A Tutorial on LLM Reasoning:
Relevant methods behind ChatGPT o1

Jun Wang
jun.wang@cs.ucl.ac.uk
UCL Centre for Artificial Intelligence

Abstract

OpenAI o1 has shown that applying reinforcement learning to integrate reasoning steps directly during inference can significantly improve a model's reasoning capabilities. This result is exciting as the field transitions from the conventional autoregressive method of generating answers to a more deliberate approach that models the slow-thinking process through step-by-step reasoning training. Reinforcement learning plays a key role in both the model's training and decoding processes. In this article, we present a comprehensive formulation of reasoning problems and investigate the use of both model-based and model-free approaches to better support this slow-thinking framework.
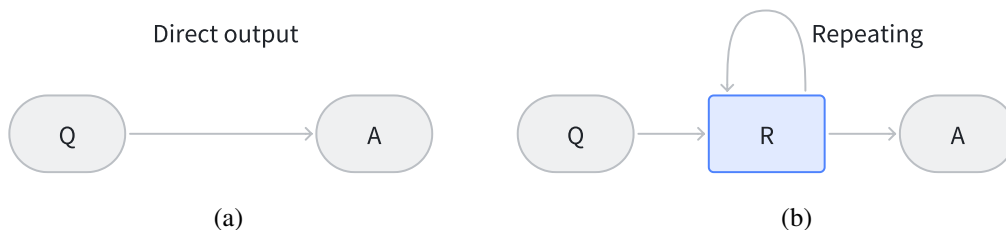
Figure 1: Inference-time computation. (a) An autoregressive LLM directly generate an answer (A) by conditioning on the given question (Q). (b) The concept of chain of thought, or step-by-step thinking, involves incorporating intermediate reasoning steps (R) before arriving at the final answer (A). These repeated operations allow for 1) revisiting and revising prior outputs, 2) progressing to subsequent reasoning stages, and 3) exploring multiple reasoning paths or trajectories.

1 Background

OpenAI has recently unveiled ChatGPT o1 [17], a groundbreaking Large Language Model (LLM) that represents a giant leap forward in strong AI. Trained using reinforcement learning techniques, o1 excels in complex reasoning tasks by explicitly embedding a *native* "Chain-of-Thought" (NCoT) process, which allows it to "deep think" through step-by-step reasoning before generating responses. The model is reported to be five times more proficient in math and coding compared to the previous ChatGPT 4o, specifically displaying exceptional performance across various domains: it ranks in the 89th percentile for competitive programming, places among the top 500 students in a prestigious US math olympiad qualifier, and surpasses human PhD-level accuracy in physics, biology, and chemistry benchmarks. A key innovation of o1 is that it allows spending more time reasoning during the inference process, marking a shift from fast, direct responses to slow, deliberate, multi-step inference-time computation (Fig. 1).

Interestingly, in human cognition, two correlated yet distinct modes of cognitive processing are presented to guide human decision-making and behaviours [8], each of which has partially distinction

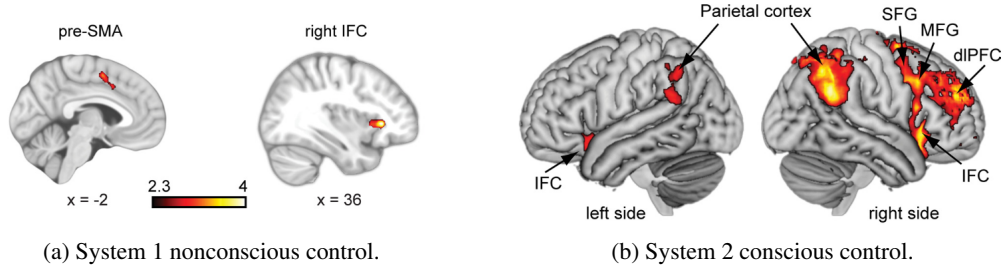(a) System 1 nonconscious control.　　　　(b) System 2 conscious control.

Figure 2: An analogy between human cognition and LLMs. (a) and (b) human actions controlled consciously or unconsciously rely on partially distinct brain circuits. (a) Unconscious control in humans is maintained by a few specialised brain regions, such as the anterior insula and the presupplementary motor area (pre-SMA). (b) while voluntary control engages a broader network, activating many regions within the parietal and prefrontal lobes [28]. Unconscious control is typically fast and instinctive, often driven by automatic processes, whereas conscious control tends to involve more deliberate, computational, and in-depth thinking, allowing for careful reflection and thorough analysis.

brain circuits and neural pathways (Fig. 2 and also see [28]). System 1 thinking is fast, automatic, and intuitive, operating effortlessly and often unconsciously. It relies on neural pathways that enable rapid processing, especially in situations needing quick reactions or when cognitive resources are constrained. System 2 thinking is deliberate, effortful, and conscious, involving focused attention and analytical reasoning. It processes information more slowly and is used for complex problem-solving, logical reasoning, and decision-making tasks. o1 is an exciting development for AI, as LLMs can now not only generate rapid responses using learned patterns but, more significantly, simulate complex reasoning processes through mechanisms like chain of thought or other forms of search, similar to how humans engage in deeper, step-by-step thinking[1].

ChatGPT o1's improved reasoning skills have many implications for multiple fields, including science, coding, and mathematics. In coding competitions, a specialised version of o1 achieved impressive results, scoring in the 49th percentile in the 2024 International Olympiad in Informatics and outperforming 93% of human competitors in simulated Codeforces contests. Beyond its technical capabilities, o1 also represents progress in AI safety and alignment. The model's chain of thought reasoning provides new opportunities for integrating human values and principles, resulting in improved performance on safety evaluations and jailbreak tests.

The idea of chain of thought reasoning and step-by-step thinking in Large Language Models (LLMs) is not new. Previous research has shown that simply adding instructions like "describe your reasoning in steps" or "explain your answer step by step" to the input questions or providing few shot examples can trigger LLMs to generate intermediate reasoning steps (as illustrated in Fig. 1) and subsequently improve problem-solving, especially in tasks like math and coding [32, 16]. However, these approaches build on existing LLMs without truly embedding the chain of thought ability within the models themselves. As a result, LLMs cannot inherently learn this reasoning capability, leading to active research on how to integrate it directly into model training. Proposed methods range from collecting specialised training data to building reward models [18, 11, 15] and increasing the computational complexity of decoding [24, 33], but none have yet achieved significant performance breakthroughs at scale.

It remains unclear whether OpenAI's o1 innovation is rooted in the model itself, rather than relying on external prompting systems. If it indeed involves explicitly embedding step-by-step reasoning natively within the architecture, this would represent a significant breakthrough. Building on substantial performance gains, OpenAI o1 has shown that the scaling principles traditionally applied during training [9, 24] are now relevant to the inference phase. We should reallocate our computational focus, balancing pre-training efforts with efficient use of inference-time computation.

---

[1]It is important to note that incorporating chain-of-thought processes in AI does not imply human-like consciousness. Instead, these mechanisms enhance reasoning and problem-solving by breaking tasks into manageable steps without suggesting any form of self-awareness or subjective experience.

Allowing LLMs to enhance their outputs with increased test-time computing is an essential step towards creating generally self-improving agents capable of managing open-ended strong reasoning and decision-making tasks. This direction, which we refer to as LLM-Native Chain-of-Thought (NativeCoT), should be able to inherently mirror the deliberate, analytical process possessed by human's System 2 thinking [8].

Given that o1 is a closed-source system, the precise techniques used to achieve such strong reasoning capabilities remain largely a mystery. In this article, we will provide a comprehensive overview of the relevant literature and offer insights into what we believe are the core techniques and methods underpinning this breakthrough. Additionally, we will propose our ideas for implementing an open-source counterpart, which could accelerate research in this area. Our proposals will draw inspiration from recent work, including ours on data acquisition, reinforcement learning based training, and search and MCTS-based decoding for improving reasoning capabilities in existing models.

In the next section, we will discuss two challenges commonly encountered by typical autoregressive LLMs, highlighting the need for a world model and a chain-of-thought mechanism. We will then present an MDP formulation for incorporating native CoT within LLMs (resulting in o1-like reasoning models) and explore its implementation details. Finally, we conclude with bibliographic remarks and suggest future research directions.

2 The Challenges with Autoregressive LLMs

Autoregressive language models (LLMs) generate sequences of text by predicting the next token (e.g., word) in the sequence given the previous tokens [29]. Mathematically, they are based on the principle of conditional probability. The task is to model the joint probability of a sequence of tokens $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, where T is the length of the sequence, by factorising it into a product of conditional probabilities using the chain rule of probability.

Given a sequence of tokens $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, an autoregressive language model estimates the joint probability $P(\mathbf{x})$ as:

$$P(\mathbf{x}) = P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, x_2, \ldots, x_{t-1}),$$

where the model predicts the probability of each token x_t based on all preceding tokens in the sequence $x_1, x_2, \ldots, x_{t-1}$. Typically, this is achieved using neural networks like transformers [29], which are trained to minimise the negative log-likelihood of the training data. For an explanation of the training steps, please refer to Appendix A.

At inference time, the model generates text by typically sampling tokens sequentially from the probability distribution $P(x_t \mid x_1, x_2, \ldots, x_{t-1})$ until a stop token is reached or a predefined maximum length is achieved. The model works as follows: Firstly, start with a given sequence or a start token (if generating from scratch). Secondly, at each step t, predict the next token x_t based on the previously generated tokens $(x_1, x_2, \ldots, x_{t-1})$. At last, continue sampling until the sequence is complete. For a simple three-token sequence $\mathbf{x} = (x_1, x_2, x_3)$, the probability of the sequence would be:
$$P(\mathbf{x}) = P(x_1) \cdot P(x_2 \mid x_1) \cdot P(x_3 \mid x_1, x_2).$$

This formulation underpins the operation of autoregressive LLMs like GPT-style models. The learning is achieved by minimising mistakes in predicting subsequent tokens (words). The first challenges is this *predicting next tokens* objective. While some people propose that predicting next tokens might pave the way for general intelligence (AGI), we intend to argue is that solely focusing on predicting the next word caps the potential for intelligence. A different optimisation target and learning paradigm might be necessary to foster deeper intelligence.

To illustrate the limitations of purely predictive models, let's consider the domain of chess mastery. In this context, each chess move can be conceptualised as a token, with a complete chess representing a "sentence" in the "language of chess" - a sequence of moves from the opening to the endgame. Suppose we have access to an extensive dataset of chess games, but all from players with Elo ratings below 2000 (a standardised measure of player skill) [5]. If we train a chess agent solely by minimising token prediction errors based on these games, we would likely constrain the agent's performance

to within the ability range of these sub-2000 Elo players. This approach would essentially optimise the agent towards emulating the average or typical play of these players, potentially incorporating their mistakes and suboptimal strategies. This phenomenon can be characterised as what we called an "intelligence upper bound," a concept that can be rigorously derived from recent research in offline reinforcement learning and imitation learning [10]. The agent, in this case, is limited by the quality of the demonstrations it learns from, unable to surpass the skill level present in its training data. This limitation underscores a crucial challenge in AI development: how to enable systems to transcend the boundaries of their training data and develop novel, potentially superior strategies.

Conversely, when data is leveraged to develop a deeper understanding, or a *world model*, of chess dynamics, it may pave the way for the evolution of sophisticated strategies and tactics that go beyond mere imitation of behaviours observed in the training data. A world model presents the agent's understanding of the environment, in this case, the chess rules, i.e., how a move would change the status of the game and what the winning chance of a given move is. Learning and refining this world model, coupled with the ability to simulate potential outcomes, could potentially empower an AI agent to surpass the 2000 Elo benchmark. The simulation capabilities afforded by these internal world models would enable deep thinking (simulation), thereby enhancing the agent's reasoning and generalisation capabilities. Model-based strategies like Monte Carlo Tree Search (MCTS) serve as classic illustrations of this approach [23]. The transition to System 2 type reasoning, as potentially exemplified by ChatGPT o1, likely relies on establishing a certain type of World Model and utilising reinforcement learning (reward maximisation) rather than solely minimising prediction errors. This shift in approach may be one of the key transitional techniques behind ChatGPT o1's enhanced reasoning capabilities.

By combining the predictive power of large language models with the strategic depth of reinforcement learning and World Modelling, AI systems like o1 can potentially engage in more sophisticated problem-solving and decision-making processes. This hybrid approach allows for both rapid pattern recognition (akin to System 1 thinking) and deliberate, step-by-step reasoning (characteristic of System 2 thinking), potentially explaining the significant leap in performance observed in o1.

The second challenge, from a computational complexity perspective, is that Large Language Models (LLMs) inherently operate within the constraints of quadratic computational complexity [13]. This limitation becomes particularly apparent when LLMs encounter multi-step mathematical challenges. However, the "chain of thoughts" concept offers a potential mitigation to this constraint [32]. It extends responses through a series of "thought" outputs, therefore allows a certain amount of additional computation resources; it essentially acts as a *limited memory* that supports writing but lacks the capacity for deletion or overwriting. While this approach has shown promise, it still falls short of a fully dynamic memory system and is not natively incorporated into the decoding stage. This necessity underscores the demand for advanced computational architectures that transcend the capabilities of current transformer decoder networks. Indeed, there is a need to implement sophisticated model-based strategies akin to Monte Carlo Tree Search (MCTS) witnin the inference and decoding stage [6].

Such an advanced inference-time computation system would enable AI models to maintain and dynamically update a representation of the problem space, facilitating more complex reasoning processes. This approach [3] aligns with the concept of working memory in cognitive science, which is crucial for complex problem-solving and deliberative thinking. By integrating these capabilities, AI systems could potentially simulate multiple steps ahead, evaluate different scenarios, and make more informed decisions — mirroring the deliberative processes observed in human expert reasoning.

3 LLM Reasoning as a Markov Decision Process

To model the process of reasoning in tasks such as question answering or problem solving, we structure the reasoning task using the $Q \rightarrow \{R\} \rightarrow A$ sequence, where:

- **Q:** Represents the question or prompt that initiates the reasoning process.

- **R:** Represents the sequence of intermediate reasoning steps the model generates to build toward the solution.

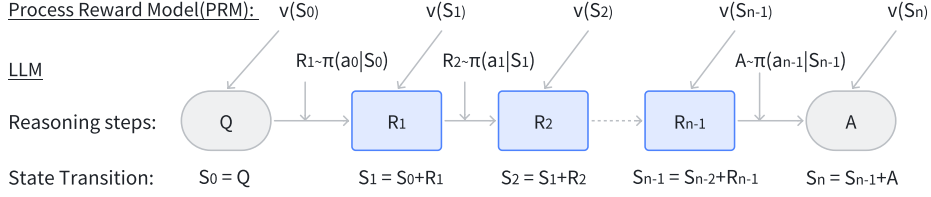- **A:** Represents the final answer or solution produced after the reasoning steps.

Figure 3: In this MDP formulation, the LLM is tasked with generating reasoning steps and the final answer to a question in a step-by-step manner. The LLM policy operates by generating tokens, which form higher-level reasoning constructs. The states represent the sequence of reasoning steps so far, and actions correspond to the selection of new reasoning steps or the final answer. The LLM policy governs the choice of actions, and the process-reward model (PRM) provides feedback on the quality of reasoning steps and the final answer. By optimising the policy to maximise the reward, the LLM can be guided by PRM to generate accurate and meaningful reasoning processes.

This structure allows the LLM to generate a sequence of reasoning steps that logically connect the question Q to the final answer A.

We can define the reasoning process as a Markov Decision Process (MDP) [1]. A MDP representation offers a flexible framework for modelling reasoning. It allows the model to autoregressively generate sequential reasoning steps toward the final answer, while also enabling a tree structure by sampling multiple paths at each step for alternative reasoning trajectories. By combining both approaches-sequential and branching reasoning-the model can explore diverse solutions, creating a versatile and comprehensive reasoning process.

We are now ready to describe the reasoning process in terms of states, actions, policies, and rewards, where the LLM's task is to incrementally generate a coherent sequence of tokens that correspond to reasoning steps and the final answer.

The state s_t at timestep t represents the current state of the reasoning process, including the question and the reasoning steps generated so far. Formally, the state is defined as:

$$s_t = (Q, R_1, \ldots, R_{t-1}),$$

where Q is the initial question or prompt, and $R_1, \ldots, R_{t-1}$ are the reasoning steps generated up to timestep t. The initial state s_0 contains just the question:

$$s_0 = Q.$$

As reasoning progresses, the intermediate states include both the question and the reasoning steps generated so far. The process continues until the final answer is generated.

An action $a_t \in A$ at timestep t corresponds to the selection of the next reasoning step or the final answer. The action space A consists of two types of actions:

- **Reasoning Step (R):** The action selects a reasoning step R_t to append to the current state.
- **Final Answer (A):** The action selects the final answer A, which concludes the reasoning process.

For intermediate steps, the action is:

$$a_t = R_t,$$

and the new state becomes:

$$s_{t+1} = s_t + R_t.$$

For the final step, the action selects the final answer:

$$a_T = A,$$

and the final state becomes:

$$s_T = s_{T-1} + A.$$

5

The policy π defines the strategy the model uses to choose the next action (i.e., reasoning step or final answer) given the current state. The policy is essentially the LLM, learned during training and represents the probability distribution over possible reasoning steps or the final answer, conditioned on the tokens generated so far:

$$\pi_{LLM}(a_t \mid s_t) = P(a_t \mid Q, R_1, \ldots, R_{t-1}).$$

At each timestep, the model uses this policy to select the next action based on the current state, incrementally building towards the final answer.

Given the autoregressive nature of the LLM, the transition from one state to the next is deterministic and also given. The next state s_{t+1} is fully determined by appending the selected action a_t (a reasoning step or the final answer) to the current state s_t. Therefore, the transition function is:

$$s_{t+1} = s_t + a_t.$$

This means that once a reasoning step R_t or final answer A is selected, the state s_{t+1} is uniquely defined by concatenating this action to the existing sequence of tokens.

The reward provides feedback on the quality of the generated reasoning steps and the final answer. In this context, the reward is obtained as the model generates reasoning steps and the final answer. The rewards can be defined as:

- **Intermediate Reward:** For generating correct or meaningful reasoning steps, intermediate rewards are assigned positive values. Incorrect or irrelevant steps may yield negative rewards.
- **Final Reward:** The largest reward is given when the model generates the correct final answer A, completing the reasoning process.

Thus, the reward at each timestep t is:

$$v_t = v(\mathrm{R}_t \mid Q, R_1, \ldots, R_{t-1}),$$

and for the final step:

$$v_T = v(A \mid Q, R_1, \ldots, R_n).$$

The model learns to optimise its policy to maximise the cumulative expected reward over the entire reasoning process.

Relationship Between Token Generation and Reasoning The LLM operates at two levels simultaneously: the level of token generation and the level of reasoning steps and final answers. At the most granular level, the LLM generates tokens autoregressively, meaning it generates one token at a time, conditioned on the previously generated tokens:

$$P(x_t \mid x_1, x_2, \ldots, x_{t-1}).$$

At each timestep t, the LLM generates a token x_t from its vocabulary based on the context provided by previous tokens. These tokens form higher-level constructs such as reasoning steps R_t and the final answer A.

- **Reasoning Steps (R):** Each reasoning step R_t is composed of a sequence of tokens $\{x_{t_1}, x_{t_2}, \ldots, x_{t_k}\}$ generated by the LLM. These tokens represent a coherent step in the reasoning process, such as a logical deduction or intermediate conclusion.
- **Final Answer (A):** The final answer A is similarly composed of a sequence of tokens that form the solution or response to the question. Once the LLM has generated sufficient reasoning steps, it produces the final answer in an autoregressive manner, token by token.

We are now ready to a world model for LLMs exactly:

Definition 1 (World Model of LLM) *A* world model of LLM *is defined as* $(\mathcal{T}, \mathcal{R})$, *where:*

- *The transition model* $\mathcal{T}(s_t, a_t)$ *is deterministic as the next state* s_{t+1} *is uniquely defined by the current state* s_t *and the action* a_t *(i.e., the generated token or reasoning step), so:*
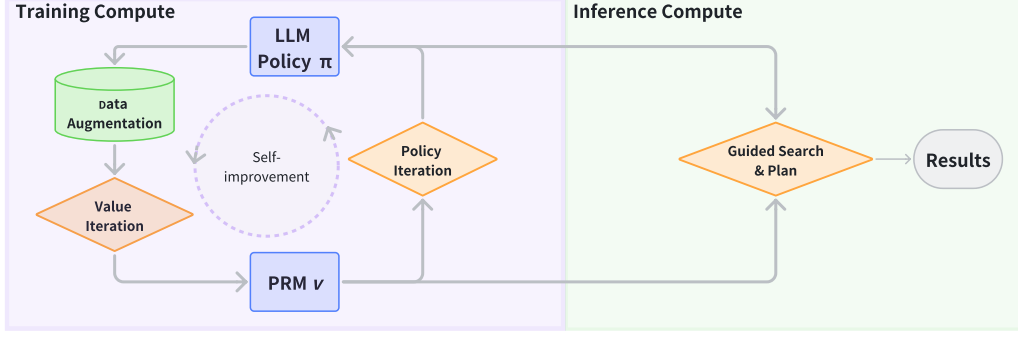
$$s_{t+1} = s_t + a_t.$$

Figure 4: Combining the value function from the PRM with the LLM's policy generation ensures guided and controlled results. During training, the generation produced by the LLM's policy and the evaluation provided by the PRM reinforce each other, leading to continuous self-improvement and refinement of both components.

- $\mathcal{V}(s_t, a_t)$ *is the process-reward model (PRM) that evaluates the quality of the action a_t taken in state s_t. It reflects how appropriate or effective the generated reasoning step or token is in progressing towards the final answer:*
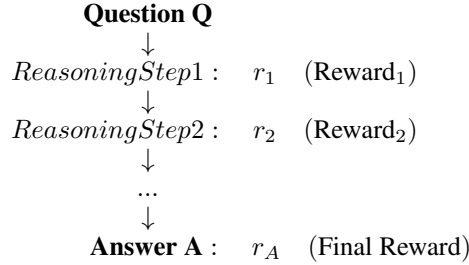
$$\mathcal{V}(s_t, a_t) = v_t.$$

Since the transition is deterministic and follows directly from the policy, the process-reward model (PRM) $\mathcal{R}(s_t, a_t)$ encapsulates the entire interaction between the LLM and its environment, evaluating how well each reasoning step or token contributes to reaching the final answer.

4 Practical Implementation

Next, we examine how to collect the intermediate reasoning data, use it to train the process-reward model (PRM), leverage the PRM to train the LLM policy, and guide the reasoning process during the decoding phase.

4.1 Automatic Acquisition of Reasoning Steps Data

As discussed, we require reasoning trajectories to stimulate advanced reasoning while covering a wide range of tasks. For fine-tuning a LLM, we typically have {Q and A} pairs, however lacking the ground-truth of underlying reasoning steps {R}:

$$\textbf{Question Q}$$
$$\downarrow$$
$$ReasoningStep1: \quad r_1 \quad (\text{Reward}_1)$$
$$\downarrow$$
$$ReasoningStep2: \quad r_2 \quad (\text{Reward}_2)$$
$$\downarrow$$
$$...$$
$$\downarrow$$
$$\textbf{Answer A}: \quad r_A \quad (\text{Final Reward})$$

A straightforward approach would be to label the reasoning steps manually by humans [27, 12]. However, a particularly effective method for collecting data and improving LLM reasoning without requiring human supervision is the Self-Taught Reasoner (STaR) technique [34], among others. In this approach, the model generates intermediate reasoning steps autonomously and uses them to validate its internal reasoning capabilities. This method builds on the ability of LLMs to reason from a question Q to a final answer A, by generating intermediate steps $\{R_1, R_2, \ldots, R_n\}$ and verifying their correctness using the model's own policy. Namely, the method begins by employing the LLM's

policy (may add few shot prompts), denoted π_{LLM}, to generate reasoning steps $\{R\}$ conditioned on the initial question Q and final answer A. This generation can be expressed as follows:

$$\{R\} \sim \pi_{\text{LLM}}(\cdot \mid Q, A),$$

where the LLM produces a sequence of intermediate reasoning steps $\{R_1, R_2, \ldots, R_n\}$ that aim to logically connect the question Q to the correct final answer A. These steps serve as a form of internal decomposition of the reasoning task, which is crucial for complex multi-step problems where direct question-answer pairs may be insufficient for training the model to reason effectively.

Once the intermediate reasoning steps $\{R\}$ are generated, the next phase involves verifying their correctness. This is achieved by using the LLM's policy again to check whether the reasoning steps, when combined with the original question Q, lead to the correct answer A. Formally, this verification step is represented by:

$$A' \sim \pi_{\text{LLM}}(\cdot \mid Q, \{R\}),$$

where, A' is the model's prediction of the answer based on the question Q and the generated reasoning steps $\{R\}$. If A' matches the original correct answer A, then the reasoning steps $\{R\}$ are considered valid. Thus, the correctness of $\{R\}$ is determined by the condition: $A' \approx A$. This self-validation mechanism enables the model to autonomously identify correct reasoning steps, reinforcing its internal logical consistency without external feedback.

The collected new reasoning steps $\{Q, \{R\}, A\}$ can be used to further train the LLM's policy π_{LLM}, reinforcing the generation of effective reasoning steps. This iterative process can be expressed as:

$$\pi_{\text{LLM}} \leftarrow \pi_{\text{LLM}} + \text{feedback from } \{Q, \{R\}, A\}.$$

For longer reasoning sequences, techniques such as Monte Carlo Tree Search (MCTS) [6, 15] are employed to guide the LLM policy to find correct reasoning steps efficiently in a more fine-grained manner. These tree-based methods help in finding optimal reasoning paths by exploring various possibilities and simulating multiple outcomes in each reasoning stage. This is particularly useful for complex tasks like math problem-solving and agent-based decision-making, where intermediate steps have multiple paths.

4.2 Self-reinforced Training

As illustrated in Fig. 4, the PRM $v(s)$ and LLM policy (π_{LLM}) can be mutually reinforced to improve themselves, which will be explained next.

4.2.1 Value Iteration for PRM

Once the reasoning data has been collected, the next step is to train the world model, also referred to as the Process-Reward Model (PRM), i.e., since the state transitions are deterministic and known, the focus shifts to learning a general reward model that can later be used to guide the search, reasoning, and decoding processes. This reward model, often called the verifier, denoted as $v_{\text{PRM}}(s)$, can be trained using a dataset of annotated reasoning steps. The training typically involves optimising a classification loss function based on the correctness of the reasoning steps [15]:

$$\mathcal{L}_{\text{PRM}} = \sum_{i=1}^{N} \left[\hat{v}_i \log v_i + (1 - \hat{v}_i) \log(1 - v_i) \right],$$

where $v_i = r_i$ represents the correctness label for the i-th example step, indicating whether the reasoning process for that example is correct. The verifier's prediction, $\hat{v}_i(s)$, is the score output by the PRM for the state s, representing the reward for the reasoning step or the final answer. Since this is a classification approach, there is no distinction between the reward for an intermediate step and the potential reward it could lead to and all the reasoning steps are assumed to be independent. The model simply evaluates whether the reasoning step or answer is correct at that point in the process, treating all rewards in a uniform manner without considering the future impact of intermediate steps.

However, an alternative approach involves viewing the PRM as a value function that can be trained via a value iteration method, enabling it to predict cumulative rewards and guide the reasoning process through optimal action selection [6]. Consider a reasoning process where the state s represents

the current reasoning state, incorporating all previous reasoning steps. The objective of the value iteration method is to learn a value function $V_\theta(s)$, parameterised by θ, that predicts the expected cumulative reward starting from state s. This value function guides the reasoning process by evaluating the potential outcomes of different actions. $r_\phi(s)$ – the reward function, which assigns a scalar reward to state s based on the correctness of intermediate reasoning steps or the final answer. γ is the discount factor, which determines the relative importance of future rewards. The Bellman equation [1] for the PRM is:

$$V_\theta(s) = r(s) + \gamma \max_a V_\theta(a + s),$$

where $s' = a + s$ is the next state reached by taking action a in state s. The reward function $r(s)$ can be sparse, providing rewards only for correct conclusions, or dense, providing partial rewards for intermediate steps. We define the TD loss function for learning the parameters θ of the value function as the squared error between the current value and the Bellman target:

$$L(\theta) = \sum_{i=1}^{N} \left(V_\theta(s_i) - \left[r(s_i) + \gamma \max_a V_\theta(s_i + a) \right] \right)^2.$$

We can then obtain the parameters θ of the value function by minimising the loss $L(\theta)$ using gradient descent or another optimisation technique.

4.2.2 Policy Iteration for LLM Policy

Once PRM obtained, one can train the LLM policy for enhanced reasoning. This requires methodologies that go beyond traditional supervised learning frameworks. PRM plays an essential role in this process by incorporating online reinforcement learning to optimise reasoning tasks [18]. However, a typical RLHF work such as [18] can be used but may not be ideal for large language model training.

Let us look at Group Relative Policy Optimisation (GRPO) [22]. We assume that for each question $Q = q$, the policy generates reasoning steps $\{o_1, o_2, \ldots, o_G\}$, and each output o_i consists of multiple steps $\{a_{i,1}, a_{i,2}, \ldots, a_{i,K_i}\}$, where K_i is the total number of reasoning steps (or tokens) in output o_i. We slightly abuse our previous notation by using o to represent all outputs, including both reasoning steps and final answers. We can now formulate the GRPO optimisation for learning the LLM policy via the PRM as follows.

For each question q, GRPO samples a group of outputs $\{o_1, o_2, \ldots, o_G\}$ from the old policy $\pi_{\theta_{\text{old}}}$, and the goal is to optimise the policy by maximising the following objective:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(O|q)} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{K_i} \sum_{t=1}^{K_i} \min \left(\hat{\rho}_{i,t} A_{i,t}, \text{clip} \left(\hat{\rho}_{i,t}, 1 - \epsilon, 1 + \epsilon \right) A_{i,t} \right) - \beta D_{\text{KL}} \left(\pi_\theta \| \pi_{\theta_{\text{ref}}} \right) \right],$$

where:

- $q \sim P(Q)$ denotes sampling a question q from a distribution of questions $P(Q)$,

- $\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(O|q)$ represents the group of outputs sampled from the old policy $\pi_{\theta_{\text{old}}}$,

- $\hat{\rho}_{i,t} = \frac{\pi_\theta(a_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(a_{i,t}|q, o_{i,<t})}$ is the importance weight (probability ratio) for action $a_{i,t}$ at step t in output o_i,

- $A_{i,t}$ is the advantage at reasoning step t of output o_i, calculated based on relative rewards (see below),

- ϵ is the clipping parameter that prevents excessive updates (as in PPO [21]),

- β is a hyperparameter controlling the strength of KL regularisation,

- $D_{\text{KL}} \left(\pi_\theta \| \pi_{\theta_{\text{ref}}} \right)$ is the KL divergence between the trained policy π_θ and a reference policy $\pi_{\theta_{\text{ref}}}$, used as regularisation.

The advantage function $A_{i,t}$ for the action $a_{i,t}$ taken at step t in output o_i is calculated based on the rewards from both reasoning steps and the final step. The rewards are normalised using the rewards

across all outputs in the group for a specific question. Let the normalised reward for step t of output o_i be:

$$\bar{r}_i^{(t)} = \frac{r_i^{(t)} - \text{mean}(R)}{\text{std}(R)},$$

where

$$R = \left\{ \left\{ r_1^{\text{index}(1)}, \ldots, r_1^{\text{index}(K_1)} \right\}, \ldots, \left\{ r_G^{\text{index}(1)}, \ldots, r_G^{\text{index}(K_G)} \right\} \right\},$$

represents the rewards from all reasoning steps across all outputs in the group G, where $\text{index}(j)$ is the end token index of the j-th step, and K_i is the total number of steps in the i-th output; and $\text{mean}(R)$ and $\text{std}(R)$ are the mean and standard deviation of the group rewards.

The advantage $A_{i,t}$ for the t-th step of output o_i is the sum of normalised rewards from step t to the final step K_i:

$$A_{i,t} = \sum_{j=t}^{K_i} \bar{r}_i^{(j)},$$

where $\bar{r}_i^{(j)}$ is the normalised reward for reasoning step j in output o_i. This advantage function encourages the model to optimise for both intermediate reasoning steps and the final step, by rewarding reasoning paths that yield higher relative performance within the group.

Rather than incorporating a KL penalty directly into the reward, GRPO regularises the policy by adding the KL divergence between the current policy π_θ and a reference policy $\pi_{\theta_{\text{ref}}}$ directly into the loss function. This ensures that the updated policy does not deviate excessively from the reference policy during training, helping maintain stability.

This GRPO formulation, specifically adapted for reasoning tasks with process reward models, optimises LLM policy by leveraging group relative rewards across reasoning steps and final steps. The normalised advantage function is computed based on relative performance, encouraging the policy to favours outputs that perform better within a group of sampled outputs. Additionally, KL regularisation ensures that the updated policy remains close to a reference policy, improving training stability and efficiency. This framework provides a robust approach for guiding LLM reasoning through PRM-based optimisation.

One can explore more efficient offline methods such as token-level DPO [35] without PRM but with sequential reasoning data. For details, please refer to the paper.

4.3 Inference-time Computation

Once trained, the LLM policy must efficiently generate outputs during inference. Autoregressive generation—where tokens are predicted one by one based on previous tokens—is widely used in LLMs. However, for reasoning tasks, more sophisticated decoding techniques are necessary.

To strike a balance between efficiency and effectiveness, the work [24, 33] found that reasoning tasks benefit from more flexible approaches like beam search. In beam search, multiple possible sequences (or beams) are generated simultaneously, and the best candidate is chosen based on cumulative probability. For even more complex reasoning tasks, look ahead model such as MCTS is used. MCTS [6] simulates multiple reasoning paths and evaluates them based on a reward system, selecting the one with the highest expected reward. This allows the model to explore a wider range of possibilities during inference, increasing its chances of arriving at an optimal solution. With an MDP, we could formally define the reasoning process structure.

Definition 2 (Native Chain-of-Thought) Native Chain-of-Thought (NCoT) *refers to the inherent reasoning capability of a large language model (LLM), which allows it to autonomously perform step-by-step, structured reasoning without external prompts. This capability is formalised as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \pi, \mathcal{R})$, where:*

- *$\mathcal{S}$ is the state space, representing the sequence of tokens or reasoning steps generated up to a given point.*

- *$\mathcal{A}$ is the action space, which consists of potential reasoning steps R_t or the final answer A.*
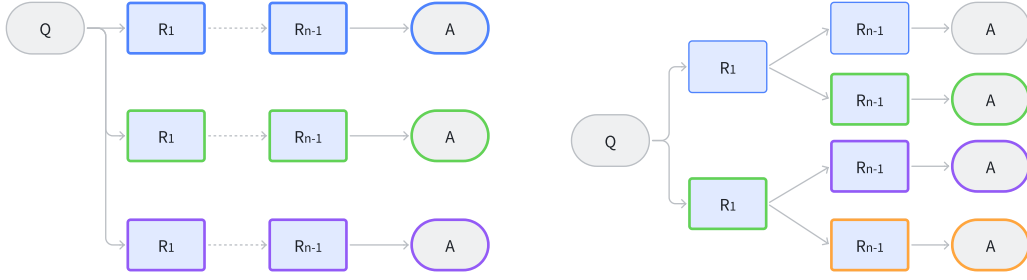
Figure 5: With the PRM, the LLM can perform non-autoregressive reasoning through three approaches: 1) sampling multiple reasoning trajectories, 2) performing a Monte Carlo search over a tree structure of potential reasoning paths, or 3) combining both methods to enhance flexibility and robustness in reasoning.

- $\pi_{LLM}(a_t \mid s_t)$ *is the policy (also the LLM) that governs the selection of actions, determining the next reasoning step or final answer based on the current state s_t.*

- $\mathcal{R}(s_t, a_t)$ *is the process-reward model (PRM), which assigns a reward r_t based on the quality and relevance of the selected action a_t, guiding the reasoning process.*

The model can either follow a sequential reasoning path by unrolling the MDP or explore multiple trajectories by sampling different reasoning steps at each state, forming a tree-like structure (Fig. 5). The process-reward model $\mathcal{R}$ provides a guided search over this space, controlling the reasoning trajectory by favouring actions that lead to more meaningful or correct reasoning steps.

5 Bibliographic Remarks

In the literature, significant attention has been given to inference-time computation, verifiers (also known as reward models), and data acquisition methods, all of which play a critical role in enhancing the reasoning capabilities of these models. In this section, we review and discuss several key papers in these areas, examining their contributions and limitations. The connection between these works and the broader research landscape is depicted in Fig. 6.

5.1 Inference-Time Computing

Several papers have focused on optimising LLM reasoning through inference-time computing. For instance, the paper [6] introduces a method that integrates Monte Carlo Tree Search (MCTS) with LLM decoding, a combination that has proven highly effective in guiding reasoning, particularly for complex, multi-step tasks. The inclusion of MCTS facilitates better decision-making by simulating potential future actions, enhancing the model's ability to plan its next steps. Similarly, the paper [24] emphasises the importance of optimising test-time computation, empirically showing that inference-time reasoning enhancements can often yield more substantial improvements than simply scaling model parameters. This reflects a growing understanding that more compute during inference can be leveraged for higher quality reasoning without necessarily increasing the model's size.

Another approach is presented in [7], which suggests using pause tokens to force models to pause and "think" during reasoning. This method introduces an implicit reasoning model, encouraging the LLM to process information in chunks, mimicking human-like deliberation.

5.2 Verifier Models

Verifier models (outcome-reward models and process-reward models) have become an important area of research in improving LLM reasoning reliability. Papers like [4] introduced the earliest formal attempt (outcome reward only) at using verifiers in mathematical reasoning tasks, laying the groundwork for subsequent research. The follow-up work [27] expands on the concept of verifiers, integrating process-based reasoning mechanisms, and was followed by OpenAI's work on Process
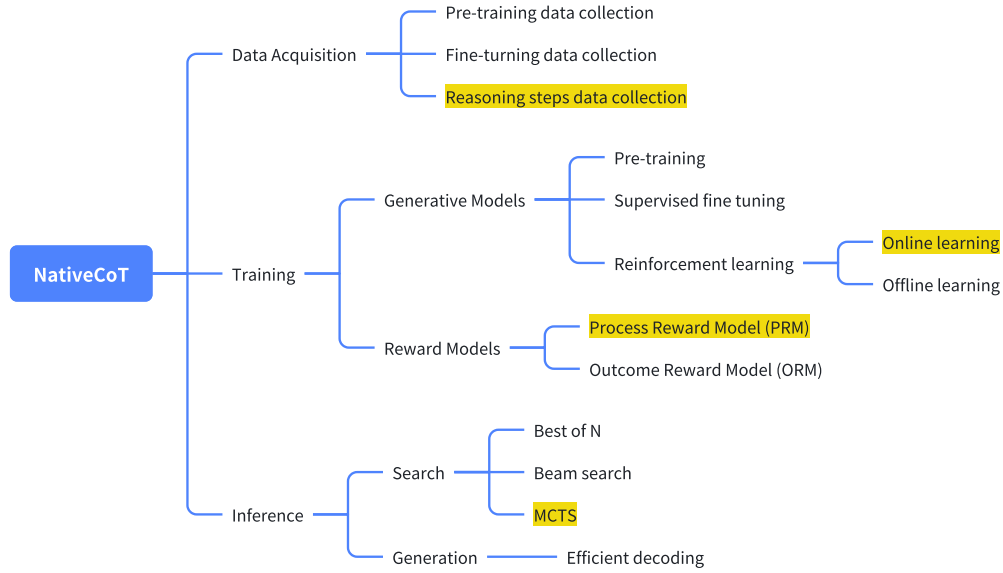
Figure 6: Research on LLM-native chain of thought.

Reward Models (PRMs) [12]. These verifiers play a crucial role in ensuring the correctness of multi-step reasoning, addressing one of the major challenges in LLMs—maintaining coherence and accuracy over extended reasoning sequences.

A more recent addition to this line of research is [11], which combines verifier models with majority voting to produce more reliable outputs in reasoning tasks. This method enhances the robustness of the verification process by cross-checking multiple reasoning paths and filtering out incorrect steps. Such advancements highlight the growing importance of verifiers in maintaining the accuracy of LLMs as they tackle increasingly complex reasoning challenges.

5.3 Data Acquisition for Reasoning Tasks

The acquisition of reasoning data has been another area of focus, particularly in papers like [34], which explores methods for automatically obtaining data related to reasoning steps. STaR introduces a self-teaching paradigm where the model improves its reasoning capabilities by generating and critiquing its own steps, leading to more reliable intermediate steps. The paper [30] takes this approach further, showing how LLMs can be trained step-by-step without the need for costly human annotations, providing a more scalable solution to the reasoning data problem.

The work in [31] highlights the importance of practical data acquisition for reasoning tasks, particularly in coding problems. MCTS has been used for acquiring data in [6], whereas it has been extended with linear search for efficiency in [15].

These papers suggest that for LLMs to advance in reasoning, innovative data acquisition methods, such as self-supervised learning and verification mechanisms, are essential to reduce the dependency on extensive human-labelled datasets.

5.4 Understanding and System-Level Improvements

Finally, there is a growing body of research aimed at understanding the mechanisms behind step-by-step reasoning in LLMs [26, 19]. The work in [25] focused its analysis from graphical models for the chain of thought mechanism. The paper [19] explores the intrinsic reasons why reasoning emerges as a natural capability in LLMs. It suggests that reasoning is a byproduct of the way language models process localised experiences and knowledge. The paper [14] provides an empirical evaluation of

LLMs' ability to critique their own reasoning, showing that self-critique is often limited, and this capability often emerges only when models are sufficiently large.

From a system perspective, the pangu-agent paper [3] introduces structured reasoning mechanisms beyond traditional models like OpenAI's o1 model. This research reflects a shift toward more generalised reasoning agents that can handle a wider array of tasks with greater precision and flexibility, providing a vision of the next generation of reasoning models.

References

[1] R. Bellman. Dynamic programming and stochastic control processes. *Information and control*, 1(3):228–239, 1958.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] F. Christianos, G. Papoudakis, M. Zimmer, T. Coste, Z. Wu, J. Chen, K. Khandelwal, J. Doran, X. Feng, J. Liu, et al. Pangu-agent: A fine-tunable generalist agent with structured reasoning. *arXiv e-prints*, pages arXiv–2312, 2023.

[4] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[5] A. E. Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[6] X. Feng, Z. Wan, M. Wen, Y. Wen, W. Zhang, and J. Wang. Alphazero-like tree-search can guide large language model decoding and training. In *ICML 2024*, 2024.

[7] S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

[8] D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011.

[9] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[10] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[11] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*, 2022.

[12] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[13] C.-C. Lin, A. Jaech, X. Li, M. R. Gormley, and J. Eisner. Limitations of autoregressive models and their alternatives. *arXiv preprint arXiv:2010.11939*, 2020.

[14] L. Luo, Z. Lin, Y. Liu, L. Shu, Y. Zhu, J. Shang, and L. Meng. Critique ability of large language models. *arXiv preprint arXiv:2310.04815*, 2023.

[15] L. Luo, Y. Liu, R. Liu, S. Phatale, H. Lara, Y. Li, L. Shu, Y. Zhu, L. Meng, J. Sun, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

[16] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, D. Dohan, J. Jiang, J. Schulman, W. Fedus, and C. Sutton. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[17] OpanAI. Learning to reason with llms. `https://openai.com/index/learning-to-reason-with-llms/`, 2014,0912.

[18] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[19] B. Prystawski, M. Li, and N. Goodman. Why think step by step? reasoning emerges from the locality of experience. *Advances in Neural Information Processing Systems*, 36, 2024.

[20] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. Li, Y. Wu, and D. Guo. Deepseek-math: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[23] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[24] C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[26] R. Tutunov, A. Grosnit, J. Ziomek, J. Wang, and H. Bou-Ammar. Why can large language models generate correct chain-of-thoughts? *arXiv preprint arXiv:2310.13571*, 2023.

[27] J. Uesato, N. Kushman, R. Kumar, F. Song, N. Siegel, L. Wang, A. Creswell, G. Irving, and I. Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[28] S. Van Gaal, K. R. Ridderinkhof, H. S. Scholte, and V. A. Lamme. Unconscious activation of the prefrontal no-go network. *Journal of neuroscience*, 30(11):4143–4150, 2010.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[30] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.

[31] Z. Wang, Y. Li, Y. Wu, L. Luo, L. Hou, H. Yu, and J. Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*, 2024.

[32] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[33] Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

[34] E. Zelikman, Y. Wu, J. Mu, and N. Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[35] Y. Zeng, G. Liu, W. Ma, N. Yang, H. Zhang, and J. Wang. Token-level direct preference optimization. *arXiv preprint arXiv:2404.11999*, 2024.

A   Standard Training Pipelines of LLMs

The training procedure for LLM typically involves several stages, each building upon the previous one. In the pre-training stage, the model is trained on a massive online corpus using an autoregressive language modelling objective. The goal is to predict the next token given the previous tokens. For a given sequence of tokens $\{x_1, x_2, \ldots, x_T\}$, the token-level cross-entropy loss sums the negative log-probabilities of the true tokens at each position:

$$\mathcal{L}_{\text{pretrain}} = -\sum_{t=1}^{T} \log P(x_t | x_{<t}; \theta),$$

where x_t is the t-th token, $x_{<t}$ represents all tokens before t, θ are the model parameters, and P is the probability distribution over the vocabulary [2]. $p(x_t \mid x_{<t})$ is the probability of the true token x_t given all previous tokens $x_{<t}$. This loss measures how well the model predicts each token in the sequence.

After pre-training, the model is then fine-tuned on collected additional {Question, Answer} pairs. The objective is to maximise the likelihood of the correct answer given the question:

$$\mathcal{L}_{\text{finetune}} = -\sum_{i=1}^{N} \log P(A_i | Q_i; \theta),$$

where Q_i and A_i are the i-th question and answer pair, respectively [20].

Next, Reinforcement Learning from Human Feedback (RLHF) [18] is then applied to further improve the model's instruction-following ability. This involves constructing a reward model $R(x)$ (by pair-wise training data) that estimates the quality of the model's outputs. The policy (language model) is then optimised using methods like Proximal Policy Optimisation (PPO) [21]:

$$\mathcal{L}_{\text{RLHF}} = \mathbb{E}[R(Q, A)] - \beta \cdot \text{KL}(\pi_\theta(Q|A) \| \pi_{\theta_{\text{old}}}(Q|A)),$$

where π_θ is the current policy, $\pi_{\theta_{\text{old}}}$ is the old policy, and β is a hyperparameter controlling the strength of the KL divergence penalty.