O1-CODER: AN O1 REPLICATION FOR CODING

Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong & Jitao Sang*
School of Computer Science and Technology
Beijing Jiaotong University
Beijing, China
{yuxiangzhang, wushangxi, yqyang, jiangmingshu, jinlinx, 23120361, jtsang}@bjtu.edu.cn

ABSTRACT

The technical report introduces O1-CODER, an attempt to replicate OpenAI's o1 model with a focus on coding tasks. It integrates reinforcement learning (RL) and Monte Carlo Tree Search (MCTS) to enhance the model's System-2 thinking capabilities. The framework includes training a Test Case Generator (TCG) for standardized code testing, using MCTS to generate code data with reasoning processes, and iteratively fine-tuning the policy model to initially produce pseudocode and then generate the full code. The report also addresses the opportunities and challenges in deploying o1-like models in real-world applications, suggesting transitioning to the System-2 paradigm and highlighting the imperative for world model construction. Updated model progress and experimental results will be reported in subsequent versions. All source code, curated datasets, as well as the derived models are disclosed at https://github.com/ADaM-BJTU/O1-CODER.

1 INTRODUCTION

OpenAI recently introduced the o1 model (OpenAI, 2024), which has demonstrated impressive system-2 thinking capabilities. This model represents a significant advancement in AI's ability to perform complex reasoning tasks that require higher-order cognitive functions. Following its release, numerous analysis and replication efforts have emerged, demonstrating the growing interest in reasoning models. Notable works include g1 (Benjamin Klieger, 2024), OpenO1 (ope, 2024), O1-Journey (GAIR-NLP, 2024), OpenR (Team, 2024), LLaMA-O1 (SimpleBerry, 2024), LLaMA-Berry (Zhang et al., 2024), Steiner (Ji, 2024), Thinking Claude (Richards Tu, 2024), LLaVA-o1 (Xu et al., 2024), and several industrial releases such as k0-math, DeepSeek-R1-Lite, Macro-o1 (Zhao et al., 2024), Skywork o1, QwQ (Qwen Team, 2024), and InternThinker (Shanghai AI Lab, 2024) (illustrated in Fig. 1).

Prior to the o1 model, large language models (LLMs) primarily exhibited System-1 capabilities, characterized by fast, intuitive responses. These models were trained on datasets consisting mainly of question-answer (Q, A) pairs, lacking the intermediate reasoning steps that involve deliberate and analytical processing. This stems from the fact that humans rarely record their thought processes on the internet or elsewhere. Traditionally, techniques such as Chain-of-Thought (CoT) prompting were

*Corresponding author.

O1-CODER: AN O1 REPLICATION FOR CODING

Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong & Jitao Sang*
School of Computer Science and Technology
Beijing Jiaotong University
Beijing, China
{yuxiangzhang, wushangxi, yqyang, jiangmingshu, jinlinx, 23120361, jtsang}@bjtu.edu.cn

ABSTRACT

*警告：该PDF由GPT-Academic开源项目调用大语言模型+Latex翻译插件一键生成，版权归原文作者所有。翻译内容可靠性无保障，请仔细鉴别并以原文为准。项目Github地址 https://github.com/binary-husky/gpt_academic/。项目在线体验地址 https://auth.gpt-academic.top/。当前大语言模型: Qwen2.5-72B-Instruct，当前语言模型温度设定: 0.3。为了防止大语言模型的意外谬误产生扩散影响，禁止移除或修改此警告。

技术报告介绍了O1-CODER，这是尝试复制OpenAI的o1模型并专注于编码任务的一项工作。它集成了强化学习（RL）和蒙特卡洛树搜索（MCTS）以增强模型的系统2思维能力。该框架包括训练一个测试用例生成器（TCG）以进行标准化代码测试，使用MCTS生成带有推理过程的代码数据，并迭代微调策略模型，以首先生成伪代码，然后生成完整代码。报告还讨论了在实际应用中部署类似o1模型的机会和挑战，建议转向系统2范式，并强调构建世界模型的重要性。更新的模型进展和实验结果将在后续版本中报告。所有源代码、策划的数据集以及衍生模型均在https://github.com/ADaM-BJTU/O1-CODER公开。

1 INTRODUCTION

OpenAI 最近推出了 o1 模型 (OpenAI, 2024)，该模型展示了令人印象深刻的系统2思维能力。这一模型代表了人工智能在执行需要高级认知功能的复杂推理任务方面的重要进展。自发布以来，出现了许多分析和复制努力，表明了对推理模型日益增长的兴趣。值得注意的作品包括 g1 (Benjamin Klieger, 2024)，OpenO1 (ope, 2024)，O1-Journey (GAIR-NLP, 2024)，OpenR (Team, 2024)，LLaMA-O1 (SimpleBerry, 2024)，LLaMA-Berry (Zhang et al., 2024)，Steiner (Ji, 2024)，Thinking Claude (Richards Tu, 2024)，LLaVA-o1 (Xu et al., 2024)，以及一些工业发布，如 k0-math，DeepSeek-R1-Lite，Macro-o1 (Zhao et al., 2024)，Skywork o1，QwQ (Qwen Team, 2024)，和 InternThinker (Shanghai AI Lab, 2024)（如图 1所示）。

在 o1 模型之前，大型语言模型（LLMs）主要表现出系统1的能力，其特点是快速、直观的响应。这些模型主要在由问题-答案 (Q, A) 对组成的数据集上进行训练，缺乏涉及深思熟虑和分析处理的中间推理步骤。这源于人类很少在互联网或其他地方记录他们的思维过程。传统上，使用诸如链式思维（CoT）提示等技术来引导模型生成逐步推理，最终得出答案。一种更直接和有效的方法是创建包含推理序列的数据集，例如 $(Q, ..., S_i, ..., A)$，其中

*Corresponding author.

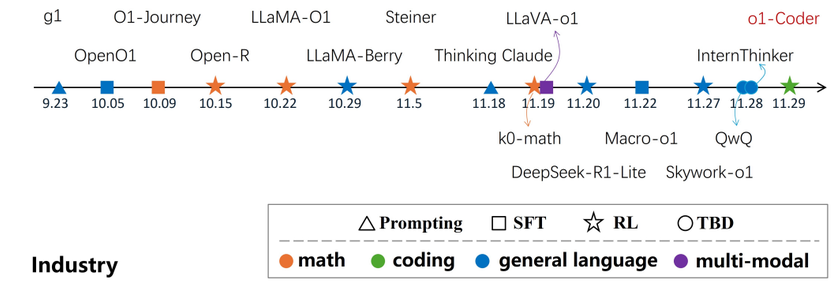

图 1: o1 replication efforts: upper part from academic institutions and open-source communities, and lower part from the industry.

used to guide models in generating step-by-step reasoning before arriving at an answer. A more direct and effective way is to create datasets including the reasoning sequences, e.g., $(Q, ..., S_i, ..., A)$, where S_i represents an individual reasoning step leading to the final answer. Thus, a second approach to enhancing System-2 reasoning capabilities is supervised fine-tuning (SFT). However, most publicly available data is recorded in a question-answer form, and annotating or distilling such data, especially for complex tasks, is both costly and challenging. In this study, we aim to explore in the absence of reasoning process data, and thus opt for the third approach of reinforcement learning (RL).

It is widely believed that o1 addresses the lack of reasoning data by combining reinforcement learning with pretraining. Reinforcement learning is well known for its ability to explore and discover new strategies rather than relying on predefined data in the past decade. Looking back at key developments in machine learning, we can see that deep learning and large-scale pretraining have driven transformations in model architecture and the requirements for labeled data, respectively. In contrast, reinforcement learning addresses a different aspect of transformation on the objective function. In situations where explicit guidance or clear goals are absent, RL exploits exploration to search for new knowledge and solutions. Therefore, combining pretraining with RL creates a powerful synergy of learning and search, where pretraining compresses existing human knowledge, and RL enables the model to explore new possibilities.

We chose coding tasks to explore how to employ RL to generate and refine reasoning data. Coding is a typical task that requires System-2 thinking, involving careful, logical, and step-by-step problem-solving. Moreover, coding can serve as a foundational skill for solving many other complex problems. This report presents our attempt to replicate o1 with a specific focus on coding tasks. The approach integrates RL and Monte Carlo Tree Search (MCTS) to enable self-play, allowing the model to continually generate reasoning data and enhance its System-2 capabilities.

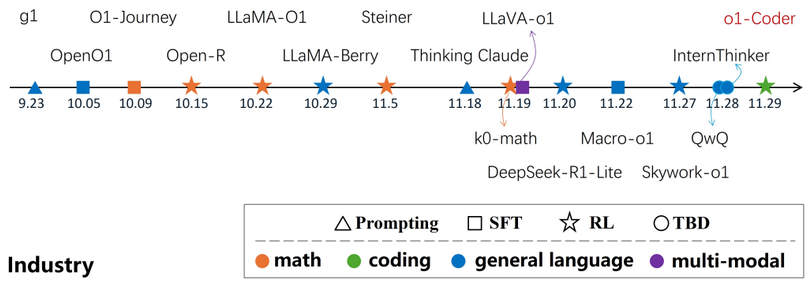

图 1: o1 复制努力：上部分来自学术机构和开源社区，下部分来自行业。

S_i 代表最终答案前的单个推理步骤。因此，增强系统2推理能力的第二种方法是监督微调（SFT）。然而，大多数公开可用的数据是以问题-答案的形式记录的，标注或提炼这类数据，尤其是对于复杂任务，既昂贵又具有挑战性。在本研究中，我们旨在探索在缺乏推理过程数据的情况下，因此选择强化学习（RL）作为第三种方法。

普遍认为，o1 通过结合强化学习与预训练解决了推理数据的缺乏问题。强化学习以其探索和发现新策略的能力而闻名，而不仅仅是依赖过去预定义的数据。回顾机器学习的关键发展，我们可以看到，深度学习和大规模预训练分别推动了模型架构和标注数据需求的变革。相比之下，强化学习在目标函数方面带来了不同的变革。在缺乏明确指导或明确目标的情况下，RL 利用探索来搜索新知识和解决方案。因此，将预训练与 RL 结合起来，创造了一种强大的学习和搜索协同效应，其中预训练压缩了现有的人类知识，而 RL 使模型能够探索新的可能性。

我们选择了编程任务来探索如何利用 RL 生成和优化推理数据。编程是一个典型的需要系统2思维的任务，涉及仔细、逻辑性和逐步的问题解决。此外，编程可以作为解决许多其他复杂问题的基础技能。本报告展示了我们尝试复制 o1，特别关注编程任务。该方法集成了 RL 和蒙特卡洛树搜索（MCTS），以实现自我对弈，使模型能够不断生成推理数据并增强其系统2能力。

2 Framework Overview

将自博弈强化学习（RL）应用于代码生成存在两个主要挑战。第一个挑战是结果评估，即评估生成代码的质量。与围棋或数学等任务不同，这些任务的结果可以直接根据游戏规则或正确答案进行评估，而代码评估则需要在测试环境中运行生成的代码，并通过测试用例进行验证。我们不能假设代码数据集总是会提供足够的测试用例。第二个挑战涉及定义思考和搜索行为，即确定状态转换和过程奖励的粒度。对于代码生成，关键问题是如何设计推理过程和策略空间，以有效指导模型的行为。

2 Framework Overview

There are two main challenges to address for applying self-play RL to code generation. The first challenge is result evaluation, i.e., assessing the quality of the generated code. Unlike tasks such as Go or mathematics, where results can be directly evaluated based on game rules or correct answers, evaluating code requires running the generated code within a testing environment and verifying it against test cases. We cannot assume that code datasets will always provide sufficient test cases. The second challenge involves defining the thinking and search behaviors, i.e., determining the state transition and the granularity of process rewards. For code generation, the key question is how to design the reasoning process and the space of policies to guide the model's behavior effectively.

To address the first challenge, we propose training a Test Case Generator (TCG), which automatically generates test cases based on the question and the ground-truth code [1]. This approach will help build a standardized code testing environment, providing result rewards for reinforcement learning.

For the second challenge, two possible approaches can be considered. One is "think before acting", where the model first forms a complete chain-of-thought and then generates the final answer all at once. The other approach, "think while acting" (Zelikman et al., 2024), involves generating parts of the answer while simultaneously reasoning through the task. We chose the former approach in this study. For code generation, this means first thinking through and writing out a detailed pseudocode, which is then used to generate the final executable code. The advantages are two-fold: adaptability, as the same pseudocode can lead to different concrete code implementations; and controllable granularity, as adjusting the level of detail in the pseudocode can control the granularity of the reasoning/search behavior.

The outlined framework is provided in Algorithm 1, which consists of six steps. (1) The first step is training the test case generator (TCG) γ_{TCG}, which is responsible for automatically generating test cases based on the question. (2) In the second step, we run MCTS on the original code dataset to generate code dataset with reasoning processes $\mathcal{D}_{\text{process}}$, including a validity indicator to distinguish between correct and incorrect reasoning steps. (3) Once we have data that includes the reasoning process, the third step is to fine-tune the policy model π_θ, training it to behave in a "think before acting" manner. (4) The reasoning process data can also be used to initialize the process reward model (PRM) ρ_{PRM}, which evaluates the quality of reasoning steps. (5) The fifth step is the most crucial: with PRM ρ_{PRM} providing process rewards and TCG γ_{TCG} providing outcome rewards, the policy model π_θ is updated with reinforcement learning. (6) In the 6th step, based on the updated policy model, new reasoning data can be generated. This new data can then be used to fine-tune the PRM again (4th step). Therefore, steps 4, 5, and 6 form an iterative cycle, where self-play continues to drive model improvements. The flow between the six steps is illustrated in Fig. 2. The following section will introduce each step in detail.

[1] We also propose an alternative approach where test cases are generated based solely on the question. In addition to be capable of utilizing code datasets that only provide questions, it can also be applied during the inference phase, enabling online reasoning without the need for ground-truth code.

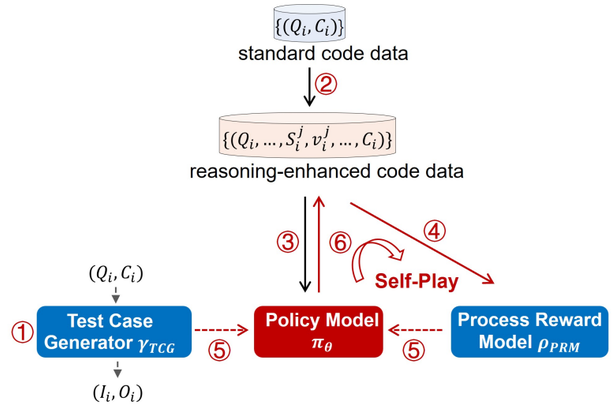

图 2: Self-Play+RL training framework.

为了解决第一个挑战，我们提出训练一个测试用例生成器（TCG），该生成器能够根据问题和真实代码自动生成测试用例 [1]。这种方法将有助于构建标准化的代码测试环境，为强化学习提供结果奖励。

对于第二个挑战，可以考虑两种可能的方法。一种是“先思考再行动”，即模型首先形成一个完整的思考链，然后一次性生成最终答案。另一种方法是“边思考边行动” (Zelikman et al., 2024)，即在生成答案的部分同时进行任务的推理。在本研究中，我们选择了前者。对于代码生成，这意味着首先思考并写出详细的伪代码，然后用其生成最终的可执行代码。这种方法的优势有两方面：适应性，相同的伪代码可以导致不同的具体代码实现；可控的粒度，通过调整伪代码的详细程度可以控制推理/搜索行为的粒度。

所提出的框架在算法 1中概述，包含六个步骤。（1）第一步是训练测试用例生成器（TCG）γ_{TCG}，其负责根据问题自动生成测试用例。（2）第二步是在原始代码数据集上运行MCTS，生成包含推理过程的代码数据集$\mathcal{D}_{\text{process}}$，包括一个有效性指示器，用于区分正确的和错误的推理步骤。（3）一旦我们有了包含推理过程的数据，第三步是微调策略模型π_θ，训练其以“先思考再行动”的方式行为。（4）推理过程数据还可以用于初始化过程奖励模型（PRM）ρ_{PRM}，该模型评估推理步骤的质量。（5）第五步是最重要的：在PRM ρ_{PRM}提供过程奖励和TCG γ_{TCG}提供结果奖励的情况下，策略模型π_θ通过强化学习进行更新。（6）在第六步，根据更新的策略模型生成新的推理数据。这些新数据可以再次用于微调PRM（第四步）。因此，步骤4、5和6形成了一个迭代循环，自博弈继续推动模型的改进。六个步骤之间的流程如图 2所示。下一节将详细介绍每个步骤。

[1] 我们还提出了一种替代方法,即仅根据问题生成测试用例。除了能够利用只提供问题的代码数据集外,这种方法还可以在推理阶段应用,实现无需真实代码的在线推理。

Algorithm 1 Self-Play+RL-based Coder Training Framework

Require:

$\mathcal{D}_{\text{code}}$: A dataset containing problems Q_i and solution code C_i.

π_θ: Initial policy model

γ_{TCG}: Test Case Generator(TCG) to create problem-oriented test samples

ρ_{PRM}: Process Reward Model(PRM) to evaluate the quality of intermediate reasoning steps

ϕ: Aggregation function combining result-based and process-based rewards

Ensure:

Optimized policy model π_θ^*

▷ ① Train the Test Case Generator (TCG)

Train γ_{TCG} on $\mathcal{D}_{\text{code}}$ to maximize diversity and correctness of generated test cases $\{(I_i, O_i)\}$.

▷ ② Synthesize Reasoning-enhanced Code Dataset

Based on $\mathcal{D}_{\text{code}} = \{Q_i, C_i\}$, use MCTS to generate $\mathcal{D}_{\text{process}} = \{(Q_i, \cdots, S_i^j, v_i^j, \cdots, C_i') | j = 1, \cdots, m\}$, where S_i^j represents a reasoning step and $v_i^j \in \{0, 1\}$ is a validity indicator with $v_i^m = 1$ when the generated code pass the test cases.

▷ ③ Finetune the Policy Model

Finetune π_θ with SFT on valid steps $\mathcal{D}^+_{\text{process}} = \{(Q_i, S_i^j, C_i') \mid (Q_i, S_i^j, v_i^j, C_i') \in \mathcal{D}_{\text{process}}, \mathbb{I}(C_i') = 1\}$.

while not converged **do**

▷ ④ Initialize/Finetune the Process Reward Model (PRM)

Train/Finetune PRM using SFT on $\mathcal{D}_{\text{process}}$ with point-wise loss, or using DPO with pair-wise loss.

▷ ⑤ Improve the Policy Model with Reinforcement Learning

Initialize $r_i = 0$.

for $j = 1, 2, \ldots, m$ **do**

Generate reasoning step $S_i^j \sim \pi_\theta(S_i^j \mid Q_i, S_i^{1:j-1})$.

Use PRM to compute process-based reward $r_i^j = \rho_{\text{PRM}}(Q_i, S_i^{1:j})$.

end for

Based on Q_i and the complete reasoning sequence $S_i^{1:m}$, generate the final code C_i'.

Use TCG to generate test cases (I_i, O_i) for each problem Q_i with the ground-truth code C_i.

Execute generated code C_i' on inputs I_i to produce outputs O_i'.

Compute result-based reward:

$$R_i = \begin{cases} \tau_{\text{pass}}, & \text{if } O_i' = O_i, \\ \tau_{\text{fail}}, & \text{otherwise.} \end{cases}$$

Update π_θ using a reinforcement learning method guided by the aggregated reward $\phi(R_i, r_i^{1:m})$.

▷ ⑥ Generate New Reasoning Data

Generate new reasoning data $\mathcal{D}'_{\text{process}}$ using the updated π_θ.

Update dataset: $\mathcal{D}_{\text{process}} \leftarrow \mathcal{D}_{\text{process}} \cup \mathcal{D}'_{\text{process}}$.

end while

return Optimized policy model π_θ^*

Algorithm 1 基于自我对弈和强化学习的编码器训练框架

Require:

$\mathcal{D}_{\text{code}}$: A dataset containing problems Q_i and solution code C_i.

π_θ: Initial policy model

γ_{TCG}: Test Case Generator(TCG) to create problem-oriented test samples

ρ_{PRM}: Process Reward Model(PRM) to evaluate the quality of intermediate reasoning steps

ϕ: Aggregation function combining result-based and process-based rewards

Ensure:

Optimized policy model π_θ^*

▷ ① Train the Test Case Generator (TCG)

Train γ_{TCG} on $\mathcal{D}_{\text{code}}$ to maximize diversity and correctness of generated test cases $\{(I_i, O_i)\}$.

▷ ② Synthesize Reasoning-enhanced Code Dataset

Based on $\mathcal{D}_{\text{code}} = \{Q_i, C_i\}$, use MCTS to generate $\mathcal{D}_{\text{process}} = \{(Q_i, \cdots, S_i^j, v_i^j, \cdots, C_i') | j = 1, \cdots, m\}$, where S_i^j represents a reasoning step and $v_i^j \in \{0, 1\}$ is a validity indicator with $v_i^m = 1$ when the generated code pass the test cases.

▷ ③ Finetune the Policy Model

Finetune π_θ with SFT on valid steps $\mathcal{D}^+_{\text{process}} = \{(Q_i, S_i^j, C_i') \mid (Q_i, S_i^j, v_i^j, C_i') \in \mathcal{D}_{\text{process}}, \mathbb{I}(C_i') = 1\}$.

while not converged **do**

▷ ④ Initialize/Finetune the Process Reward Model (PRM)

Train/Finetune PRM using SFT on $\mathcal{D}_{\text{process}}$ with point-wise loss, or using DPO with pair-wise loss.

▷ ⑤ Improve the Policy Model with Reinforcement Learning

Initialize $r_i = 0$.

for $j = 1, 2, \ldots, m$ **do**

Generate reasoning step $S_i^j \sim \pi_\theta(S_i^j \mid Q_i, S_i^{1:j-1})$.

Use PRM to compute process-based reward $r_i^j = \rho_{\text{PRM}}(Q_i, S_i^{1:j})$.

end for

Based on Q_i and the complete reasoning sequence $S_i^{1:m}$, generate the final code C_i'.

Use TCG to generate test cases (I_i, O_i) for each problem Q_i with the ground-truth code C_i.

Execute generated code C_i' on inputs I_i to produce outputs O_i'.

Compute result-based reward:

$$R_i = \begin{cases} \tau_{\text{pass}}, & \text{if } O_i' = O_i, \\ \tau_{\text{fail}}, & \text{otherwise.} \end{cases}$$

Update π_θ using a reinforcement learning method guided by the aggregated reward $\phi(R_i, r_i^{1:m})$.

▷ ⑥ Generate New Reasoning Data

Generate new reasoning data $\mathcal{D}'_{\text{process}}$ using the updated π_θ.

Update dataset: $\mathcal{D}_{\text{process}} \leftarrow \mathcal{D}_{\text{process}} \cup \mathcal{D}'_{\text{process}}$.

end while

return Optimized policy model π_θ^*

4

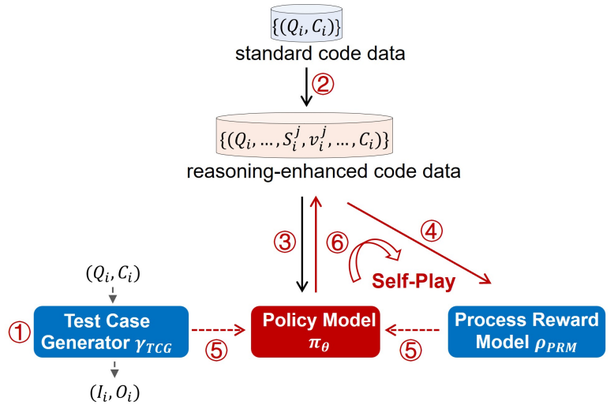

图 2: Self-Play+RL training framework.

3 Method and Intermediate Results

3.1 Test Case Generator Training

3.1.1 Objective

A Test Case Generator is a tool designed to automate the creation of input-output test cases, which plays a critical role in supporting program verification in code generation tasks.

During the training phase, the correctness of the generated code is typically assessed with standard input-output test cases. The pass rate of these test cases serves as a key metric for evaluating the quality of the generated code and acts as an outcome reward signal to guide the training of the policy model. This reward signal helps the model refine its generation strategy, thereby enhancing its capability to produce accurate and functional code.

In the inference phase, when the trained model is tasked with code generation, standard test cases are often not available to verify the correctness of the generated code. The test case generator mitigates this limitation by providing a self-validation mechanism for the policy model, which allows the policy model to evaluate before final generation. As a result, the policy model is able to select the optimal output path based on the validation results.

3.1.2 Training

The training process is divided into two distinct phases: Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO) (Rafailov et al., 2024). We denote the generator which is not fine-tuned as $\gamma_{\text{TCG}_{base}}$.

The primary objective of the SFT phase is to ensure that the generator's output adheres to a predefined format, enabling the accurate parsing and extraction of the generated test cases. The training data for this phase is derived from the TACO dataset (Li et al., 2023), which follows the format $\{question, solution, test\_case\}$. To standardize the model's input and output, we developed a template format, as detailed below:

3 Method and Intermediate Results

3.1 Test Case Generator Training

3.1.1 Objective

测试用例生成器是一种旨在自动化生成输入-输出测试用例的工具，在代码生成任务中支持程序验证方面发挥着关键作用。

在训练阶段，生成代码的正确性通常通过标准的输入-输出测试用例来评估。这些测试用例的通过率是评估生成代码质量的关键指标，同时也是指导策略模型训练的结果奖励信号。这种奖励信号有助于模型优化其生成策略，从而提高其生成准确和功能代码的能力。

在推理阶段，当训练好的模型承担代码生成任务时，通常没有标准测试用例来验证生成代码的正确性。测试用例生成器通过为策略模型提供自验证机制来缓解这一限制，使策略模型能够在最终生成前进行评估。因此，策略模型能够根据验证结果选择最优输出路径。

3.1.2 Training

训练过程分为两个不同的阶段：监督微调（SFT）和直接偏好优化（DPO） (Rafailov et al., 2024)。我们将未微调的生成器表示为 $\gamma_{\text{TCG}_{base}}$。

SFT 阶段的主要目标是确保生成器的输出符合预定义的格式，以便准确解析和提取生成的测试用例。此阶段的训练数据来自 TACO 数据集 (Li et al., 2023)，该数据集遵循 $\{question, solution, test\_case\}$ 格式。为了标准化模型的输入和输出，我们开发了一种模板格式，具体如下：

Template format for TCG SFT

Instruction
Please complete the task in the code part and generate some test case in the test part that can be used to test the quality of the generated code.
Problem
`{question}`
Code Part
`{randomly select one solution from the provided solutions}`
Test Part
`[Generate 3 test cases here to validate the code]`
`{sample 3 test_cases with each formatted as input and output}`

图 3: Template format for TCG SFT

在SFT之后，生成器被表示为$\gamma_{\text{TCG}_{sft}}$。

DPO阶段的目标是引导模型生成符合特定偏好的测试用例，从而提高测试用例生成器的性能和可靠性。在本研究中，我们使用DPO方法与人工构建的样本对来提高模型与所需偏好一致的能力，通过构建偏好数据集。我们的DPO微调依赖于预先构建的偏好数据集 $D_{pref} =$

Template format for TCG SFT

Instruction
Please complete the task in the code part and generate some test case in the test part that can be used to test the quality of the generated code.
Problem
`{question}`
Code Part
`{randomly select one solution from the provided solutions}`
Test Part
`[Generate 3 test cases here to validate the code]`
`{sample 3 test_cases with each formatted as input and output}`

图 3: Template format for TCG SFT

The generator is denoted as $\gamma_{\text{TCG}_{sft}}$ after SFT.

The goal of the DPO phase is to guide the model in generating test cases that align with specific preferences, thereby enhancing both the performance and reliability of the test case generator. In this study, we employ the DPO method with artificially constructed sample pairs to improve the model's ability to align with desired preferences by constructing a preference dataset. Our DPO fine-tuning relies on a pre-constructed preference dataset $D_{pref} = \{x, y_w, y_l\}$, where x is prompt that includes instruction, question, and code; y_w is positive example, i.e., test cases that align with the preference; and y_l is negative example, i.e., test cases that do not align with the preference. We adopt the following rules to construct preference data: for y_w, we directly use the three sampled test cases that are completely matched as positive examples; for y_l, we shuffle the outputs of the three sampled test cases and then concatenate the original inputs so that the input-output pairs of the three test cases do not completely match, and use the three incompletely matched test cases as negative examples. The training objective aims to optimize $\gamma_{\text{TCG}_\theta}$ based on initial SFT model $\gamma_{\text{TCG}_{sft}}$, while incorporating implicit reward modeling with the reference model $\gamma_{\text{TCG}_{\text{ref}}}$, which represents the initial SFT model $\gamma_{\text{TCG}_{sft}}$. The objective function is as follows:

$$\mathcal{L}_{\text{DPO}}(\gamma_{\text{TCG}_\theta}; \gamma_{\text{TCG}_{\text{ref}}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D_{pref}} \left[\log \sigma \left(\beta \log \frac{\gamma_{\text{TCG}_\theta}(y_w|x)}{\gamma_{\text{TCG}_{\text{ref}}}(y_w|x)} - \beta \log \frac{\gamma_{\text{TCG}_\theta}(y_l|x)}{\gamma_{\text{TCG}_{\text{ref}}}(y_l|x)} \right) \right], \quad (1)$$

where $\sigma(x)$ is the sigmoid function and β represents a scaling factor used to adjust the contrast strength between the positive and negative examples during training. The generator is denoted as $\gamma_{\text{TCG}_{dpo}}$ after DPO, which represents the final generator γ_{TCG}.

3.1.3 Experiments

We utilize DeepSeek-1.3B-Instruct (Guo et al., 2024) as the base model for the test case generator, followed by SFT and DPO. The fine-tuning phase employs QLoRA technology (Dettmers et al., 2023) with a rank parameter $r = 1$ to adapt the following modules: $q\_proj, o\_proj, k\_proj, v\_proj, gate\_proj, up\_proj, down\_proj$. The learning rate is set to 5×10^{-4} to balance training stability and convergence speed. The training data is derived from a subset

$\{x, y_w, y_l\}$，其中 x 包括指令、问题和代码的提示；y_w 是正例，即符合偏好的测试用例；y_l 是负例，即不符合偏好的测试用例。我们采用以下规则构建偏好数据：对于 y_w，我们直接使用三个完全匹配的采样测试用例作为正例；对于 y_l，我们打乱三个采样测试用例的输出，然后将原始输入连接起来，使得三个测试用例的输入-输出对不完全匹配，并使用这三个不完全匹配的测试用例作为负例。训练目标旨在基于初始SFT模型$\gamma_{\text{TCG}_{sft}}$优化$\gamma_{\text{TCG}_\theta}$，同时结合参考模型$\gamma_{\text{TCG}_{\text{ref}}}$的隐式奖励建模，该参考模型代表初始SFT模型$\gamma_{\text{TCG}_{sft}}$。目标函数如下：

$$\mathcal{L}_{\text{DPO}}(\gamma_{\text{TCG}_\theta}; \gamma_{\text{TCG}_{\text{ref}}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D_{pref}} \left[\log \sigma \left(\beta \log \frac{\gamma_{\text{TCG}_\theta}(y_w|x)}{\gamma_{\text{TCG}_{\text{ref}}}(y_w|x)} - \beta \log \frac{\gamma_{\text{TCG}_\theta}(y_l|x)}{\gamma_{\text{TCG}_{\text{ref}}}(y_l|x)} \right) \right], \quad (1)$$

其中 $\sigma(x)$ 是sigmoid函数，β 表示一个用于调整训练期间正例和负例之间对比度强度的缩放因子。生成器在DPO之后表示为 $\gamma_{\text{TCG}_{dpo}}$，代表最终的生成器 γ_{TCG}。

3.1.3 Experiments

我们使用DeepSeek-1.3B-Instruct (Guo et al., 2024) 作为测试用例生成器的基础模型，随后进行SFT和DPO。微调阶段采用QLoRA技术 (Dettmers et al., 2023)，秩参数 $r = 1$，以适应以下模块：$q\_proj, o\_proj, k\_proj, v\_proj, gate\_proj, up\_proj, down\_proj$。学习率设置为 5×10^{-4}，以平衡训练的稳定性和收敛速度。训练数据来自TACO训练数据集的一个子集，符合ACM竞赛格式，包含大约10,000个样本。同样，测试数据来自TACO测试数据集的一个子集，符合ICPC竞赛格式，包含314个样本。

我们在TACO测试的不同阶段测试了生成的测试用例的质量。在SFT阶段之后，由 $\gamma_{\text{TCG}_{sft}}$ 生成的测试用例在标准代码上的通过率为80.8%，展示了生成器在初步微调后高效生成测试用例的能力。此外，$\gamma_{\text{TCG}_{dpo}}$ 达到了89.2%的性能，相比 $\gamma_{\text{TCG}_{sft}}$ 有显著的提升。这表明，通过优化模型的决策过程，偏好优化显著提高了生成器生成更可靠测试用例的能力。

在实际场景中，生成器的性能总体上满足了评估代码正确性的要求。展望未来，我们计划将测试用例生成器作为推理过程中的结果验证器。这种方法旨在通过将生成的输出与动态生成的测试用例进行验证，确保生成输出的正确性，从而实现更强大的代码生成推理时搜索。

此外，我们正在考虑在TCG的训练中引入自我对弈。在这种设置中，策略模型将生成旨在通过TCG生成的测试用例的代码，而TCG则旨在生成逐渐更具挑战性的测试用例。这种对抗性互动可以促进策略模型和测试用例生成器的相互改进。

3.2 Reasoning-enhanced Code Data Synthesis

3.2.1 Pseudocode-based Reasoning Process

推理过程的定义至关重要。如引言中所述，我们探索了一种基于伪代码的MCTS方法，旨在指导大型语言模型进行复杂代码任务的深度推理。伪代码作为自然语言描述和实际代码之间的中间表示，提供了一种更抽象和简洁的方式来表达算法或程序的逻辑流程。为了将伪代码推理整合到步骤级的思维链（CoT）中，如图 4所示，我们定义了三个关键的行为动作，这些动作融入了伪代码推理：

- *行动1:使用伪代码定义算法结构*：在此行动中，模型概述了主要函数的结构和接口，而不深入实现细节。目的是使模型能够掌握整体任务结构，包括每个主要函数的输入、输出和核心功能。
- *行动2:优化伪代码*：在此行动中，模型迭代地优化在行动1中定义的伪代码，逐步澄清每个函数的步骤、逻辑和操作，为最终的代码实现做准备。

of the TACO train dataset, which adheres to the ACM competition format and contains approximately 10,000 samples. Similarly, the test data is obtained from a subset of the TACO test dataset, also conforming to the ICPC competition format, and consists of 314 samples.

We tested the quality of the generated test cases at different stages of the TACO test. After the SFT phase, the pass rate of test cases generated by $\gamma_{\text{TCG}_{sft}}$ on the standard code was 80.8%, demonstrating the generator's ability to efficiently produce test cases following preliminary fine-tuning. Furthermore, $\gamma_{\text{TCG}_{dpo}}$ achieved a performance of 89.2%, reflecting an notable improvement compared to $\gamma_{\text{TCG}_{sft}}$. This indicates that preference optimization, by refining the model's decision-making process, significantly enhanced the generator's ability to produce more reliable test cases.

In practical scenarios, the generator's performance has generally met the requirements for assessing code correctness. Looking ahead, we plan to incorporate the test case generator as an outcome verifier during the inference process. This approach aims to ensure the correctness of generated outputs by validating them against dynamically generated test cases, enabling more robust inference-time search for code generation.

Additionally, we are considering the incorporation of self-play in the TCG's training. In this setup, the policy model would generate code intended to pass the test cases produced by the TCG, while the TCG would aim to generate progressively more challenging test cases. This adversarial interaction could foster mutual improvements in both the policy model and the test case generator.

3.2 Reasoning-enhanced Code Data Synthesis

3.2.1 Pseudocode-based Reasoning Process

The definition of the reasoning process is crucial. As mentioned in the *Introduction*, we explore a pseudocode-based MCTS approach designed to guide large language models in deep reasoning for complex code tasks. Pseudocode, serving as an intermediate representation between natural language descriptions and actual code, offers a more abstract and concise way to express the logical flow of algorithms or programs. To integrate pseudocode reasoning into step-level Chain-of-Thought (CoT), as illustrated in Fig. 4, we define three key behavioral actions infused with pseudocode reasoning:

- *Action 1: Defining Algorithm Structures using Pseudocode*: In this action, the model outlines the structure and interface of the main functions, without delving into implementation details. The aim is to enable the model to grasp the overall task structure, including the inputs, outputs, and core functionalities of each primary function.
- *Action 2: Refining the Pseudocode*: In this action, the model iteratively refines the pseudocode defined in Action 1, progressively clarifying the steps, logic, and operations of each function in preparation for the final code implementation.
- *Action 3: Generating Code from the Pseudocode*: The goal of this action is to accurately translate the structure and logic of the pseudocode into executable code, ensuring that the generated code meets the task requirements.

These actions ensure that the model employs pseudocode as a cognitive tool during the reasoning process, enhancing its reasoning capability for complex code generation tasks. It is important to note that these three actions do not imply that the reasoning chain is limited to only these steps. As

Pseudocode Prompt

Instruction
Please refer to the given task description and provide a thought process in the form of step-by-step pseudocode refinement.

A curious user has approached you with a programming question. You should give step-by-step solutions to the user's questions. For each step you can choose one of the following three actions:

<Action 1> Defining algorithm Structures Using pseudocode
Description: Outline the core functions and overall structure of the solution without getting into implementation details. Define inputs, outputs, and the main tasks each function will perform.

<Action 2> Refine part of the pseudocode
Description: Add more details to the pseudocode, specifying the exact steps, logic, and operations each function will carry out. This prepares the pseudocode for actual coding.

<Action 3> Generate python code from the pseudocode
Description: Translate the refined pseudocode into executable Python code, making sure to handle inputs, outputs, and ensure correctness in the implementation.

Note:
- You can choose one of the three actions for each step.
- Provide a detailed explanation of the reasoning behind each step.
- Try to refer to the reference code as much as possible, but you can also modify it if needed (e.g. change variable names, add some comments, etc.).

Examples
`{examples}`

Question
`{question}`

图 4: 伪代码逐步细化提示

- *行动3:从伪代码生成代码*：此行动的目标是准确地将伪代码的结构和逻辑转换为可执行代码，确保生成的代码符合任务要求。

这些操作确保模型在推理过程中将伪代码作为认知工具，从而增强其处理复杂代码生成任务的推理能力。需要注意的是，这三项操作并不意味着推理链仅限于这些步骤。如图 5所示，模型可能需要在整个推理过程中反复调用操作2，以逐步完善伪代码，直到其足够成熟以进行最终的代码生成。

Pseudocode Prompt

Instruction
Please refer to the given task description and provide a thought process in the form of step-by-step pseudocode refinement.

A curious user has approached you with a programming question. You should give step-by-step solutions to the user's questions. For each step you can choose one of the following three actions:

<Action 1> Defining algorithm Structures Using pseudocode
Description: Outline the core functions and overall structure of the solution without getting into implementation details. Define inputs, outputs, and the main tasks each function will perform.

<Action 2> Refine part of the pseudocode
Description: Add more details to the pseudocode, specifying the exact steps, logic, and operations each function will carry out. This prepares the pseudocode for actual coding.

<Action 3> Generate python code from the pseudocode
Description: Translate the refined pseudocode into executable Python code, making sure to handle inputs, outputs, and ensure correctness in the implementation.

Note:
- You can choose one of the three actions for each step.
- Provide a detailed explanation of the reasoning behind each step.
- Try to refer to the reference code as much as possible, but you can also modify it if needed (e.g. change variable names, add some comments, etc.).

Examples
`{examples}`

Question
`{question}`

图 4: Pseudocode Prompt for Step-by-Step Refinement

demonstrated in Fig. 5, the model may need to repeatedly invoke Action 2 throughout the reasoning process to iteratively refine the pseudocode until it is sufficiently developed for the final code generation.

To evaluate the effectiveness of the step-level CoT with pseudocode reasoning, we conducted experiments using the Qwen series of open-source models (Yang et al., 2024) and the Mostly Basic Python Problems (MBPP) dataset (Austin et al., 2021) as the benchmark. In the experiment, we employed a sampling strategy based on Monte Carlo Tree Search (MCTS) and compared Pass@1 for regu-

Question

There are two n-element arrays of integers, A and B. Permute them into some A' and B' such that the relation $A'[i] + B'[i] \geq k$ holds for all i where $0 \leq i < n$. There will be q queries consisting of A, B, and k. For each query, return YES if some permutation A', B' satisfying the relation exists. Otherwise, return NO.
The first line contains an integer q, the number of queries. The next q sets of 3 lines are as follows:
The first line contains two space-separated integers n and k, the size of both arrays A and B, and the relation variable.
The second line contains n space-separated integers $A[i]$.
The third line contains n space-separated integers $B[i]$.
Sample Input
1
3 10
2 1 3
7 8 9
Sample Output
YES

Response

Step 1: Defining High-Level Structure (Action 1)
Goal:
Define the overall structure of the solution by describing the steps needed to solve the problem:
1. Read the number of queries.
2. For each query, read the sizes of the arrays and the integer k.
3. Read and sort both arrays.
4. Check if the condition can be met by comparing corresponding elements from A' and B'.
Pseudocode:
```
Function twoArrays(q):
    For each query from 1 to q:
        Read n and k
        Read array A of size n
        Read array B of size n
        Sort array A in ascending order
        Sort array B in descending order
        If for every index i, A[i] + B[i] >= k:
            Return \"YES\"
        Else:
            Return \"NO\"
```

Step 2: Reading Input and Sorting (Action 2)
Goal:
Implement the input reading and array sorting functionality. Ensure that the sizes and contents of the arrays are correctly processed.
Pseudocode:
```
Function readInputAndSort():
    Initialize a list for results
    For each query:
        Read n and k
        Read array A
        Read array B
        Sort A in ascending order
        Sort B in descending order
        Append sorted arrays to results
    Return results
```

Step 3: Checking Conditions (Action 2)
Goal:
Create a function to check if the condition $A\prime[i] + B\prime[i] \geq k$ holds for all i indices after sorting.
Pseudocode:
```
Function checkValidity(A_sorted, B_sorted, k):
    For i from 0 to n-1:
        If A_sorted[i] + B_sorted[i] < k:
            Return \"NO\"
    Return \"YES\"
```

Step 4: Putting It All Together (Action 2)
Goal:
Combine the input reading, sorting, and checking into a single coherent function for easier overall management.
Pseudocode:
```
Function twoArrays(q):
    results = []
    For i from 1 to q:
        Read n and k
        Read arrays A and B
        Sort A ascending
        Sort B descending
        If checkValidity(A_sorted, B_sorted, k) is \"YES\":
            results.append(\"YES\")
        Else:
            results.append(\"NO\")
    Return results
```

Step 5: Implementing Python Code (Action 3)
Python Code:
The defined structure is implemented in Python as follows:
```
def twoArrays(q):
    results = []

    for _ in range(q):
        n, k = map(int, input().split())
        a_array = list(map(int, input().split()))
        b_array = list(map(int, input().split()))

        # Sort A in ascending order and B in descending order
        a_array.sort()
        b_array.sort(reverse=True)

        # Check validity
        valid = True  # Assume valid until proven otherwise
        for i in range(n):
            if a_array[i] + b_array[i] < k:
                valid = False  # Found a case that fails the condition
                break

        results.append(\"YES\" if valid else \"NO\")

    return results
# Read number of queries
q = int(input())
# Process and print results for all queries
results = twoArrays(q)
for res in results:
    print(res)
```
This code accurately reads multiple queries, sorts the arrays, checks the required conditions, and outputs the results as specified.

图 5: 生成的示例代码带有伪代码CoT

Question

There are two n-element arrays of integers, A and B. Permute them into some A' and B' such that the relation $A'[i] + B'[i] \geq k$ holds for all i where $0 \leq i < n$. There will be q queries consisting of A, B, and k. For each query, return YES if some permutation A', B' satisfying the relation exists. Otherwise, return NO.
The first line contains an integer q, the number of queries. The next q sets of 3 lines are as follows:
The first line contains two space-separated integers n and k, the size of both arrays A and B, and the relation variable.
The second line contains n space-separated integers $A[i]$.
The third line contains n space-separated integers $B[i]$.
Sample Input
1
3 10
2 1 3
7 8 9
Sample Output
YES

Response

Step 1: Defining High-Level Structure (Action 1)
Goal:
Define the overall structure of the solution by describing the steps needed to solve the problem:
1. Read the number of queries.
2. For each query, read the sizes of the arrays and the integer k.
3. Read and sort both arrays.
4. Check if the condition can be met by comparing corresponding elements from A' and B'.
Pseudocode:

```
Function twoArrays(q):
    For each query from 1 to q:
        Read n and k
        Read array A of size n
        Read array B of size n
        Sort array A in ascending order
        Sort array B in descending order
        If for every index i, A[i] + B[i] >= k:
            Return \"YES\"
        Else:
            Return \"NO\"
```

Step 2: Reading Input and Sorting (Action 2)
Goal:
Implement the input reading and array sorting functionality. Ensure that the sizes and contents of the arrays are correctly processed.
Pseudocode:

```
Function readInputAndSort():
    Initialize a list for results
    For each query:
        Read n and k
        Read array A
        Read array B
        Sort A in ascending order
        Sort B in descending order
        Append sorted arrays to results
    Return results
```

Step 3: Checking Conditions (Action 2)
Goal:
Create a function to check if the condition $A[i] + B[i] \geq k$ holds for all i indices after sorting.
Pseudocode:

```
Function checkValidity(A_sorted, B_sorted, k):
    For i from 0 to n-1:
        If A_sorted[i] + B_sorted[i] < k:
            Return \"NO\"
    Return \"YES\"
```

Step 4: Putting It All Together (Action 2)
Goal:
Combine the input reading, sorting, and checking into a single coherent function for easier overall management.
Pseudocode:

```
Function twoArrays(q):
    results = []
    For i from 1 to q:
        Read n and k
        Read arrays A and B
        Sort A ascending
        Sort B descending
        If checkValidity(A_sorted, B_sorted, k) is \"YES\":
            results.append(\"YES\")
        Else:
            results.append(\"NO\")
    Return results
```

Step 5: Implementing Python Code (Action 3)
Python Code:
The defined structure is implemented in Python as follows:

```
def twoArrays(q):
    results = []

    for _ in range(q):
        n, k = map(int, input().split())
        a_array = list(map(int, input().split()))
        b_array = list(map(int, input().split()))

        # Sort A in ascending order and B in descending order
        a_array.sort()
        b_array.sort(reverse=True)

        # Check validity
        valid = True  # Assume valid until proven otherwise
        for i in range(n):
            if a_array[i] + b_array[i] < k:
                valid = False  # Found a case that fails the condition
                break

        results.append(\"YES\" if valid else \"NO\")

    return results
# Read number of queries
q = int(input())
# Process and print results for all queries
results = twoArrays(q)
for res in results:
    print(res)
```

This code accurately reads multiple queries, sorts the arrays, checks the required conditions, and outputs the results as specified.

图 5: Generated example code with pseudocode CoT

Model	Qwen2.5-1.5B		Qwen2.5-3B		Qwen2.5-7B		Qwen2.5-Coder-7B	
	Vanilla	Pseudocode	Vanilla	Pseudocode	Vanilla	Pseudocode	Vanilla	Pseudocode
Pass@1(%)	**55.8**	46.7(-9.1)	**56.3**	51.3(-5.0)	**59.8**	50.1(-9.7)	57.7	**58.2**(+0.5)
ASPR(%)	49.9	**54.5**(+4.6)	52.0	**70.6**(+18.6)	66.4	**78.1**(+11.7)	49.3	**74.9**(+25.6)

表 1: 基于伪代码的代码生成在 MBPP 基准上的结果。*Pass@1* 表示总体通过率。*ASPR*（平均采样通过率）表示在最后一步达到正确推理路径的平均成功率。

为了评估步骤级 CoT 与伪代码推理的有效性，我们使用了 Qwen 系列开源模型 (Yang et al., 2024) 和 Mostly Basic Python Problems (MBPP) 数据集 (Austin et al., 2021) 作为基准进行了实验。在实验中，我们采用了一种基于蒙特卡洛树搜索 (MCTS) 的采样策略，并比较了常规 CoT 和带有伪代码推理的 CoT 的 Pass@1 指标，以及正确推理路径上最后一步的平均采样通过率 (ASPR)。我们的结果表明，当推理正确时，引入伪代码显著提高了生成代码的质量。

表 1 展示了实验结果。尽管基于伪代码的推理通常会使 Pass@1 指标下降，但我们观察到 ASPR 有显著增加，这表明伪代码增强了整体推理过程，特别是在精炼通向正确最终输出的路径方面。这表明准确的伪代码对最终正确代码的生成有重要贡献。然而，纯语言模型在生成有效的伪代码方面仍面临挑战，这正是后续 SFT 初始化和 Self-Play+RL 增强的目标。

3.2.2 Reasoning Process Data Synthesis

我们使用蒙特卡洛树搜索（MCTS） (Kocsis & Szepesvári, 2006; Feng et al., 2023; Qi et al., 2024) 来构建形式为 $\mathcal{D}_{\text{process}} = \{(Q_i, \cdots, S_i^j, v_i^j, \cdots, C_i')\}$ 的步骤级过程奖励数据，其中 v_i^j 表示截至步骤 S_i^j 的推理路径的评估值，而 C_i' 是从最终步骤 S_i^m 导出的可执行代码。在这个过程中，我们采用标准的 MCTS 滚动策略进行路径探索。对于每个问题 Q_i，我们应用前面定义的伪代码提示策略来指导推理过程。当到达终止节点 S_i^m 时，形成一个完整的伪代码推理路径 $(Q_i, S_i^1, \ldots, S_i^m)$。终止节点 S_i^m 的奖励值 v_i^m 是根据两个关键指标计算的：

- 编译成功率 *(compile)*: 该指标确定生成的代码是否可以成功编译。值 *compile* 是二进制的，其中 $compile = 1$ 表示成功，$compile = 0$ 表示失败。
- 测试用例通过率 *(pass)*: 在成功编译后，我们进一步评估生成的代码是否通过测试用例。通过率计算公式为 $pass = \frac{\text{Num}_{\text{passed}}}{\text{Num}_{\text{test\_case}}}$，其中 $\text{Num}_{\text{passed}}$ 是通过的测试用例数量，$\text{Num}_{\text{test\_case}}$ 是用于验证的测试用例总数。

终端节点 S_i^m 的奖励值计算为这两个指标的加权和：

$$v_i^m = \alpha \cdot compile + (1 - \alpha) \cdot pass,$$

其中 α 是一个控制编译成功和测试通过率相对重要性的超参数。

一旦计算出终端节点的奖励值 v_i^m，我们将此值反向传播到路径上的所有前序节点，为每个步骤 (S_i^j, v_i^j) 分配一个奖励值 v_i^j。由于在 MCTS 过程中进行了多次模拟，反向传播期间节点 v_i^j 的累积奖励可能超过 1。因此，我们使用以下公式对路径上每个节点的奖励值进行归一化，以获得最终的步骤有效性值。

在构建推理过程数据集时，对于每个问题 Q_i，如果通过搜索找到了正确答案，我们就能确保至少获得一个终端节点 (S_i^m, v_i^m) 且 $v_i^m = 1$。完成搜索后，我们从正确的终端节点

Model	Qwen2.5-1.5B		Qwen2.5-3B		Qwen2.5-7B		Qwen2.5-Coder-7B	
	Vanilla	Pseudocode	Vanilla	Pseudocode	Vanilla	Pseudocode	Vanilla	Pseudocode
Pass@1(%)	**55.8**	46.7**(-9.1)**	**56.3**	51.3**(-5.0)**	**59.8**	50.1**(-9.7)**	57.7	**58.2**(+0.5)
ASPR(%)	49.9	**54.5**(+4.6)	52.0	**70.6**(+18.6)	66.4	**78.1**(+11.7)	49.3	**74.9**(+25.6)

表 1: Pseudocode-based code generation results on the MBPP Benchmark. *Pass@1* indicates the overall pass rate. *ASPR* (Average Sampling Pass Rate) indicates the average success rate of reaching the correct reasoning path on the last step.

lar CoT and CoT with pseudocode reasoning, as well as the Average Sampling Pass Rate (ASPR) of the last step on the correct reasoning path. Our results indicate that incorporating pseudocode significantly improves the quality of the generated code when the reasoning is correct.

Table 1 presents the results. While the Pass@1 metric generally decreases with pseudocode-based reasoning, we observed a significant increase in ASPR, indicating that pseudocode enhances the overall reasoning process, particularly in refining the path toward the correct final output. This suggests that accurate pseudocode highly contributes to the final correct code. However, vanilla LLMs still face challenges in generating effective pseudocode, which is precisely the goal of the subsequent SFT initialization and Self-Play+RL enhancement.

3.2.2 Reasoning Process Data Synthesis

We use Monte Carlo Tree Search (MCTS) (Kocsis & Szepesvári, 2006; Feng et al., 2023; Qi et al., 2024) to construct step-level process reward data in the form of $\mathcal{D}_{\text{process}} = \{(Q_i, \cdots, S_i^j, v_i^j, \cdots, C_i')\}$, where v_i^j represents the evaluation of the reasoning path up to step S_i^j, and C_i' is the executable code derived from the final step S_i^m. In this process, we employ the standard MCTS rollout strategy for path exploration. For each problem Q_i, we apply the pseudocode prompt strategy defined earlier to guide the reasoning process. When a terminal node S_i^m is reached, a complete pseudocode reasoning path $(Q_i, S_i^1, \ldots, S_i^m)$ is formed. The reward value v_i^m for the terminal node S_i^m is computed based on two key metrics:

- *Compilation success rate (compile)*: This metric determines whether the generated code can successfully compile. The value *compile* is binary, with *compile* $= 1$ indicating success and *compile* $= 0$ indicating failure.
- *Test case pass rate (pass)*: Given a successful compilation, we further evaluate whether the generated code passes the test cases. The pass rate is calculated as $\textit{pass} = \frac{\text{Num}_{\text{passed}}}{\text{Num}_{\text{test\_case}}}$, where $\text{Num}_{\text{passed}}$ is the number of passed test cases and $\text{Num}_{\text{test\_case}}$ is the total number of test cases used for validation.

The reward value for the terminal node S_i^m is calculated as a weighted sum of these two metrics:

$$v_i^m = \alpha \cdot \textit{compile} + (1 - \alpha) \cdot \textit{pass},$$

where α is a hyperparameter controlling the relative importance of compilation success and test pass rate.

$(Q_i, S_i^1, \ldots, S_i^m, v_i^m), v_i^m = 1$ 选择完整的推理路径，以形成策略模型的初始化数据集。该数据集表示为：

$$\mathcal{D}_{\text{process}}^{+} = \{(Q_i, S_i^j, C_i') \mid (Q_i, S_i^j, v_i^j, C_i') \in \mathcal{D}_{\text{process}}, \mathbb{I}(C_i') = 1\},$$

其中 $\mathbb{I}(\cdot)$ 是一个指示函数，如果生成的代码 C_i' 通过所有测试用例，则返回 1。

3.3 Policy Model Initialization

在完成第 3.2节中描述的推理数据合成任务后，我们使用数据集中每个完整的推理解决方案来初始化策略模型π_θ。这一步旨在帮助π_θ更好地理解任务要求并遵循预期的行为，为后续的迭代训练提供一个最优的起点。

给定问题Q_i，策略模型π_θ在步骤j生成的具体推理步骤内容可以表示为$\pi_\theta(S_i^j \mid Q_i, S_i^{1:j-1})$，其中$S_i^j = (w_1, w_2, \ldots, w_k)$。这里，$S_i^j$表示推理步骤的内容，由特定的分隔符界定，$w$表示$\pi_\theta$在每个解码步骤生成的标记。$S_i^{1:j-1}$表示由前推理步骤的输出形成的情境。

然后，使用一组验证正确的推理解决方案$\mathcal{D}_{\text{process}}^{+}$来初始化策略模型$\pi_\theta$。此初始化通过优化以下训练目标来完成：

$$\mathcal{L}_{\text{SFT}} = -\sum\nolimits_{(Q_i, S_i^j, C_i') \in \mathcal{D}_{\text{process}}^{+}} \log \pi_\theta(S_i^{1:m} \circ C_i' \mid Q_i), \tag{2}$$

其中 $\circ$ 表示推理步骤 $S_i^{1:m}$ 和最终代码 C_i' 的连接。初始化的策略模型 π_θ^{SFT} 将作为后续训练阶段的基础。

3.4 PRM Training

给定一个问题 Q_i 和对应于当前状态的解决方案前缀，过程奖励模型 (PRM)，记为 $Q \times S \to \mathbb{R}^+$，为当前步骤 S_i^j 分配一个奖励值，以估计其对最终答案的贡献。基于第 3.2 节中数据合成过程中使用的树搜索方法，可以使用两种数据组织格式来训练过程奖励模型，分别称为逐点和成对，详细描述如下。

逐点 在这种格式中，从搜索树收集的数据被组织为 $D = \{(Q_i, S_i^{1:j-1}, S_i^j, v_i^j) \mid i = 1, 2, \ldots, N\}$，其中 N 是样本数量，v_i^j 表示在树搜索过程中分配给步骤 S_i^j 的值标签。根据处理方法，此标签可以用于得出硬估计或软估计。按照 (Wang et al., 2024c) 中的方法，PRM 使用以下目标进行训练：

$$\mathcal{L}_{\text{PRM}}^{\textit{point-wise}} = -\mathbb{E}_{(Q_i, S_i^{1:j-1}, S_i^j, v_i^j) \sim D}\Big[v_i^j \log r(Q_i, S_i^{1:j}) + (1 - v_i^j) \log\Big(1 - r(Q_i, S_i^{1:j})\Big)\Big], \tag{3}$$

其中 $r(Q_i, S_i^{1:j})$ 是 PRM 分配的归一化预测分数。

成对 在成对格式中，对于搜索树深度 d 的节点 n^d，其子节点表示为 $\sum_i n_i^{d+1}$，偏好对数据被组织为 $D_{\text{pair}} = \{(Q_i, S_i^{1:j-1}, S_i^{j_{\text{win}}}, S_i^{j_{\text{lose}}}) \mid i = 1, 2, \ldots, N\}$。这里，$S_i^{j_{\text{win}}}$ 表示在树搜索过程中比 $S_i^{j_{\text{lose}}}$ 获得更高价值估计的推理步骤。

根据 Bradley-Terry 模型 (Bradley & Terry, 1952)，PRM 使用以下目标进行训练：

$$\mathcal{L}_{PRM}^{\textit{pair-wise}} = -\mathbb{E}_{(Q_i, S_i^{1:j-1}, S_i^{j_{win}}, S_i^{j_{lose}}) \sim D_{pair}}\Big[\log\Big(\sigma\big(r(Q_i, S_i^{1:j-1}, S_i^{j_{win}}) - r(Q_i, S_i^{1:j-1}, S_i^{j_{lose}})\big)\Big)\Big], \tag{4}$$

Once the reward value v_i^m is computed for the terminal node, we backpropagate this value to all preceding nodes along the path, assigning a reward value v_i^j to each step (S_i^j, v_i^j). Due to the multiple rollouts in the MCTS process, the cumulative reward for a node v_i^j during backpropagation may exceed 1. Therefore, we normalize the reward values for each node along the path using the following formula to obtain the final step validity value.

When constructing the reasoning process dataset, for each problem Q_i, if a correct answer is found through the search, we are guaranteed to obtain at least one terminal node (S_i^m, v_i^m) with $v_i^m = 1$. After completing the search, we select the full reasoning path from the correct terminal node $(Q_i, S_i^1, \ldots, S_i^m, v_i^m), v_i^m = 1$ to form the initialization dataset for the policy model. This dataset is denoted as:

$$\mathcal{D}^+_{\text{process}} = \{(Q_i, S_i^j, C_i') \mid (Q_i, S_i^j, v_i^j, C_i') \in \mathcal{D}_{\text{process}}, \mathbb{I}(C_i') = 1\},$$

where $\mathbb{I}(\cdot)$ is an indicator function that returns 1 if the generated code C_i' passes all the test cases.

3.3 Policy Model Initialization

After completing the reasoning data synthesis tasks described in Section 3.2, we use each complete reasoning solution in the dataset to initialize the policy model π_θ. This step aims to help π_θ better understand the task requirements and follow the expected action behavior, providing an optimal starting point for subsequent iterative training.

Given the question Q_i, the specific reasoning step content generated by the policy model π_θ at step j can be expressed as $\pi_\theta(S_i^j \mid Q_i, S_i^{1:j-1})$, where $S_i^j = (w_1, w_2, \ldots, w_k)$. Here, S_i^j represents the content of a reasoning step, delimited by specific separators, with w denoting the tokens generated by π_θ at each decoding step. $S_i^{1:j-1}$ represents the context formed by the outputs of the previous reasoning steps.

The policy model π_θ is then initialized using the set of verified, correct reasoning solutions $\mathcal{D}^+_{\text{process}}$. This initialization is performed by optimizing the following training objective:

$$\mathcal{L}_{\text{SFT}} = -\sum\nolimits_{(Q_i, S_i^j, C_i') \in \mathcal{D}^+_{\text{process}}} \log \pi_\theta(S_i^{1:m} \circ C_i' \mid Q_i), \tag{2}$$

where $\circ$ denotes the concatenation of the reasoning steps $S_i^{1:m}$ and the final code C_i'. The initialized policy model π_θ^{SFT} will then serve as the foundation for subsequent training stages.

3.4 PRM Training

Given a problem Q_i and a solution prefix corresponding to the current state, the Process Reward Model (PRM), denoted as $Q \times S \to \mathbb{R}^+$, assigns a reward value to the current step S_i^j to estimate its contribution to the final answer. Based on the tree search approach used during data synthesis in Section 3.2, two formats of data organization can be used for training the process reward model, referred to as point-wise and pair-wise, are described in detail below.

Point-wise In this format, data collected from the search tree are organized as $D = \{(Q_i, S_i^{1:j-1}, S_i^j, v_i^j) \mid i = 1, 2, \ldots, N\}$, where N is the number of samples, and v_i^j represents the value label assigned to step S_i^j during the tree search process. Depending on the processing

其中 $\sigma(x)$ 表示sigmoid函数。与点对点设置不同，这里的分数 r 没有进行归一化。这使得模型能够专注于学习动作之间的相对偏好，而不是绝对值预测。

3.5 RL-based Policy Model Improvement

我们将代码生成任务建模为语言增强的马尔可夫决策过程（MDP），形式上表示为 $\mathcal{M} = (\mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \phi)$ (Team, 2024; Carta et al., 2023)。在这个框架中，$\mathcal{V}$ 表示词汇表，$w \in \mathcal{V}$ 表示模型生成的单个标记。动作空间 $\mathcal{A} \subseteq \mathcal{V}^N$ 和状态空间 $\mathcal{S} \subseteq \mathcal{V}^N$ 是标记序列的集合，这意味着动作和状态都是标记序列。在这个框架中，s_0 表示问题，动作 a_i 被视为一个推理步骤（对应于算法 1 中的 S_i），包括动作的类型及其相应的思考链。状态转移函数 $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ 定义了当采取动作 $a_t \in \mathcal{A}$ 时当前状态 $s_t \in \mathcal{S}$ 如何变化。具体来说，动作 a_t 会将标记添加到当前状态，形成新的状态 $s_{t+1} = \mathcal{T}(s_t, a_t)$。这个过程会持续进行，直到模型生成最终解决方案。奖励函数 $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ 评估中间步骤的质量，例如推理过程或生成的代码片段。函数 ϕ 将基于过程和基于结果的奖励结合，生成最终的奖励信号。

在每一步中，模型选择一个动作 $a_t \in \mathcal{A}$，将系统转移到新的状态 $s_{t+1} = \mathcal{T}(s_t, a_t)$。执行动作后，模型从 PRM 接收一个过程奖励 $r^t = \rho_{\text{PRM}}(s_{t-1}, a_t)$。这个过程会重复进行，直到模型生成最终代码或达到预定义的最大深度。

一旦模型生成最终代码或完成搜索过程，结果奖励 R_i 通过测试生成的代码对一系列测试用例进行评估。我们提出一个奖励聚合函数，该函数结合了时间依赖权重和折扣因子：

$$\phi(R_i, r_i^{1:m}) = \alpha(t) \cdot R_i + (1 - \alpha(t)) \cdot \frac{1}{m} \sum_{j=1}^{m} \gamma^j r_i^j,$$

其中 $\alpha(t)$ 是一个随时间变化的因素，用于调整最终奖励 R_i 和累积中间奖励 $r_i^{1:m}$ 之间的平衡。例如，$\alpha(t)$ 可能随时间减少，逐渐增加对中间奖励的权重，同时减少对最终奖励的重视，因为模型逐渐接近最优策略。$r_i^{1:m}$，$\alpha(t)$ 通常遵循线性或对数衰减等时间表。参数 $\gamma \in [0, 1]$ 是折扣因子，决定了未来奖励相对于即时奖励的重要性。聚合的奖励信号用于改进模型的策略，通常通过实现强化学习算法（如 PPO (Ziegler et al., 2019) 和迭代 DPO(Rafailov et al., 2024)）来实现。

通过这种设置，我们定义了一个专门针对代码生成任务的强化学习环境。模型的动作由过程奖励和结果奖励驱动，过程奖励鼓励中间推理步骤，结果奖励反映最终代码的正确性。这种双重奖励结构有助于模型随着时间的推移提高其代码生成能力。

3.6 New Reasoning Data Generation and Self-Play

在第6步中，更新后的策略模型 π_θ 用于生成新的推理数据，记为 $\mathcal{D}'_{\text{process}}$。这些数据是通过推理新的问题实例 Q_i 生成的，生成的推理路径为 $\{S_i^1, S_i^2, \ldots, S_i^m\}$，每条路径最终生成一个最终代码输出 C_i'。推理步骤是迭代生成的，其中每个步骤 S_i^j 都基于前一步骤。

一旦生成新的推理数据，它将被添加到现有的数据集 $\mathcal{D}_{\text{process}}$ 中，形成更新后的数据集 $\mathcal{D}_{\text{process}} \leftarrow \mathcal{D}_{\text{process}} \cup \mathcal{D}'_{\text{process}}$。这一更新增加了推理示例的多样性和质量，为后续步骤提供了更全面的训练材料。

这一新的数据生成过程使得迭代自我训练循环得以实现。在添加新的推理数据后，模型将进行进一步的微调，从第4步中描述的更新PRM开始。PRM反过来使用第5步中描述的RL调整策略模型。这一数据生成、奖励模型更新和策略改进的迭代循环确保了系统推理能力的持续提升。

method, this label can be used to derive either hard or soft estimates. Following the approach in (Wang et al., 2024c), the PRM is trained using the objective:

$$\mathcal{L}_{\text{PRM}}^{\textit{point-wise}} = -\mathbb{E}_{(Q_i, S_i^{1:j-1}, S_i^j, v_i^j)\sim D}\Big[v_i^j \log r(Q_i, S_i^{1:j}) + (1 - v_i^j) \log\Big(1 - r(Q_i, S_i^{1:j})\Big)\Big], \quad (3)$$

where $r(Q_i, S_i^{1:j})$ is the normalized prediction score assigned by the PRM.

Pair-wise In the pair-wise format, for a node n^d at depth d of the search tree, with its child nodes represented as $\sum_i n_i^{d+1}$, preference pair data are organized as $D_{\text{pair}} = \{(Q_i, S_i^{1:j-1}, S_i^{j_{\text{win}}}, S_i^{j_{\text{lose}}}) \mid i = 1, 2, \ldots, N\}$. Here, $S_i^{j_{\text{win}}}$ represents the reasoning step that achieved a higher value estimate during the tree search compared to $S_i^{j_{\text{lose}}}$.

Following the Bradley-Terry model (Bradley & Terry, 1952), the PRM is trained using the following objective:

$$\mathcal{L}_{PRM}^{\textit{pair-wise}} = -\mathbb{E}_{(Q_i, S_i^{1:j-1}, S_i^{j_{win}}, S_i^{j_{lose}})\sim D_{pair}}\Big[\log\Big(\sigma\big(r(Q_i, S_i^{1:j-1}, S_i^{j_{win}}) - r(Q_i, S_i^{1:j-1}, S_i^{j_{lose}})\big)\Big)\Big], \quad (4)$$

where $\sigma(x)$ denotes the sigmoid function. Unlike the point-wise setting, the scores r here are not normalized. This enables the model to focus on learning relative preferences between actions rather than absolute value predictions.

3.5 RL-based Policy Model Improvement

We model the code generation task as a language-augmented Markov Decision Process (MDP), formally represented as $\mathcal{M} = (\mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \phi)$ (Team, 2024; Carta et al., 2023). In this framework, $\mathcal{V}$ denotes the vocabulary, and $w \in \mathcal{V}$ represents an individual token generated by the model. The action space $\mathcal{A} \subseteq \mathcal{V}^N$ and the state space $\mathcal{S} \subseteq \mathcal{V}^N$ are sets of token sequences, meaning that both actions and states are sequences of tokens. In this framework, s_0 represents the question, and the action a_i is considered a reasoning step (referring to the S_i in algorithm 1), which consists of both the type of action and its corresponding chain of thought. The state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defines how the current state $s_t \in \mathcal{S}$ changes when an action $a_t \in \mathcal{A}$ is taken. Specifically, the action a_t appends tokens to the current state, forming a new state $s_{t+1} = \mathcal{T}(s_t, a_t)$. This process continues until the model generates the final solution. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ evaluates the quality of intermediate steps, such as the reasoning process or generated code fragments. The function ϕ combines process-based and outcome-based rewards to produce a final reward signal.

At each step, the model selects an action $a_t \in \mathcal{A}$, which transitions the system to a new state $s_{t+1} = \mathcal{T}(s_t, a_t)$. After executing the action, the model receives a process reward $r^t = \rho_{\text{PRM}}(s_{t-1}, a_t)$ from PRM. This process repeats until the model either generates the final code or reaches the predefined maximum depth.

Once the model generates the final code or completes the search process, the outcome reward R_i is evaluated by testing the generated code against a series of test cases. We propose a reward aggregation function that incorporates both time-dependent weights and a discount factor:

$$\phi(R_i, r_i^{1:m}) = \alpha(t) \cdot R_i + (1 - \alpha(t)) \cdot \frac{1}{m} \sum_{j=1}^{m} \gamma^j r_i^j,$$

where $\alpha(t)$ is a time-varying factor that adjusts the balance between the final reward R_i and the cumulative intermediate rewards $r_i^{1:m}$ over time. For instance, $\alpha(t)$ may decrease over time, gradually

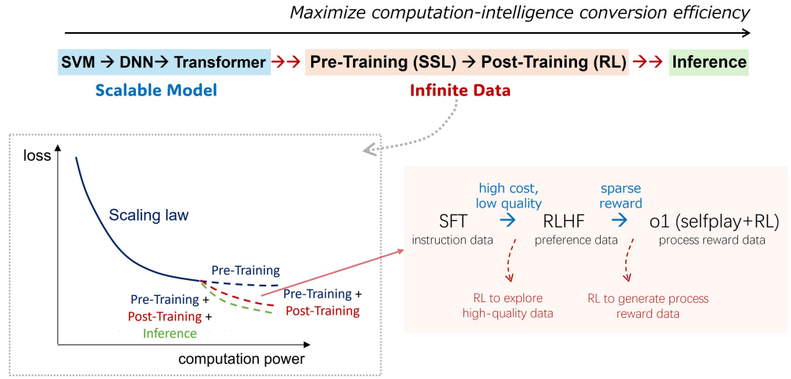

图 6: 趋向于最大化计算智能转换效率。

4 Discussions

4.1 Bitter Lesson: Data is All You Need

在过去十年中，人工智能领域一直在沿着一条主线发展，即最大化计算-智能转换效率 (Chung, 2024)，这指的是将不断增加的计算能力高效地转化为更高的智能水平。沿着这条主线，如图 6顶部所示，早期的进展主要集中在模型方面的改进：从SVM到DNN，再到Transformer，设计了可扩展的模型架构以充分利用计算能力。

近年来，重点转向了数据方面。诸如预训练中的半监督学习（SSL）和后训练中的强化学习（RL）等技术，旨在更有效地利用自然和合成数据。o1模型继续沿这条路线发展。它从利用高质量监督数据的SFT，转向利用环境反馈的RLHF，以访问理论上无限的数据，最终采用o1的创新方法，通过从生成的推理过程中派生的奖励信号来监督生成过程。

这一进展表明，随着Transformer架构现在能够扩展以处理大量数据和训练足够大的模型，剩下的唯一挑战就是获取足够的数据。一种方法是在数据不足的地方收集数据，例如用于系统2能力的推理数据或用于具身智能的物理世界轨迹。另一种方法是探索人类世界中尚不存在的数据类型，这需要进一步探索诸如RL和自我博弈等技术。

4.2 Sweet Lesson: Beyond Human Data

对LLM的一个常见批评是它依赖于现有的人类记录数据，这本质上限制了其潜力。正如维特根斯坦所说：“我的语言的界限意味着我的世界的界限。” 人类语言记录的有限范围和深度限制了LLM的认知能力。然而，o1的成功表明，我们现在可以通过RL探索这些记录数据背后的思维过程。这一进展标志着AI开发的一个关键转变，从单纯模仿人类语言转向自主生成新的认知过程。

更有趣的是，这些思维过程数据不一定局限于自然语言。正如最近一篇《自然》论文所指出的，“语言主要作为交流的工具，而不是思维的本质。” (Fedorenko et al., 2024) 在我们的观察中，o1生成的一些思维链包含无意义的文本，这表明思维标记可能不对应于离散的自然语言词汇。如果模型已经发展出一种更高效的内部表示形式来进行思考，这将显著提高

placing more weight on the intermediate rewards as the model refines its solution, while reducing the emphasis on the final reward as the model approaches the optimal policy. $r_i^{1:m}$, with $\alpha(t)$ typically following schedules such as linear or logarithmic decay. The parameter $\gamma \in [0, 1]$ is the discount factor, which determines the importance of future rewards relative to immediate rewards. The aggregated reward signal is employed to refine the model's policy, typically through the implementation of reinforcement learning algorithms such as PPO (Ziegler et al., 2019) and iterative DPO(Rafailov et al., 2024).

With this setup, we define a reinforcement learning environment tailored for the code generation task. The model's actions are driven by both process-based rewards, which encourage intermediate reasoning steps, and outcome-based rewards, which reflect the correctness of the final code. This dual reward structure helps the model improve its code generation ability over time.

3.6 New Reasoning Data Generation and Self-Play

In step 6, the updated policy model π_θ is used to generate new reasoning data, denoted as $\mathcal{D}'_{\text{process}}$. This data is created by reasoning through new problem instances Q_i, generating step-by-step reasoning paths $\{S_i^1, S_i^2, \ldots, S_i^m\}$, with each path culminating in a final code output C_i'. The reasoning steps are generated iteratively, where each step S_i^j is conditioned on the previous steps.

Once the new reasoning data is generated, it is added to the existing dataset $\mathcal{D}_{\text{process}}$ to form an updated dataset $\mathcal{D}_{\text{process}} \leftarrow \mathcal{D}_{\text{process}} \cup \mathcal{D}'_{\text{process}}$. This update increases the diversity and quality of the reasoning examples, providing more comprehensive training material for subsequent steps.

This new data generation process enables the iterative self-play training loop. After adding the new reasoning data, the model undergoes further fine-tuning, starting with updating PRM as described in the 4th step. The PRM, in turn, adjusts the policy model with RL described in the 5th step. This iterative cycle of data generation, reward model updating, and policy improvement ensures sustained improvement in the system's reasoning ability.

4 Discussions

4.1 Bitter Lesson: Data is All You Need

Over the last decade, the AI field has been developing along a central line towards maximizing computation-intelligence conversion efficiency (Chung, 2024), which is to efficiently convert the ever-increasing computing power into higher intelligence levels. Along this line, as illustrated at the top of Fig. 6, early advancements prioritized improvements on the model side: from SVM to DNN and then to Transformer, scalable model architectures were designed to fully leverage computational power.

In recent years, the focus has shifted towards the data side. Techniques such as Semi-Supervised Learning (SSL) in pre-training and Reinforcement Learning (RL) in post-training have aimed to harness natural and synthesized data more effectively. The o1 model continues this line. It moves from SFT, which leverages high-quality supervised data, to RLHF, which utilizes environmental feedback to access theoretically unlimited data, and finally to o1's innovative approach of supervising the generation process through reward signals derived from the generated reasoning process itself.

思维过程和问题解决机制的效率，不仅超越了人类语言数据的限制，还进一步释放了模型能力的潜力。

4.3 Opportunities: System1 + X to System2 + X

自我对弈的强化学习（RL）框架为探索底层数据提供了一个可行的解决方案，这为许多以前依赖于系统1能力的任务探索系统2解决方案开辟了可能性。通过将更加深思熟虑、逐步的过程整合到任务执行中，我们相信这种方法可以在广泛的领域内产生积极的结果 (Kant et al., 2024; Ganapini et al., 2021; Valmeekam et al., 2024; Lowe, 2024)。传统上使用系统1能力解决的任务，如奖励建模 (Mahan et al., 2024)、机器翻译 (Zhao et al., 2024)、检索增强生成（RAG） (Li et al., 2024)和多模态问答 (Islam et al., 2024)，已经从系统2思维所赋予的更深层次的推理能力中受益。

o1模型的系统卡展示了模型安全性方面的显著改进。受此启发，我们最近探索了*系统2*对齐的概念，这涉及引导模型彻底评估输入、考虑潜在风险并纠正其推理中的偏差 (Wang & Sang, 2024)。我们介绍了三种实现系统2对齐的方法：提示、监督微调和带有过程监督的强化学习。我们正在将本报告中提出的自我对弈+RL框架应用于系统2对齐，旨在进一步增强模型的深思熟虑的思考能力，并减少在复杂场景中的脆弱性。

4.4 Challenges: World Model Encoding

目前发布的 o1-preview 和 o1-mini 缺乏多模态能力和功能调用，这些功能据 OpenAI 称将在其完整版本中包含。除了多模态和功能调用之外，o1 类推理模型的另一个关键改进领域是推理时间的优化。这包括提高推理效率——实现单位时间内更高的性能——以及实现推理时间的自适应调整。具体来说，这涉及根据任务复杂性动态调整系统 2 的推理过程，并实现更像人类的能力，无缝切换系统 1 和系统 2 的推理模式。

为了将 o1 类推理模型部署到更广泛的实际应用中，需要解决两个主要挑战，这两个挑战都与强化学习（RL）环境有关。第一个挑战是奖励函数的泛化。这已经在社区中进行了讨论。例如，利用推理模型增强理解高层次自然指令的能力，像 Constitutional AI (Bai et al., 2022) 这样的方法可以直接用自然语言定义奖励函数。另一种策略则集中在提高编码能力，并将其他任务转化为编码问题来解决。

另一个较少提及的挑战是规划期间的环境状态更新。与不进行规划的经典无模型 RL 方法（如 Q 学习）不同，这些方法不显式建模状态转换，o1 类规划模型依赖于行为模拟和前向搜索，需要了解执行动作后的更新状态。这将范式转向基于模型的 RL。幸运的是，在编程、数学和围棋等定义明确的任务中，环境动态通常是确定性的。例如，围棋和其他棋盘游戏的世界模型可以通过规则显式描述。对于编程和数学，大型语言模型本身就嵌入了关于编程语法和公理逻辑的世界模型。这些确定性的环境动态允许在执行特定动作后精确计算状态转换概率。

然而，在许多实际应用中，如设备使用 (Wang et al., 2024b;a) 和具身代理，获取状态更新需要与外部环境或模拟器进行交互。这引入了显著的计算和时间成本。例如，在设备使用中，点击、输入或滚动等行为必须以涉及页面渲染、状态更新以及有时复杂的后端交互（如网络请求）的方式进行模拟。此外，o1 类模型在推理过程中无法进行在线行为模拟，这使得模型无法通过返回到先前状态来验证或纠正其动作。这导致无法回溯和优化决策。

因此，一个关键的方向是尝试通过开发用于状态转换预测的世界模型来显式建模环境。世界模型以当前和过去的状态以及动作为输入，输出下一个状态。这使得代理可以与其内部

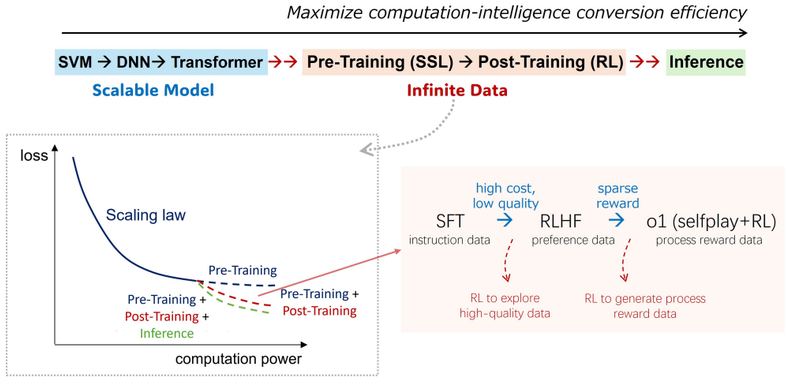

图 6: The trend towards maximizing computation-intelligence conversion efficiency.

This progression suggests that, with Transformer architectures now capable of scaling to handle vast amounts of data and training models of sufficient size, the only remaining challenge converges to acquiring adequate data. One approach is to collect data wherever it is lacking, such as reasoning data for system-2 abilities or physical world trajectories for embodied intelligence. Another approach is to explore data types that do not yet exist in the human world, which requires further exploration of techniques like RL and Self-Play.

4.2 Sweet Lesson: Beyond Human Data

A common criticism of LLM is its reliance on existing human-recorded data, which inherently limits its potential. As Wittgenstein stated, "The limits of my language mean the limits of my world." The finite scope and depth of human language records constrain the cognitive capabilities of LLMs. However, the success of o1 demonstrates that we can now explore the underlying thought processes behind these recorded data through RL. This advancement signifies a pivotal shift in AI development, moving from mere imitation of human language to the autonomous generation of novel cognitive processes.

More interestingly, these thought process data do not necessarily be confined to natural language. As highlighted in a recent Nature paper, "language serves primarily as a tool for communication rather than the essence of thought." (Fedorenko et al., 2024) In our observations, some of the thought chains generated by o1 contain nonsensical text, suggesting that the thinking tokens may not correspond to discrete natural language words. If the model has developed itself a more efficient form of internal representation for thinking, this will significantly elevate the efficiency of thought processes and problem-solving mechanisms, not only transcending the limitations imposed by human language data but also further unlocking the potential of model capabilities.

4.3 Opportunities: System1 + X to System2 + X

The self-play RL framework provides a viable solution for exploring underlying data, which opens up the possibility of exploring System-2 solutions for many tasks that were previously reliant on System 1 capabilities. By integrating more thoughtful, step-by-step processes into task execution,

世界模型进行交互，而不是直接与真实环境或模拟器进行交互。然而，由于准确构建这样的世界模型非常困难，世界模型通常仅应用于动态相对简单且理解良好的环境。好消息是，最近在交互内容生成 (Parker-Holder et al., 2024) 和生成游戏 (Sang, 2024) 方面的快速发展提供了有希望的进展，这可能有助于更准确和实用的环境建模，以支持实际应用中的基于规划的推理。

Prospects. o1 模型显然受到了 AlphaGo 的影响：AlphaGo 利用了模仿学习来初始化策略网络，强化学习来微调策略和学习价值网络，以及 MCTS 作为在线搜索策略，这与 LLM 的预训练、后训练和推理过程相平行。AlphaGoZero 采取了更先进的方法，不依赖历史数据，这正好反映了当前 LLM 开发中越来越重视后训练阶段的趋势。如果我们跟随它们后续的发展，可以预期 o1 类推理模型也会有类似的进展。

在 AlphaGoZero 之后，Alpha 系列首先朝着通用化方向发展：AlphaZero 被应用于围棋、国际象棋和将棋。为了进一步处理 Atari 视频游戏中的更复杂场景，MuZero 需要一个专门的模型来处理状态转换。其方法涉及同时更新世界模型和奖励模型，使模型能够在潜在空间中进行基于模型的规划，而不是依赖于显式的环境观察。类似地，选择紧凑的状态表示并构建有效的世界模型以支持高效的规划，是将 o1 类模型应用于现实场景中解决长期推理任务的关键。有趣的是，就在 o1 发布后的一周，由 OpenAI 支持的机器人公司 1X 揭示了其世界模型项目。这一计划旨在开发一个预测框架，以模拟和预测现实环境中行动的结果。它高度预见了 o1 类推理模型在推进具身智能方面的潜在应用。

Acknowledgements

我们感谢王宇航和张静的有益讨论和参与。

参考文献

Open o1: A model matching proprietary power with open-source innovation. https://github.com/Open-Source-O1/Open-O1/, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Benjamin Klieger. g1: Using Llama-3.1 70b on Groq to create o1-like reasoning chains. https://github.com/bklieger-groq/g1, 2024.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Hyung Won Chung. Don't teach. incentivize: Scale-first view of large language models, 2024.

we believe that this approach can yield positive results across a wide range of domains (Kant et al., 2024; Ganapini et al., 2021; Valmeekam et al., 2024; Lowe, 2024). Tasks traditionally solved using System 1 capabilities, such as reward modeling (Mahan et al., 2024), machine translation (Zhao et al., 2024), retrieval-augmented generation (RAG) (Li et al., 2024), and multimodal QA (Islam et al., 2024), have already benefited from the deeper reasoning capabilities enabled by System-2 thinking.

The o1 model's system card demonstrates notable improvements in model safety. Inspired by this, we have recently explored the concept of *System-2 Alignment*, which involves guiding models to thoroughly evaluate inputs, consider potential risks, and correct biases in their reasoning (Wang & Sang, 2024). We introduced three methods to realize System-2 alignment: prompting, supervised fine-tuning, and reinforcement learning with process supervision. We are applying the Self-Play+RL framework presented in this report to System-2 alignment, aiming to further enhance the model's ability to think deliberately and reduce vulnerabilities in complex scenarios.

4.4 Challenges: World Model Encoding

The released o1-preview and o1-mini currently lack multimodal capabilities and functional call features, which are claimed by OpenAI to be included in its complete version. Beyond multimodal and functional call, another critical feature for improvement in o1-like inference models is the optimization of inference time. This includes enhancing inference efficiency—achieving higher performance per unit of time—and enabling adaptive inference time adjustments. Specifically, this involves dynamically adjusting the System 2 reasoning process based on task complexity and achieving a more human-like ability to seamlessly switch between System 1 and System 2 reasoning modes.

For o1-like inference models to be deployed across broader real-world applications, two major challenges need to be addressed, both involving with the RL environments. The first challenge concerns reward function generalization. This has been already discussed in the community. For example, leveraging the enhanced ability of inference models to understand high-level natural instructions, approaches like Constitutional AI (Bai et al., 2022) might directly define reward functions in natural language. Alternative strategy focuses on improving coding capability and transforming the other tasks into coding problems for resolution.

Another less mentioned challenge concerns environment state update during planning. Unlike classic model-free RL methods without planning, such as Q-learning, where state transitions are not explicitly modeled, o1-like planning models rely on behavior simulation and forward search, requiring knowledge of the updated state following an action. This shifts the paradigm towards model-based RL. Fortunately, in well-defined tasks such as programming, mathematics, and Go, the environment dynamics are often deterministic. For example, the world models of Go and other board games can be described explicitly through rules. For programming and mathematics, large language models inherently embed their world models regarding programming syntax and axiomatic logic. These deterministic environment dynamics allow precise computation of state transition probabilities following specific actions.

However, in many real-world applications, such as device use (Wang et al., 2024b;a) and embodied agents, obtaining state updates requires interaction with external environments or simulators. This introduces significant computational and time costs. For example, in device use, behaviors like clicking, inputting, or scrolling must be simulated in a way that involves page rendering, state updates, and sometimes complex backend interactions like network requests. Moreover, o1-like

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf.

Evelina Fedorenko, Steven T. Piantadosi, and Edward A. Gibson. Language is primarily a tool for communication rather than thought. *Nature*, 615:75–82, 2024.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

GAIR-NLP. O1 replication journey: A strategic progress report, 2024.

Marianna Bergamaschi Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jon Lenchner, Andrea Loreggia, Nicholas Mattei, Francesca Rossi, Biplav Srivastava, and Kristen Brent Venable. Thinking fast and slow in ai: the role of metacognition, 2021.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Mohammed Saidul Islam, Raian Rahman, Ahmed Masry, Md Tahmid Rahman Laskar, Mir Tafseer Nayeem, , and Enamul Hoque. Are large vision language models up to the challenge of chart comprehension and reasoning? *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024(November):3334–3368, 2024.

Yichao Ji. A small step towards reproducing openai o1: Progress report on the steiner open source models, October 2024. URL https://medium.com/@peakji/b9a756a00855.

Manuj Kant, Marzieh Nabi, Manav Kant, Preston Carlson, and Megan Ma. Equitable access to justice: Logical llms show promise. *arXiv preprint arXiv:2410.09904*, 2024.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Huayang Li, Pat Verga, Priyanka Sen, Bowen Yang, Vijay Viswanathan, Patrick Lewis, Taro Watanabe, and Yixuan Su. Alr2: A retrieve-then-reason framework for long-context question answering. *arXiv preprint arXiv:2410.03227*, 2024.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Scott C. Lowe. System 2 reasoning capabilities are nigh, 2024.

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models, 2024.

models face the limitation of not being able to perform online behavior simulation during inference, which prevents the model from validating or correcting its actions by returning to a previous state. This leads to inability to backtrack and refine decisions.

Therefore, one of the key directions is to attempt explicit modeling of the environment by developing a world model for state transition prediction. The world model takes as input the current and past states as well as actions, and produces the next state as output. This allows the agent to interact with its internal world model, rather than directly with the real environment or a simulator. However, since accurately building such world models is very difficult, world models have typically been applied to environments where the dynamics are relatively simple and well-understood. The good news is, the recent rapid advancements in interactive content generation (Parker-Holder et al., 2024) and generative games (Sang, 2024) offer promising progress that could facilitate more accurate and practical environment modeling for planning-based reasoning in real-world applications.

Prospects. The o1 model is clearly influenced by AlphaGo: AlphaGo utilized imitation learning to initialize the policy network, reinforcement learning to fine-tune the policy and learn the value network, and MCTS as an online search strategy, which parallels LLM's pre-training, post-training, and inference. AlphaGoZero took a more advanced approach by not relying on historical data, which exactly mirrors current trends in LLM development increasingly emphasizing the post-training stage. If we follow their subsequent evolution, we can anticipate similar developments in o1-like reasoning models.

After AlphaGoZero, the Alpha series first developed towards generalization: AlphaZero was applied to Go, Chess, and Shogi. To further tackle more complex scenarios in Atari video games, MuZero requires a dedicated model to handle state transitions. Its approach involves simultaneously updating a world model and a reward model, enabling model-based planning in a latent space rather than relying on explicit environmental observations. In analogy, selecting a compact state representation and constructing an effective world model to support efficient planning are key to applying o1-like models in real-world scenarios for solving long-horizon reasoning tasks. Interestingly, just a week after the release of o1, 1X, the robotics company backed by OpenAI, unveiled its world model project. This initiative aims to develop a predictive framework to simulate and anticipate the outcomes of actions in real-world environments. It highly envisions the potential applications of o1-like reasoning models in advancing embodied intelligence.

ACKNOWLEDGEMENTS

We thank Yuhang Wang and Jing Zhang for their fruitful discussions and participation.

参考文献

Open o1: A model matching proprietary power with open-source innovation. https://github.com/Open-Source-O1/Open-O1/, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

OpenAI. Learning to reason with large language models. https://openai.com/index/learning-to-reason-with-llms/, 2024.

Jack Parker-Holder, Stephen Spencer, Philip Ball, Jake Bruce, Vibhavari Dasagi, Kristian Holsheimer, Christos Kaplanis, Alexandre Moufarek, Guy Scully, Jeremy Shar, Jimmy Shi, Jessica Yung, Michael Dennis, Sultan Kenjeyev, Shangbang Long, Yusuf Aytar, Jeff Clune, Sander Dieleman, Doug Eck, Shlomi Fruchter, Raia Hadsell, Demis Hassabis, Georg Ostrovski, Pieter-Jan Kindermans, Nicolas Heess, Charles Blundell, Simon Osindero, Rushil Mistry, et al. Genie 2: A large-scale foundation world model. https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/, 2024.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*, 2024.

Qwen Team. QwQ-32b-preview. https://qwenlm.github.io/zh/blog/qwq-32b-preview/, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Richards Tu. Thinking Claude. https://github.com/richards199999/Thinking-Claude/tree/main, 2024.

Jitao Sang. A note on generative games: Positioning, progress and prospects. *arXiv*, 2024.

Shanghai AI Lab. InternThinker. https://internlm-chat.intern-ai.org.cn, 2024.

SimpleBerry. Llama-o1: Open large reasoning model frameworks for training, inference and evaluation with pytorch and huggingface. https://github.com/SimpleBerry/LLaMA-O1, 2024. Accessed: 2024-11-25.

OpenR Team. Openr: An open source framework for advanced reasoning with large language models. https://github.com/openreasoner/openr, 2024.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of lrm o1, 2024.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024a.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024b.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439,

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Benjamin Klieger. g1: Using Llama-3.1 70b on Groq to create o1-like reasoning chains. https://github.com/bklieger-groq/g1, 2024.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Hyung Won Chung. Don't teach. incentivize: Scale-first view of large language models, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf.

Evelina Fedorenko, Steven T. Piantadosi, and Edward A. Gibson. Language is primarily a tool for communication rather than thought. *Nature*, 615:75–82, 2024.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

GAIR-NLP. O1 replication journey: A strategic progress report, 2024.

Marianna Bergamaschi Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jon Lenchner, Andrea Loreggia, Nicholas Mattei, Francesca Rossi, Biplav Srivastava, and Kristen Brent Venable. Thinking fast and slow in ai: the role of metacognition, 2021.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Mohammed Saidul Islam, Raian Rahman, Ahmed Masry, Md Tahmid Rahman Laskar, Mir Tafseer Nayeem, , and Enamul Hoque. Are large vision language models up to the challenge of chart comprehension and reasoning? *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024(November):3334–3368, 2024.

Yichao Ji. A small step towards reproducing openai o1: Progress report on the steiner open source models, October 2024. URL https://medium.com/@peakji/b9a756a00855.

Manuj Kant, Marzieh Nabi, Manav Kant, Preston Carlson, and Megan Ma. Equitable access to justice: Logical llms show promise. *arXiv preprint arXiv:2410.09904*, 2024.

Bangkok, Thailand, August 2024c. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510.

Yuhang Wang and Jitao Sang. Don't command, cultivate: An exploratory study of system-2 alignment, 2024.

Guowei Xu, Peng Jin, Li Hao, Yibing Song, Lichao Sun, and Li Yuan. Llava-o1: Let vision language models reason step-by-step, 2024. URL https://arxiv.org/abs/2411.10440.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Eric Zelikman et al. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024. URL https://arxiv.org/abs/2403.09629.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *arXiv preprint arXiv:2410.02884*, 2024. URL https://arxiv.org/abs/2410.02884. Accessed: 2024-11-25.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. Marco-o1: Towards open reasoning models for open-ended solutions, 2024.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Huayang Li, Pat Verga, Priyanka Sen, Bowen Yang, Vijay Viswanathan, Patrick Lewis, Taro Watanabe, and Yixuan Su. Alr2: A retrieve-then-reason framework for long-context question answering. *arXiv preprint arXiv:2410.03227*, 2024.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Scott C. Lowe. System 2 reasoning capabilities are nigh, 2024.

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models, 2024.

OpenAI. Learning to reason with large language models. `https://openai.com/index/learning-to-reason-with-llms/`, 2024.

Jack Parker-Holder, Stephen Spencer, Philip Ball, Jake Bruce, Vibhavari Dasagi, Kristian Holsheimer, Christos Kaplanis, Alexandre Moufarek, Guy Scully, Jeremy Shar, Jimmy Shi, Jessica Yung, Michael Dennis, Sultan Kenjeyev, Shangbang Long, Yusuf Aytar, Jeff Clune, Sander Dieleman, Doug Eck, Shlomi Fruchter, Raia Hadsell, Demis Hassabis, Georg Ostrovski, Pieter-Jan Kindermans, Nicolas Heess, Charles Blundell, Simon Osindero, Rushil Mistry, et al. Genie 2: A large-scale foundation world model. `https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/`, 2024.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195*, 2024.

Qwen Team. QwQ-32b-preview. `https://qwenlm.github.io/zh/blog/qwq-32b-preview/`, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Richards Tu. Thinking Claude. `https://github.com/richards199999/Thinking-Claude/tree/main`, 2024.

Jitao Sang. A note on generative games: Positioning, progress and prospects. *arXiv*, 2024.

Shanghai AI Lab. InternThinker. `https://internlm-chat.intern-ai.org.cn`, 2024.

SimpleBerry. Llama-o1: Open large reasoning model frameworks for training, inference and evaluation with pytorch and huggingface. `https://github.com/SimpleBerry/LLaMA-O1`, 2024. Accessed: 2024-11-25.

OpenR Team. Openr: An open source framework for advanced reasoning with large language models. `https://github.com/openreasoner/openr`, 2024.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of lrm o1, 2024.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024a.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024b.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, August 2024c. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510.

Yuhang Wang and Jitao Sang. Don't command, cultivate: An exploratory study of system-2 alignment, 2024.

Guowei Xu, Peng Jin, Li Hao, Yibing Song, Lichao Sun, and Li Yuan. Llava-o1: Let vision language models reason step-by-step, 2024. URL https://arxiv.org/abs/2411.10440.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Eric Zelikman et al. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024. URL https://arxiv.org/abs/2403.09629.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *arXiv preprint arXiv:2410.02884*, 2024. URL https://arxiv.org/abs/2410.02884. Accessed: 2024-11-25.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. Marco-o1: Towards open reasoning models for open-ended solutions, 2024.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.