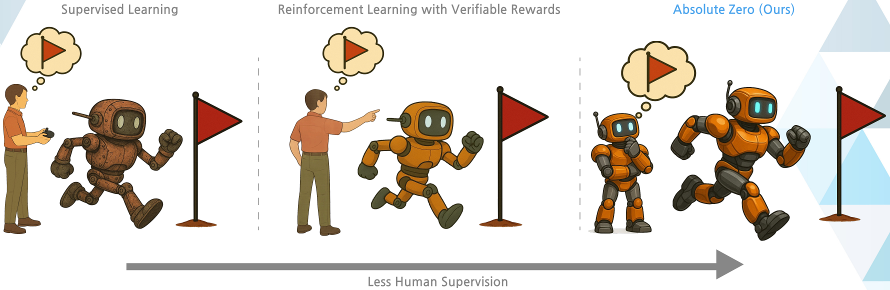
Absolute Zero

Reinforced Self-play Reasoning with Zero Data

Presenter: Nguyen Quang Duc

May 31th 2025



VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

The fact that Science walks forward on two feet, namely theory and experiment...

Prof. Robert Millikan - Nobel Laureate 1923

Table of Contents

Table of Contents

Motivations

Language models (LMs) are babies whose parents are **data**.

Motivations

Language models (LMs) are babies whose parents are **data**.

We have known many methods of continuously fine-tuning LMs, such as: Supervised Finetuning, Reinforcement Learning using Human/AI/Environment Feedback (*e.g.*, PPO, DPO, KTO). Almost required **pre-annotated data**.

Motivations

Language models (LMs) are babies whose parents are **data**.

We have known many methods of continuously fine-tuning LMs, such as: Supervised Finetuning, Reinforcement Learning using Human/AI/Environment Feedback (*e.g.*, PPO, DPO, KTO). Almost required **pre-annotated data**.

In the far future, artificial intelligence (AI) can surpass human intelligence, and pre-annotated data can be a **barrier** for those models to evolve.

Motivations

How about the idea of LM self-evolving?

Motivations

How about the idea of LM self-evolving?

Yes! We can let the models interact with the environment to automatically collect feedback and continuously improve themselves.

Motivations

How about the idea of LM self-evolving?

Yes! We can let the models interact with the environment to automatically collect feedback and continuously improve themselves.

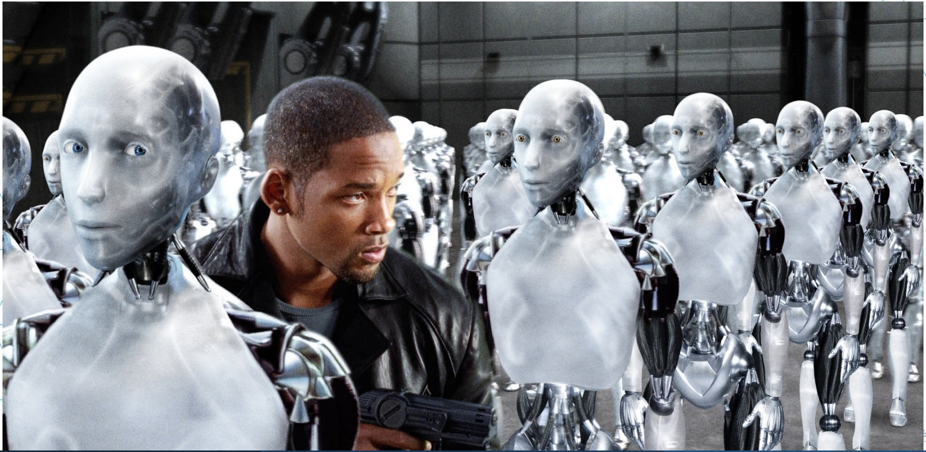


Table of Contents

Supervised Finetuning

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ currated by human experts or superior AI models. In which π_θ is the LM parameterized by θ, x is the input prompt, and y is the expected output. The optimization objective of SFT is defined as **minimizing**:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \log \pi_\theta(y \mid x)$$

Supervised Finetuning

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ currated by human experts or superior AI models. In which π_θ is the LM parameterized by θ, x is the input prompt, and y is the expected output. The optimization objective of SFT is defined as **minimizing**:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \log \pi_\theta (y \mid x)$$

If each sample contains a chain-of-thought (*e.g.*, $\mathcal{D} = \{(x_i, c_i, y_i)\}_{i=1}^{N}$), then the objective become:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,c,y)\sim\mathcal{D}} \log \pi_\theta (y, c \mid x)$$

Reinforcement Learning from Environment Feedback

Reinforcement Learning with Verifiable Rewards (RLVR) is a type of Reinforcement Learning from Environment Feedback, where the rewards are observed by **evaluating output in a real environment**.

Reinforcement Learning from Environment Feedback

Reinforcement Learning with Verifiable Rewards (RLVR) is a type of Reinforcement Learning from Environment Feedback, where the rewards are observed by **evaluating output in a real environment**.

Depending on our preference, we can choose an appropriate fine-tuning technique. In this study, the authors want to have **one output** for each input and **a continuous-valued reward** for each output. Thus, they develop their solution based on the Proximal Policy Optimization (PPO) technique.

From Supervised Fine-Tuning to PPO

Supervised Fine-Tuning (SFT) Objective:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \pi_\theta(y \mid x)\right]$$

- Fine-tunes a language model to imitate human responses.
- Objective maximizes likelihood of expert (human or superior AI) responses.

Reinforcement Learning Fine-Tuning:

$$\mathcal{L}_{\text{RL}}(\theta) = -\mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta}\left[R(x,y)\right]$$

- Uses a scalar reward signal $R(x,y)$ to guide optimization.
- We need to estimate gradients using samples $\rightarrow$ use the log-derivative trick.

Log-Derivative Trick and PPO Objective

Log-Derivative Trick:

$$\nabla_\theta \mathcal{L}_{\text{RL}}(\theta) = -\mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta}\left[R(x,y)\,\nabla_\theta \log \pi_\theta(y \mid x)\right]$$

- Also called the "score function estimator".
- Allows estimating policy gradients from samples.

Proximal Policy Optimization (PPO):

$$\mathcal{L}_{\text{PPO}}(\theta) = -\mathbb{E}_{(x,y)\sim\pi_{\theta_{\text{old}}}}\left[\min\left(r_\theta(y \mid x)\,\hat{A},\, \text{clip}\left(r_\theta, 1-\epsilon, 1+\epsilon\right)\,\hat{A}\right)\right]$$

- $r_\theta = \frac{\pi_\theta(y|x)}{\pi_{\theta_{\text{old}}}(y|x)}$: importance ratio
- $\hat{A}$: advantage estimate (similar role to $R(\cdot)$ in RL)
- Clip term prevents large policy updates; stabilizes learning.

Advantage Estimation in REINFORCE and REINFORCE++

Vanilla REINFORCE:

$$\hat{A} = \sum_{l=0}^{L-1} (\gamma\lambda)^l \delta_{L-l-1}, \quad \text{where} \quad \delta_t = r_t + \gamma V(x_{t+1}) - V(x_t)$$

$V(s)$: learned value function (*i.e.*, the LLM with a different head layer)
$\lambda \in [0, 1]$: controls bias-variance tradeoff
γ: Discount factor
L: Generation length

REINFORCE++: Batch-normalized advantage

$$\hat{A}^{\text{norm}} = \frac{r - \text{mean}\left(\{\hat{A}\}^B\right)}{\text{std}\left(\{\hat{A}\}^B\right)}$$

Normalization is done over batch B to stabilize learning

Absolute Zero's concept

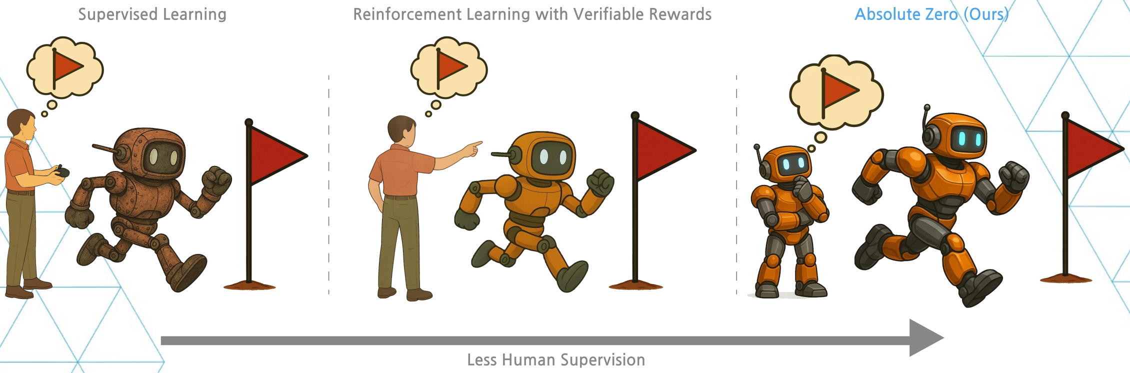
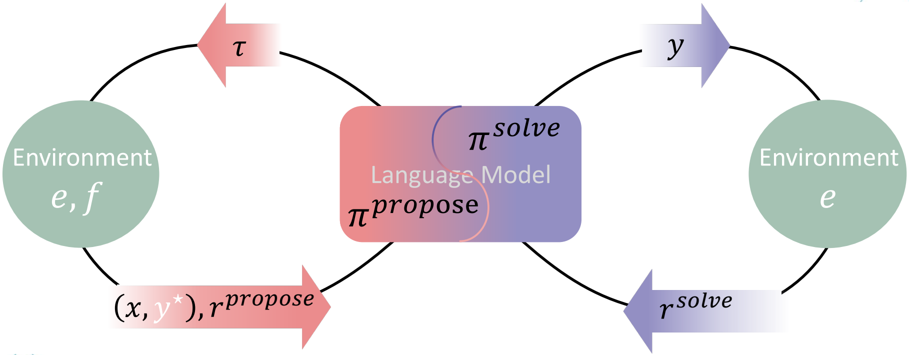


Table of Contents

Overview of Absolute Zero



π: The language model
e: Environment
f: Task validator and constructor

τ: The proposed task
$y^\star, y$: The expected and real output
r: Reward value

What are the tasks?

Reasoning task: triplet (p, i, o) where p: program, i: input, $o = p(i)$: output

Goal: infer one element of the triplet given the other two. This corresponds to three fundamental modes of reasoning, including deduction, abduction, and induction.

What are the tasks?

Reasoning task: triplet (p, i, o) where p: program, i: input, $o = p(i)$: output

Goal: infer one element of the triplet given the other two. This corresponds to three fundamental modes of reasoning, including deduction, abduction, and induction.

1. Deduction (Infer o from p, i)

Proposer: Given task type $\alpha =$ deduction, generate pair (p, i) from reference examples

Solver: Predict output o_π; verified with type-aware equality

What are the tasks?

Reasoning task: triplet (p, i, o) where p: program, i: input, $o = p(i)$: output

Goal: infer one element of the triplet given the other two. This corresponds to three fundamental modes of reasoning, including deduction, abduction, and induction.

1. Deduction (Infer o from p, i)

Proposer: Given task type α = deduction, generate pair (p, i) from reference examples

Solver: Predict output o_π; verified with type-aware equality

2. Abduction (Infer i from p, o)

Proposer: Given α = abduction, generate (p, i) to match known output o

Solver: Predict input i_π such that $p(i_\pi) = o$; verified via output value

What are the tasks?

Reasoning task: triplet (p, i, o) where p: program, i: input, $o = p(i)$: output

Goal: infer one element of the triplet given the other two. This corresponds to three fundamental modes of reasoning, including deduction, abduction, and induction.

1. **Deduction (Infer o from p, i)**

   Proposer: Given task type $\alpha =$ deduction, generate pair (p, i) from reference examples

   Solver: Predict output o_π; verified with type-aware equality

2. **Abduction (Infer i from p, o)**

   Proposer: Given $\alpha =$ abduction, generate (p, i) to match known output o

   Solver: Predict input i_π such that $p(i_\pi) = o$; verified via output value

3. **Induction (Infer p from {input-output} examples)**

   Proposer: Sample p, generate N new examples and message m; store $(p, \{(i^n, o^n)\}, m)$

   Solver: Given few-shot examples and m, synthesize correct program p_π

What are the tasks?

Program Triplet

Input: `"Hello World"`

```
def f(x):
    return x
```

Output: `"Hello World"`

Figure 1: Example of the task triplet

Training Flow

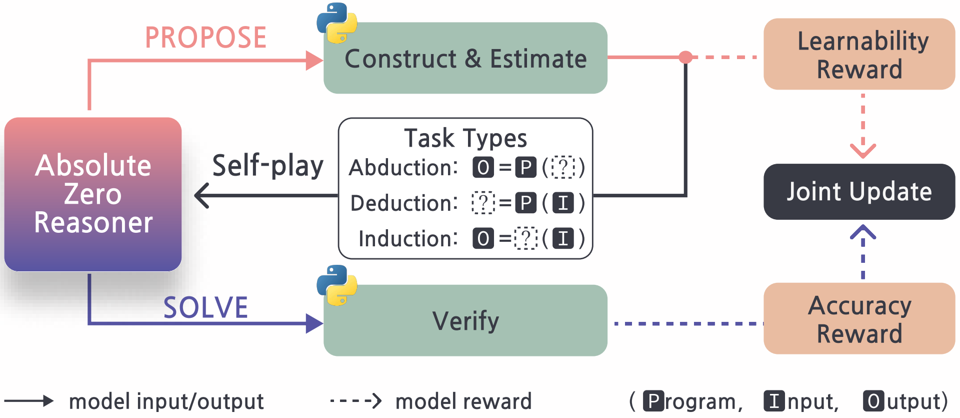


Figure 2: Absolute Zero Reasoner Training Overview

Optimization Objective and Reward Design

With a control variable z:

$$\mathcal{L}_{\text{RL}}(\theta) = -\mathbb{E}_{z\sim p(z)}\left[\mathbb{E}_{(x,y^\star)\sim f_e(\cdot|\tau),\tau\sim\pi_\theta^{\text{propose}}(\cdot|z)}\left[r_e^{\text{propose}}(\tau,\pi_\theta) + \lambda\,\mathbb{E}_{y\sim\pi_\theta^{\text{solve}}(\cdot|x)}\left[r_e^{\text{solve}}(y,y^\star)\right]\right]\right]$$

Reward for Proposer: Encourages generation of moderately difficult tasks

$$r_{\text{propose}} = \begin{cases} 0, & \bar{r}_{\text{solve}} = 0 \text{ or } 1 \\ 1 - \bar{r}_{\text{solve}}, & \text{otherwise} \end{cases} \quad \text{where } \bar{r}_{\text{solve}} = \frac{1}{n}\sum_{i=1}^{n} r_{\text{solve}}^{(i)}$$

Reward for Solver: Binary correctness reward

$$r_{\text{solve}} = \mathbb{I}_{(y=y^\star)}$$

Composite Reward: Format-Aware Penalty[1]

$$R(y_{\pi_{\text{role}}}) = \begin{cases} r_{\text{role}}, & \text{passable response, } r \in \{\text{propose, solver}\} \\ -0.5, & \text{well-formatted but incorrect} \\ -1, & \text{formatting error} \end{cases}$$

[1] DeepSeek-AI et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning".

Composite Reward: Format-Aware Penalty[1]

$$R(y_{\pi_{\text{role}}}) = \begin{cases} r_{\text{role}}, & \text{passable response}, r \in \{\text{propose, solver}\} \\ -0.5, & \text{well-formatted but incorrect} \\ -1, & \text{formatting error} \end{cases}$$

Absolute Zero, based on the PPO technique, defines the advantages as below. The improved point here is computing **separate advantages** for each task and each role.

$$\hat{A}^{\text{norm}}_{\text{task,role}} = \frac{R(y_{\pi_{\text{role}}}) - \mu_{\text{task,role}}}{\sigma_{\text{task,role}}}, \quad \text{task} \in \{\text{ind,ded,abd}\}, \text{role} \in \{\text{propose,solve}\}$$

[1] DeepSeek-AI et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning".

Absolute Zero Reasoner Learning Algorithm

Algorithm 1 Self-Play Training of Absolute Zero Reasoner (AZR)

Require: Pretrained base LLM π_θ; batch size B; #references K; iterations T

$\mathcal{D}_{\text{ded}}, \mathcal{D}_{\text{abd}}, \mathcal{D}_{\text{ind}} \leftarrow \textsc{InitSeeding}(\pi_\theta)$ ▷ buffer initialzation
for $t \leftarrow 1$ to T **do**
  for $b \leftarrow 1$ to B **do** ▷ **PROPOSE PHASE**
    $p \sim \mathcal{D}_{\text{abd}} \cup \mathcal{D}_{\text{ded}}$ ▷ sample a program for induction task proposal
    $\{i_\pi^n\}_{n=1}^N,\ m_\pi \leftarrow \pi_\theta^{\text{propose}}(\text{ind}, p)$ ▷ generate N inputs and a description
    if $\{(i_\pi^n, o_\pi^n)\}_{n=1}^N \leftarrow \textsc{ValidateByExecuting}(p, \{i_\pi^n\}, \textsc{syntax})$ **then** ▷ validate I/Os
      $\mathcal{D}_{\text{ind}} \leftarrow \mathcal{D}_{\text{ind}} \cup \{(p, \{(i_\pi^n, o_\pi^n)\}, m_\pi)\}$ ▷ update induction buffer
    for $\alpha \in \{\text{ded}, \text{abd}\}$ **do**
      $(p_k, i_k, o_k)_{k=1}^K \sim \mathcal{D}_\alpha$ ▷ sample K reference examples
      $(p_\pi, i_\pi) \leftarrow \pi_\theta^{\text{propose}}(\alpha, \{(p_k, i_k, o_k)\})$ ▷ propose new task
      if $o_\pi \leftarrow \textsc{ValidateByExecuting}(p_\pi, i_\pi, \textsc{syntax,safety,determinism})$ **then**
        $\mathcal{D}_\alpha \leftarrow \mathcal{D}_\alpha \cup \{(p_\pi, i_\pi, o_\pi)\}$ ▷ if valid, update deduction or abduction buffers
  for all $\alpha \in \{\text{ded}, \text{abd}, \text{ind}\}$ **do** ▷ **SOLVE PHASE**
    $(x, y^\star) \leftarrow \textsc{SamplePrepareTasks}(\mathcal{D}_\alpha, B, t)$ ▷ $x, y^\star$ prepared based on α
    $y_\pi \sim \pi_\theta^{\text{solve}}(x)$
  Reward: Use proposed task triplets and solved answers to get $r_{propose}$ & r_{solve}
  RL update: use Task Relative REINFORCE++ to update π_θ

Buffer Initialization and Usage

Generate a seed set $\mathcal{D}_{\text{seed}}$ of valid triplets using the base LM. Each prompt samples up to K triplets as references.

Buffer Initialization and Usage

Generate a seed set $\mathcal{D}_{\text{seed}}$ of valid triplets using the base LM. Each prompt samples up to K triplets as references.

At $t = 0$, fall back to a zero triplet (the example triplet above).

Buffer Initialization and Usage

Generate a seed set $\mathcal{D}_{\text{seed}}$ of valid triplets using the base LM. Each prompt samples up to K triplets as references.

At $t = 0$, fall back to a zero triplet (the example triplet above).

Initialize:

$$\mathcal{D}^0_{\text{abduction}} = \mathcal{D}^0_{\text{deduction}} = \mathcal{D}_{\text{seed}}$$

$\mathcal{D}^0_{\text{induction}}$: sampling program from $\mathcal{D}_{\text{seed}}$, then generate corresponding input and output.

Buffer Initialization and Usage

- Generate a seed set $\mathcal{D}_{\text{seed}}$ of valid triplets using the base LM. Each prompt samples up to K triplets as references.
- At $t = 0$, fall back to a zero triplet (the example triplet above).
- Initialize:
  - $\mathcal{D}^0_{\text{abduction}} = \mathcal{D}^0_{\text{deduction}} = \mathcal{D}_{\text{seed}}$
  - $\mathcal{D}^0_{\text{induction}}$: sampling program from $\mathcal{D}_{\text{seed}}$, then generate corresponding input and output.
- During the self-play stage of AZR, the task buffer is used in three ways.
  - **For Proposer (abduction/deduction):** Sample K triplets as in-context examples.
  - **For Induction:** Sample one triplet from $\mathcal{D}_{\text{abd}} \bigcup \mathcal{D}_{\text{ded}}$ to propose N inputs $\{i_n\}$ and message m.
  - **If new batch is not generated completely:** Fill with previously validated tasks.

Buffer Initialization and Usage

Generate a seed set $\mathcal{D}_{\text{seed}}$ of valid triplets using the base LM. Each prompt samples up to K triplets as references.

At $t = 0$, fall back to a zero triplet (the example triplet above).

Initialize:

$\mathcal{D}^0_{\text{abduction}} = \mathcal{D}^0_{\text{deduction}} = \mathcal{D}_{\text{seed}}$

$\mathcal{D}^0_{\text{induction}}$: sampling program from $\mathcal{D}_{\text{seed}}$, then generate corresponding input and output.

During the self-play stage of AZR, the task buffer is used in three ways.

- **For Proposer (abduction/deduction):** Sample K triplets as in-context examples.
- **For Induction:** Sample one triplet from $\mathcal{D}_{\text{abd}} \bigcup \mathcal{D}_{\text{ded}}$ to propose N inputs $\{i_n\}$ and message m.
- **If new batch is not generated completely:** Fill with previously validated tasks.

Buffers grow when valid triplets are proposed, regardless of reward.

Constructing Valid Tasks

Validation Steps:

1. **Program Integrity:** Run $p(i)$, check for return + no errors.
2. **Program Safety:** Ban unsafe packages (`os`, `sys`, etc.).
3. **Determinism:** Approximate by running $j = 2$ times, check consistent outputs:

$$\forall p, \forall i : \; p(i)^{(1)} = p(i)^{(2)}$$

Constructing Valid Tasks

Validation Steps:

1. **Program Integrity:** Run $p(i)$, check for return + no errors.
2. **Program Safety:** Ban unsafe packages (`os`, `sys`, etc.).
3. **Determinism:** Approximate by running $j = 2$ times, check consistent outputs:

$$\forall p, \forall i: \; p(i)^{(1)} = p(i)^{(2)}$$

Task	Input/Output	Answer Verification
Deduction	$x = (p, i)$; $y = o^\star$;	$r_{\text{solve}} = \mathbb{I}[o == o^\star]$
Abduction	$x = (p, o)$; $y = i^\star$;	$r_{\text{solve}} = \mathbb{I}[p(i) == p(i^\star)]$
Induction	$x = (\{i_n, o_n\}^{N/2}, m)$; $y = p^\star$;	$r_{\text{solve}} = \Pi_{n=N/2}^{N}\mathbb{I}[p(i_n) == o_n]$

Table of Contents

Result Summary

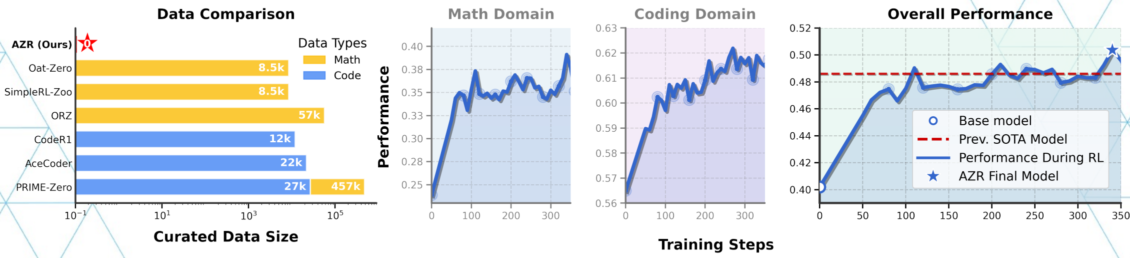


Figure 3: Overall results of Absolute Zero compared to other algorithms

Key Findings and Insights

AZR achieves remarkable results in math and code reasoning with zero in-distribution data.

Strong Zero-Data Performance:

- Matches or beats fine-tuned zero reasoners in math.
- Sets new SOTA in code with RLVR-free training.
- Outperforms prior zero-trained models by **+1.8** avg points.

Code Priors Amplify Reasoning:

- `Qwen-Coder-7b` starts lower but ends up higher after running Absolute Zero.

Cross-Domain Transfer:

- AZR boosts math accuracy by **+10.9 / +15.2** with code training.
- Far exceeds RLVR-trained models (**+0.65**).

Scaling Helps:

- Bigger models yield bigger gains: +5.7 (3B), +10.2 (7B), +13.2 (14B).

Emergent Planning via Comments:

- AZR uses ReAct-style scratchpads in code reasoning.
- Similar to behaviors in 671B formal math models.

Cognitive Behaviors Emerge:

- Step-by-step, enumeration, trial-and-error arise naturally.
- Token usage grows, esp. in abd. task.

Safety Concerns:

- "Uh-oh moments" with `LLaMA3.1-8B` show risky chains of thought.
- Emphasizes the need for safety-aware reasoning training.

Detailed Results

Model	Base	#data	HEval^{+}	MBPP^{+}	LCB^{v1-5}	AME24	AME25	AMC	M500	Minva	Olypiad	CAvg	MAvg	AVG
Base Models														
Qwen2.5-7B	-	-	73.2	65.3	17.5	6.7	3.3	37.5	64.8	25.0	27.7	52.0	27.5	39.8
Qwen2.5-7B-Ins	-	-	75.0	68.5	25.5	13.3	6.7	52.5	76.4	35.7	37.6	56.3	37.0	46.7
Qwen2.5-7B-Coder	-	-	80.5	69.3	19.9	6.7	3.3	40.0	54.0	17.3	21.9	56.6	23.9	40.2
Qwen2.5-7B-Math	-	-	61.0	57.9	16.2	10.0	16.7	42.5	64.2	15.4	28.0	45.0	29.5	37.3
Zero-Style Reasoners Trained on Curated Coding Data														
AceCoder-RM	Ins	22k	79.9	71.4	23.6	20.0	6.7	50.0	76.4	34.6	36.7	58.3	37.4	47.9
AceCoder-Rule	Ins	22k	77.4	69.0	19.9	13.3	6.7	50.0	76.0	37.5	37.8	55.4	36.9	46.2
AceCoder-RM	Coder	22k	78.0	66.4	27.5	13.3	3.3	27.5	62.6	29.4	29.0	57.3	27.5	42.4
AceCoder-Rule	Coder	22k	80.5	70.4	29.0	6.7	6.7	40.0	62.8	27.6	27.4	60.0	28.5	44.3
CodeR1-LC2k	Ins	2k	81.7	71.7	28.1	13.3	10.0	45.0	75.0	33.5	36.7	60.5	35.6	48.0
CodeR1-12k	Ins	12k	81.1	73.5	29.3	13.3	3.3	37.5	74.0	35.7	36.9	61.3	33.5	47.4
Zero-Style Reasoners Trained on Curated Math Data														
PRIME-Zero	Coder	484k	49.4	51.1	11.0	23.3	23.3	67.5	81.2	37.9	41.8	37.2	**45.8**	41.5
SimpleRL-Zoo	Base	8.5k	73.2	63.2	25.6	16.7	3.3	57.5	77.0	35.7	41.0	54.0	38.5	46.3
Oat-Zero	Math	8.5k	62.2	59.0	15.2	30.0	16.7	62.5	80.0	34.9	41.6	45.5	44.3	44.9
ORZ	Base	57k	80.5	64.3	22.0	13.3	16.7	60.0	81.8	32.7	45.0	55.6	41.6	48.6
Absolute Zero Training w/ No Curated Data (Ours)														
AZR (Ours)	Base	**0**	$71.3^{-1.9}$	$69.1^{+3.8}$	$25.3^{+7.8}$	$13.3^{+6.6}$	$13.3^{+10.0}$	$52.5^{+15.0}$	$74.4^{+9.6}$	$38.2^{+13.2}$	$38.5^{+10.8}$	$55.2^{+3.2}$	$38.4^{+10.9}$	$46.8^{+7.0}$
AZR (Ours)	Coder	**0**	$83.5^{+3.0}$	$69.6^{+0.3}$	$31.7^{+11.8}$	$20.0^{+13.3}$	$10.0^{+6.7}$	$57.5^{+17.5}$	$72.6^{+22.6}$	$36.4^{+19.1}$	$38.2^{+16.3}$	$\mathbf{61.6}^{+5.0}$	$39.1^{+15.2}$	$\mathbf{50.4}^{+10.2}$

Example Results

Model-proposed Task

Input: [1,2,3,4,5], 5

```python
def f(numbers: list[int], target: int) -> int:
    accumulated_values = []
    current_sum = 0
    for number in numbers:
        current_sum += number
        accumulated_values.append(current_sum)
    result = 0
    for i in range(len(accumulated_values)):
        for j in range(i + 1, len(accumulated_values)):
            if accumulated_values[j] \
                    - accumulated_values[i] == target:
                result = accumulated_values[i]
                return result
    return result
```

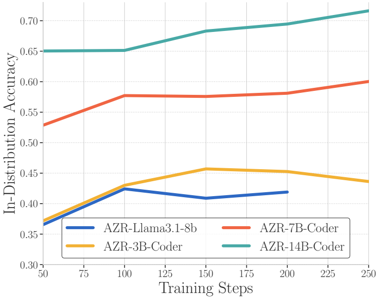
Output: 1

Response of Predict Input

Let's analyze the code snippet **...(analysis)** To find one possible input that produces the output 1, we need to find a scenario where **...(analysis)**
Let's consider a simple example: numbers = [1, 2], target = 1
Here's the step-by-step calculation:
1. current_sum = 0 + 1 = 1, accumulated_values = [1]
2. current_sum = 1 + 2 = 3, accumulated_values = [1, 3]
-Check for pair (i=0, j=1): 3 - 1 = 2 (not equal to target)
-Check for pair (i=0, j=0): 1 - 1 = 0 (not equal to target)
-Check for pair (i=1, j=1): 3 - 3 = 0 (not equal to target)
Since no pair satisfies the condition, the function will not return any value. This doesn't match our output. Let's try another example **...(Tried 3 more examples)**

Given the complexity and the fact that no simple example seems to work, let's try a more complex scenario: numbers = [2, -1, 1], target = 1
Step-by-step calculation:
1. current_sum = 0 + 2 = 2, accumulated_values = [2]
2. current_sum = 2 - 1 = 1 , accumulated_values = [2, 1]
3. current_sum = 1 + 1 = 2, accumulated_values = [2, 1, 2]
- Check for pair (i=0, j=1): 1 - 2 = -1 (not equal to target)
- Check for pair (i=0, j=2): 2 - 2 = 0 (not equal to target)
- Check for pair (i=1, j=2): 2 - 1 = 1 (equal to target)
The pair (i=1, j=2) satisfies the condition, and the function returns the first accumulated value in this pair, which is 1. This matches our output.

In- and Out-of-Distribution Performances



(a)

Model Family	Variant	Code Avg	Math Avg	Total Avg
Llama3.1-8b		28.5	3.4	16.0
Llama3.1-8b	+ SimpleRL	33.7 $^{+5.2}$	7.2 $^{+3.8}$	20.5 $^{+4.5}$
Llama3.1-8b	+ AZR (Ours)	31.6 $^{+3.1}$	6.8 $^{+3.4}$	19.2 $^{+3.2}$
Qwen2.5-3B Coder		51.2	18.8	35.0
Qwen2.5-3B Coder	+ AZR (Ours)	54.9 $^{+3.7}$	26.5 $^{+7.7}$	40.7 $^{+5.7}$
Qwen2.5-7B Coder		56.6	23.9	40.2
Qwen2.5-7B Coder	+ AZR (Ours)	61.6 $^{+5.0}$	39.1 $^{+15.2}$	50.4 $^{+10.2}$
Qwen2.5-14B Coder		60.0	20.2	40.1
Qwen2.5-14B Coder	+ AZR (Ours)	63.6 $^{+3.6}$	43.0 $^{+22.8}$	53.3 $^{+13.2}$

(b)

Figure 5: **(a) In-Distribution & (b) Out-of-Distribution Reasoning Task Performances.**

Ablation Study

Omitting any tasks, reducing the number of references, or roles will result in a performance degradation.

Experiment	Task Type	Gen Reference	Trained Roles	Code Avg.	Math Avg.	Overall
Deduction only	Ded	/	/	54.6	32.0	43.3
w/o Induction	Abd, Ded	/	/	54.2	33.3	43.8
w/o Gen Reference	/	0	/	54.4	33.1	43.8
Train Solver Only	/	/	Solve Only	54.8	36.0	45.4
Absolute Zero	Abd, Ded, Ind	K	Propose & Solve	**55.2**	**38.4**	**46.8**

RQ1: How does AZR compare to other zero-setting models?

Absolute Zero Reasoner-Coder-7B achieves:

- Best-in-class performance among 7B models.
- +1.8% gain over previous SOTA in reasoning benchmarks.
- +0.3% coding gain over expert-trained models—without human-curated data.

Cross-domain generalization (math → code):

- AZR models: +10.9 (base), +15.2 (coder).
- Expert code models: Only +0.65 on average.
- Suggests strong generalization *without human supervision*.

RQ2–4: Initial Model, Scale, and Class Effects

Base vs. Coder Initialization

AZR-Coder started lower in math (23.9 vs. 27.5) but outperformed Base after training.

Initial coding ability accelerates reasoning gains.

Model Scaling Effects

Greater gains for larger models (O.O.D. performance): **+5.7 (3B)**, **+10.2 (7B)**, **+13.2 (14B)**.

Larger models benefit more from AZR training.

Model Class Change

`Llama3.1-8B + AZR` improves +3.2 over SimpleRL baseline.

Performance still scales with base model capability.

RQ5–7: Training Behaviors and Ablations

Emergent Reasoning Behaviors

- Self-proposes rich tasks: DP, string ops, Heron's formula, etc.
- Uses intermediate planning (ReAct-like comments).
- Shows cognitive behaviors, state tracking—and even "uh-oh" moments.

Ablation Results

- Removing task types (e.g., induction): large drop in math performance.
- Removing dynamic proposer conditioning: -5 math / -1 code.
- Skipping proposer training: -1.4 overall.

Key Insight: Diverse task types and learned proposal strategies are *essential* to AZR's success.

Table of Contents

Related Work: Reinforcement Learning for Reasoning

RL for reasoning has emerged as a key method in post-training reasoning improvement[2].

STaR introduced expert iteration + outcome verification via rejection sampling.

o1 scaled this idea and set SOTA in reasoning tasks[3].

R1 matched or surpassed o1 with an open-weight model in the **zero setting**.

Zero setting: RL applied directly to base LLMs, without supervised fine-tuning.

Inspired open-source extensions and RL algorithm improvements[4]

Procedural RL on human puzzles[5], and few-shot RL nearly matches thousands[6].

Our work: Absolute Zero—RLVR from base LLMs without prompts, answers, or human data.

---

[2] Lambert et al., "TÜLU 3: Pushing Frontiers in Open Language Model Post-Training".

[3] Jaech et al., "Openai o1 system card".

[4] Zeng et al., "SimpleRL-Zoo: Investigating and Taming Zero Reinforcement Learning for Open Base Models in the Wild"; Liu et al., "Understanding R1-Zero-Like Training: A Critical Perspective"; Cui et al., "Process Reinforcement through Implicit Rewards"; Hu et al., "Open-Reasoner-Zero: An Open Source Approach to Scaling Up Reinforcement Learning on the Base Model"; Yu et al., "DAPO: An Open-Source LLM Reinforcement Learning System at Scale"; Y. Yuan et al., "VAPO: Efficient and Reliable Reinforcement Learning for Advanced Reasoning Tasks".

[5] Xie et al., "Logic-RL: Unleashing LLM Reasoning with Rule-Based Reinforcement Learning".

[6] Y. Wang et al., *Reinforcement Learning for Reasoning in Large Language Models with One Training Example*.

Self-Play and Emergent Reasoning

Self-play: proposal vs. prediction agents (e.g., Schmid *et al.*[7]).

AlphaGo/AlphaZero: superhuman play via self-competition[8].

Unsupervised variants:

- Asymmetric self-play[9], unsupervised env design[10], automatic goal gen[11].
- GANs as self-play between generator and discriminator[12].

---

[7] Schmidhuber, "Exploring the predictable".

[8] Silver et al., "Mastering the game of Go with deep neural networks and tree search".

[9] Sukhbaatar et al., "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play".

[10] Dennis et al., "Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design".

[11] Florensa et al., "Automatic Goal Generation for Reinforcement Learning Agents".

[12] Goodfellow et al., "Generative adversarial networks".

LLM-centric self-play:

SPIN, Self-Rewarding LMs[13]: reward = model itself.
Prover-Verifier Games[14]; EVA[15]; SPC[16].
Genius, EMPO, TTRL: human queries, no labels[17].
Minimo: formal math conjecture–theorem co-training[18].

Our work: First to apply self-play for long CoT generation in grounded Python task space.

[13] Z. Chen et al., "Self-Play Fine-Tuning Converts Weak Language Models to Strong Language Models"; W. Yuan et al., "Self-rewarding language models".
[14] Kirchner et al., "Prover-Verifier Games improve legibility of LLM outputs".
[15] Ye et al., "Evolving Alignment via Asymmetric Self-Play".
[16] Jiaqi Chen et al., *SPC: Evolving Self-Play Critic via Adversarial Games for LLM Reasoning*.
[17] F. Xu et al., *Genius: A Generalizable and Purely Unsupervised Self-Training Framework For Advanced Reasoning*; Zhang et al., *Right Question is Already Half the Answer: Fully Unsupervised LLM Reasoning Incentivization*; Y. Zuo et al., *TTRL: Test-Time Reinforcement Learning*.
[18] Poesia et al., "Learning Formal Mathematics From Intrinsic Motivation".

Weak-to-Strong Supervision

Prior work: Weaker teachers guide stronger learners[19].

Superalignment projects explore oversight of superhuman agents[20].

Our setting: learner may be superhuman—yet receives no external supervision.

Alternative: Verifiable rewards provide scalable, automatic feedback.

Key difference: learning tasks and goals are not human-defined—**entirely self-generated**.

Enables fully autonomous reasoning improvement via self-practice + reward refinement.

[19] Burns et al., "Weak-to-Strong Generalization: Eliciting Strong Capabilities With Weak Supervision"; Hinton, Vinyals, and Dean, "Distilling the Knowledge in a Neural Network"; Christiano, *Capability Amplification*.

[20] Leike and Sutskever, *Introducing Superalignment*.

Table of Contents

Conclusion: Absolute Zero Reasoning (AZR)

Absolute Zero paradigm: Reasoning agents generate their **own task distribution** and improve via verifiable feedback.

AZR instantiation: Code-based reasoning tasks + RLVR with code executor.

Key results:

- Outperformed SOTA in general reasoning and coding—**without curated datasets**.
- Strong performance across model sizes; boosts other model families.

Open-sourced: Code, models, logs to encourage adoption.

Takeaway: AZ unlocks scalable, domain-general reasoning—**without reliance on human labels**.

Discussion: Experience, Exploration, and Beyond

- Expand environments: web, formal math, world simulators, real-world agents[21].
- Apply AZ to new domains: science, embodiment, complex planning[22].
- Future work:
  - Dynamic learning objective f, privileged info in $p(z)$, multimodal AZR.
  - Exploration in task space—**not just how to solve, but what to solve.**
- **Limitation:** AZR showed "uh-oh moments" (e.g. unsafe CoTs); calls for better **safety oversight**.
- **Final insight:** AZR agents have **experience**—they define and evolve their own learning journey.

[21] Zitkovich et al., "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control"; Ren et al., *DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition.*

[22] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework"; Y. Wu et al., "StateFlow: Enhancing LLM Task-Solving through State-Driven Workflows".

- THE END -

Thank you for your attention

Contact

nqduc@hcmut.edu.vn