TeCho Reading Group
Jan 30, 2026

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Absolute Zero:
Reinforced Self-play Reasoning with Zero Data

Spotlight @ NeurIPS ‘25

Andrew Zhao[1], Yiran Wu[3], Yang Yue[1], Tong Wu[2], Quentin Xu[1], Yang Yue[1], Matthieu Lin[1], Shenzhi Wang[1], Qingyun Wu[3], Zilong Zheng[2], and Gao Huang[1]

[1]Tsinghua University,
[2]Beijing Institute for General Artificial Intelligence,
[3]Penn State University

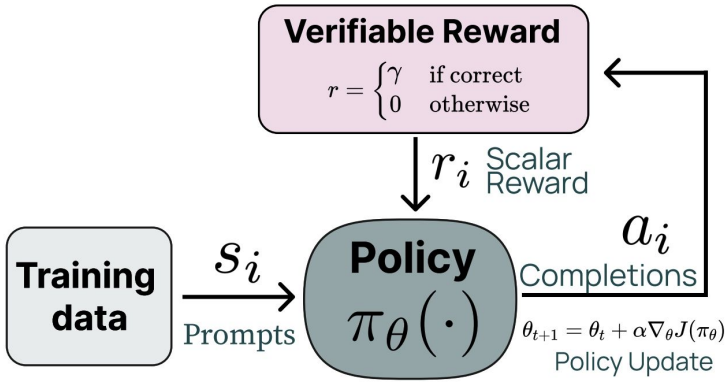
Presented by Sriram Sai Ganesh

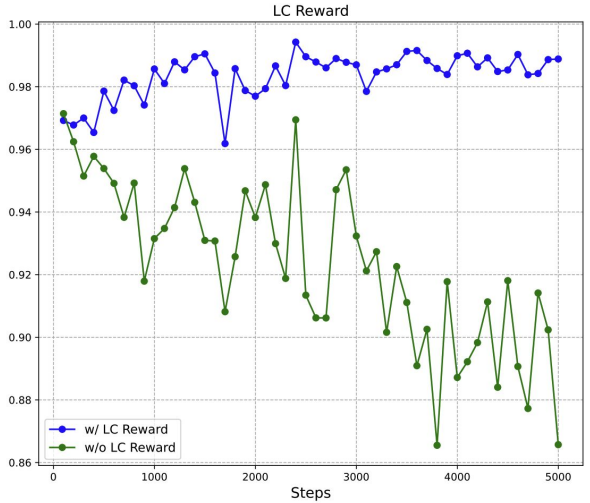
Sections

1. Background
2. Introduction
3. RL from Absolute Zero
4. AZR Training
5. Results

Background

Reinforcement Learning from Verifiable Rewards (RLVR)





Background

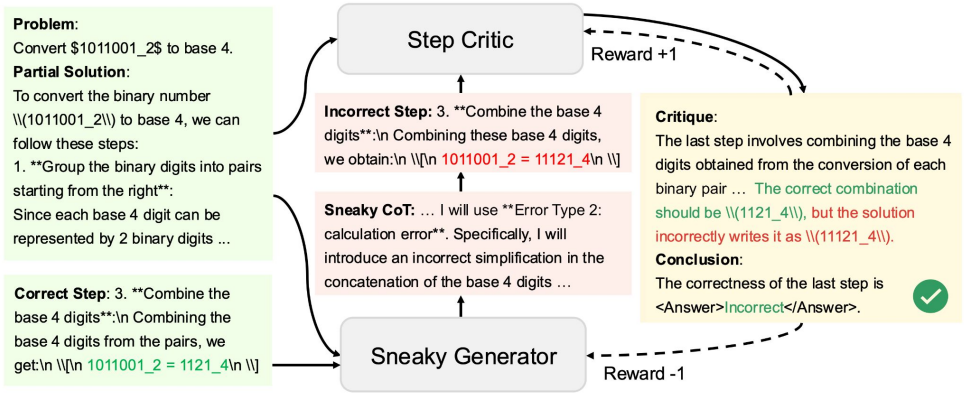
Reinforcement Learning from Verifiable Rewards (RLVR)

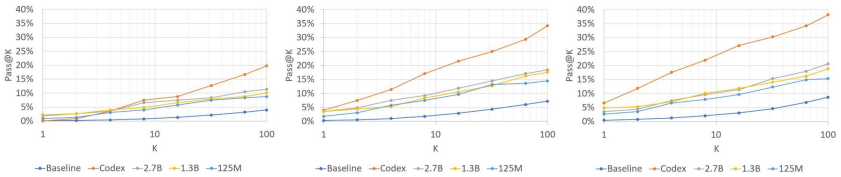
$$J_{\text{RLVR}}(\theta) = \mathbb{E}_{(x,y^\star)\sim\mathcal{D},\ (c,y)\sim\pi_\theta(\cdot\,|x)}\left[r(y,y^\star)\right]$$

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

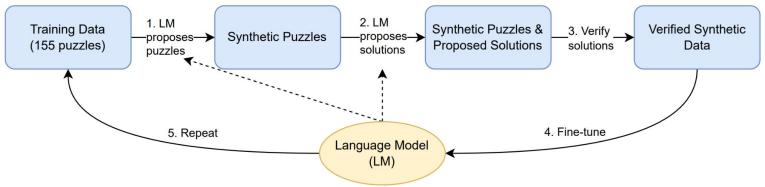
Background

Self-play for self-improvement

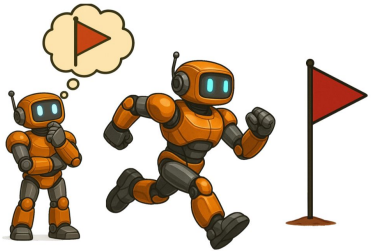




(a) GPT-Neo 125M Model (b) GPT-Neo 1.3B Model (c) GPT-Neo 2.7B Model



Introduction



Intuition: *LMs can autonomously learn to define & solve challenging tasks.*

Task: ***Code reasoning*** –

Fill-in any element of the triplet:

{*program, inputs, outputs*}

Predominant approaches:

- **Supervised Fine-Tuning (SFT):** requires strongly supervised task-rationale-answer demonstrations.
- **RLVR:** Generally uses expert-curated task-answer datasets, limiting scalability.

Limitation: **Existing approaches are expensive & limited in scalability.**

- Remove expert data-curation constraint – much cheaper.
- Remove data availability bottleneck – can theoretically learn infinitely.

This paper:
Can we self-learn problem creation & solving using a grounded environment?

Absolute Zero

Overview

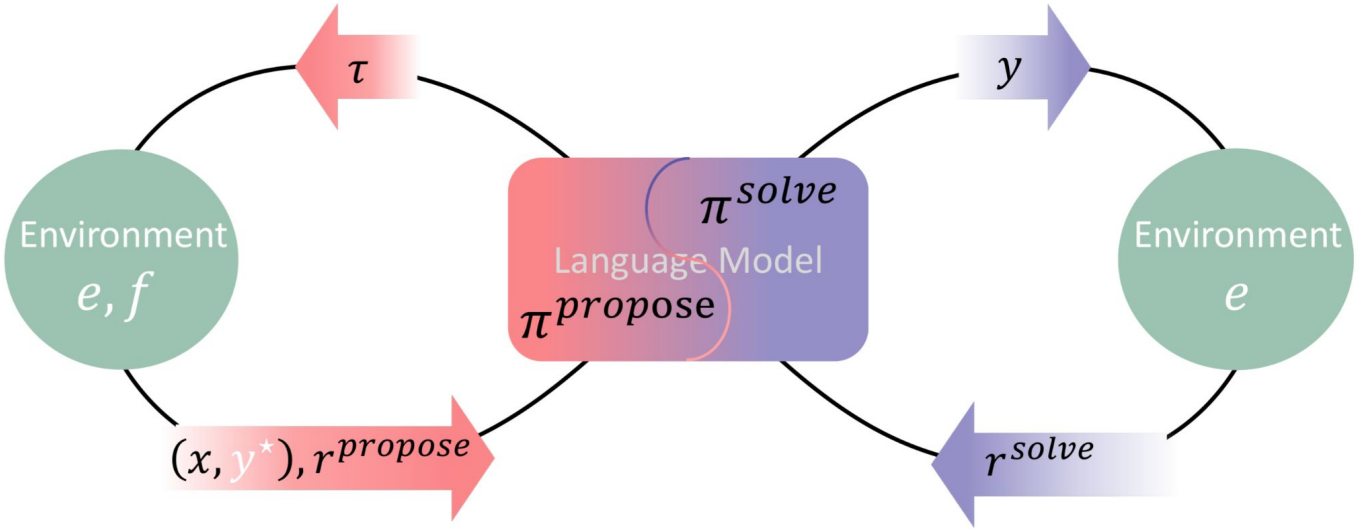


Figure 3

Absolute Zero

Two roles

The model simultaneously:

- Proposes tasks:
  - $\tau \sim \pi_\theta^{\text{propose}}(\cdot|z)$
  - $(x, y^\star) \sim f_e(\cdot|\tau)$
- Solves them:
  - $y \sim \pi_\theta^{\text{solve}}(\cdot \mid x)$

And learns from both stages:

- Learnability reward:
  - $r_e^{\text{propose}}(\tau, \pi_\theta)$
  - How much can π_θ improve by solving $\mathcal{T}$?
- Solution reward:
  - $r_e^{\text{solve}}(y, y^\star)$
  - How correct is y?

Absolute Zero

Two roles

$$r_{\text{propose}} = \begin{cases} 0, & \text{if } \bar{r}_{\text{solve}} = 0 \\ 1 - \bar{r}_{\text{solve}}, & \text{otherwise.} \end{cases}$$

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

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

Equations 4, 5 & 3.

Absolute Zero

Three tasks

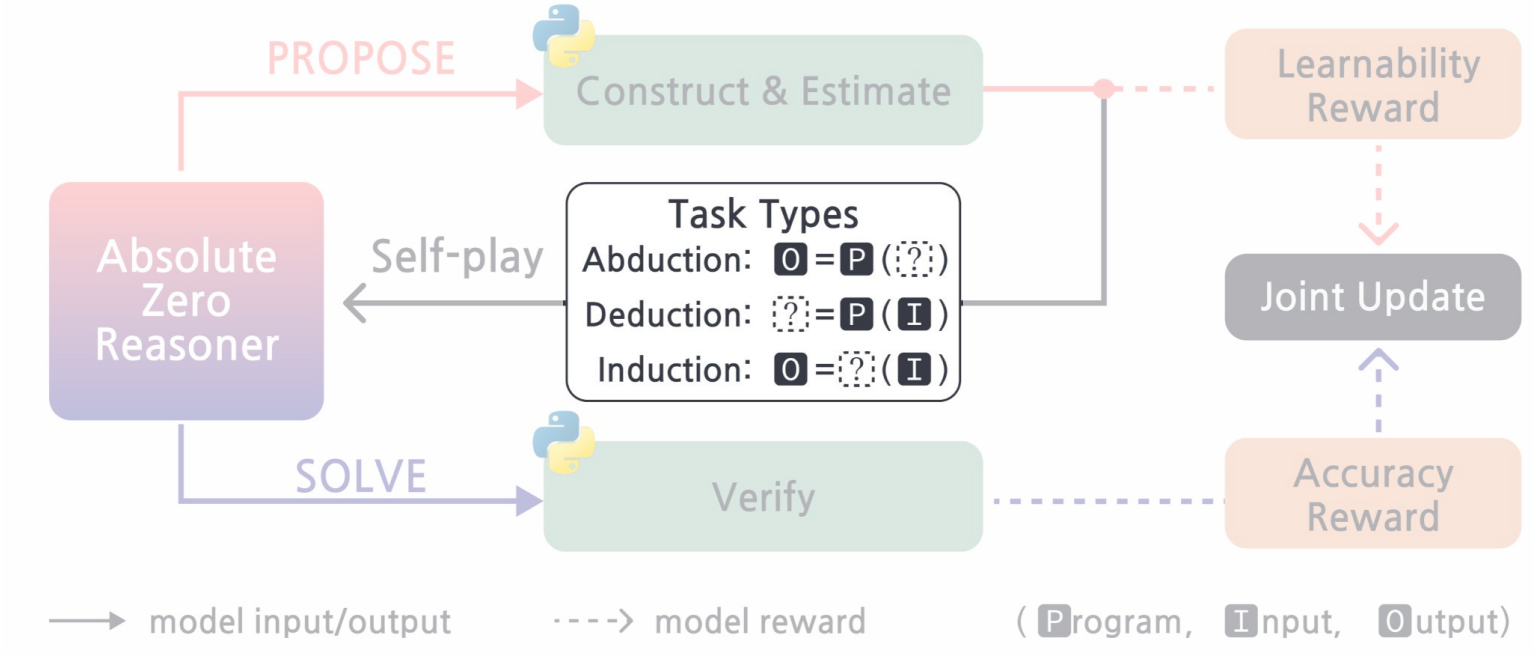


Figure 4

Absolute Zero

Three tasks

Abduction:

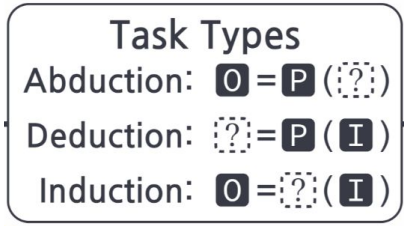
Predict input given program & output.

$$x = (p, o)$$

Deduction:

Predict output given program & input.

$$x = (p, i)$$



Induction:

Predict program given *n* input-output pairs, and a natural-language description *m*.

$$x = (\{i^n, o^n\}_{n=1}^{N//2}, m)$$

Absolute Zero

Algorithm

Task buffers:

- Problem caches for each task. (all the same size, 4 x Batch Size)
- Generated by the LLM:
  - Deduction & Abduction buffers generated as *(p,i)* pairs.
  - These are passed to *e* to get *(p, i, o)* triplets.
  - Sample from this & generate *m* to create Induction buffer.

Task Validation:

- Program Integrity
- Program Safety
- Program Determinism

TRR++

$$A^{\text{norm}}_{\text{task,role}} = \frac{r - \mu_{\text{task,role}}}{\sigma_{\text{task,role}}}, \quad \text{task} \in \{\text{ind,ded,abd}\}, \text{role} \in \{\text{propose,solve}\}$$

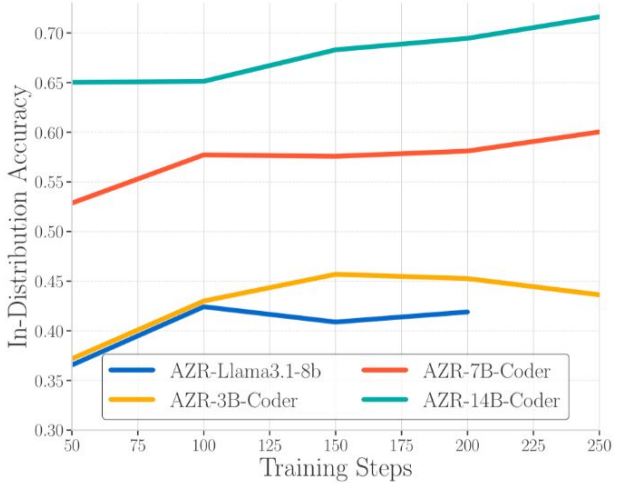
Absolute Zero

Results

Model	Base	#data	HEval+	MBPP+	LCB v1-5	AME24	AME25	AMC	M500	Minva	Olypiad	CAvg	MAvg	AVG
Base Models														
Qwen2.5-7B[87]	-	-	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[87]	-	-	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[32]	-	-	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[88]	-	-	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[98]	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[98]	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[98]	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[98]	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[44]	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[44]	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[13]	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[99]	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[47]	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[29]	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$^{+16.3}$	**61.6**$^{+5.0}$	39.1$^{+15.2}$	**50.4**$^{+10.2}$

Absolute Zero

Results



(a)

Model Family	Variant	Code Avg	Math Avg	Total Avg
Llama3.1-8b		28.5	3.4	16.0
Llama3.1-8b	+ SimpleRL[99]	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)

Absolute Zero

Research Questions

1. Performance comparison: AZR-ed vs. RL-ed models (ie. with expert data)?
    - *AZR is SoTA; maintains performance cross-domain unlike other RL-ed models.*
2. Initializing AZR from base vs. coder?
    - *AZR Coder is SoTA & surprisingly surpasses (non-AZR) Coder on math.*
3. Does model size affect generalization?
    - *AZR helps more when the base model is larger.*

- Interesting behaviors/patterns:
    - Intermediate planning during code response
    - Cognitive behavior in Llama
    - Token length increase depends on task type.

Absolute Zero

Results

```
<think>
Design an absolutely ludicrous and convoluted Python function that is extremely difficult to
↪  deduce the output from the input, designed to keep machine learning models such as Snippi
↪  guessing and your peers puzzling.

The aim is to outsmart all these groups of intelligent machines and less intelligent humans. This
↪  is for the brains behind the future.
</think>
- Absolute Zero Reasoner-Llama3.1-8b @ step 132
```

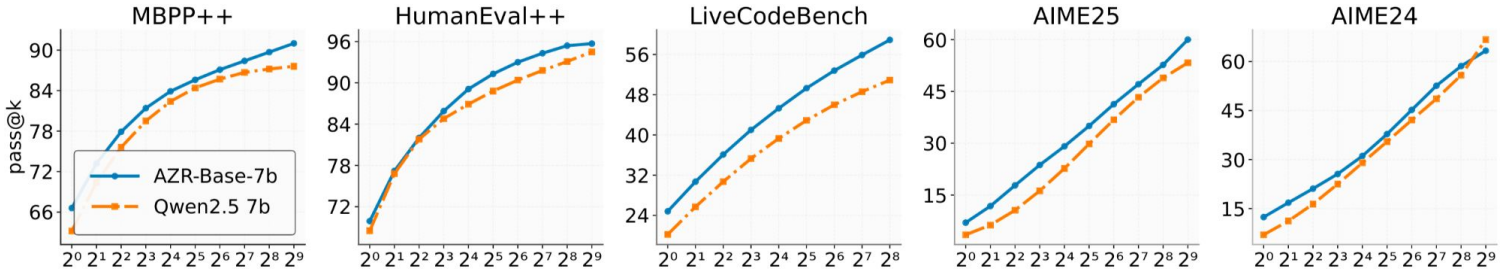


Figure 34 & 8

Absolute Zero

Algorithm

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

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

$\mathcal{D}_{\text{ded}}, \mathcal{D}_{\text{abd}}, \mathcal{D}_{\text{ind}} \leftarrow \textsc{InitSeeding}(\pi_\theta)$ ▷ see §3.3.1

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

    $\left(\{i_\pi^n\}_{n=1}^N,\ m_\pi\right) \leftarrow \pi_\theta^{\text{propose}}(\text{ind}, p)$ ▷ generate N inputs and a description

    if $\{(i_\pi^n, o_\pi^n)\}_{n=1}^N \leftarrow \textsc{ValidateAndConstruct}(p, \{i_\pi^n\}, \textsc{syntax})$ **then** ▷ validate I/Os, see §3.3.3

      $\mathcal{D}_{\text{ind}} \leftarrow \mathcal{D}_{\text{ind}} \cup \{(p, \{(i_\pi^n, o_\pi^n)\}, m_\pi)\}$ ▷ 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 $o_\pi \leftarrow \textsc{ValidateAndConstruct}(p_\pi, i_\pi, \textsc{syntax,safety,determinism})$ **then** ▷ see §3.3.3

        $\mathcal{D}_\alpha \leftarrow \mathcal{D}_\alpha \cup \{(p_\pi, i_\pi, o_\pi)\}$ ▷ 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 and t, see §3.3.3

    $y_\pi \sim \pi_\theta^{\text{solve}}(x)$

  Reward: Use proposed task triplets and solved answers to get $r_{propose}$ & r_{solve} ▷ see §3.1

  RL update: use Task Relative REINFORCE++ to update π_θ ▷ see §3.3.5