CS 957, System-2 AI Planning & Learning

Mahdieh Soleymani

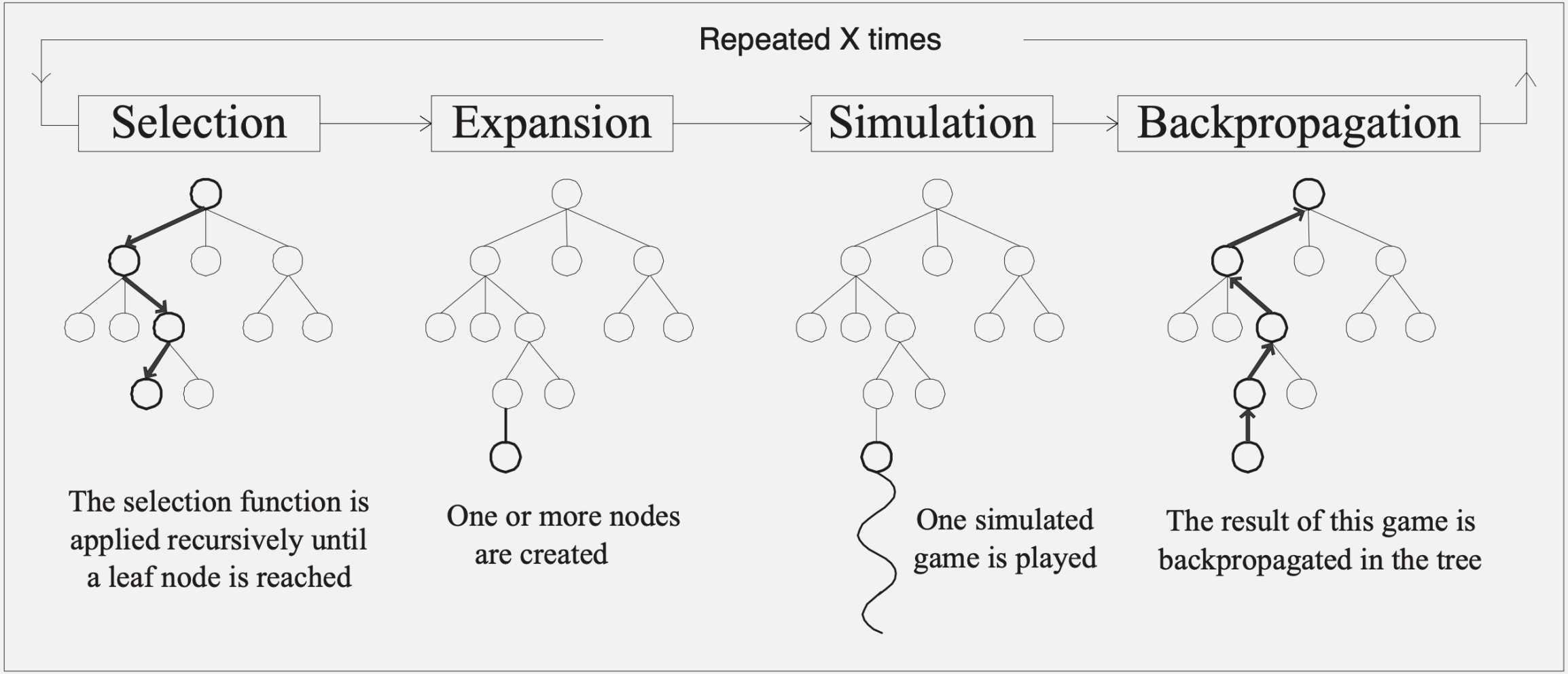
April 2025

Sharif University of Technology

Efficiently search without expert knowledge

- Look-ahead search
- Exploration: Learn the values of actions we are uncertain about
- Exploitation: Focus the search on the most promising parts of the tree

Monte Carlo Tree Search (MCTS)



G. Chaslot, Monte-Carlo Tree Search: A New Framework for Game AI, 2008

Monte Carlo Tree Search (MCTS) [Coulom, 2006]

- MCTS is a sampling-based search algorithm
- It handles large search spaces by sampling intelligently
  - concentrates the search on the most promising branches of the tree

- The algorithm iteratively:
  1) Selects a path (from the root) according to a heuristic
  2) Expands a new node (a previously unvisited state) from the end of that path
  3) Repeatedly simulates a random rollout from that state to get an outcome
  4) Backpropagates the result to update values of ancestor nodes and inform future selections.

Selection Policy on s_t:
if s_t is not fully expanded, choose new a_t
else select child with best score

Multi-Armed Bandits

- Slot machines payout according to their own reward distributions.

- Goal: maximize total expected reward earned over time by choosing which arm to pull.
  - Need to balance exploration (learning the effects of different actions) vs. exploitation (using the best known action).

Upper Confidence Bound Algorithm

- Record the mean reward for each arm.
- Construct a confidence interval for each expected reward
- Optimistically select the arm with the highest upper confidence bound.
  - Increase the required confidence over time.

$$\frac{\sum_{t=1}^{n_i} r_t^i}{n_i} + c\sqrt{\frac{\log N}{n_i}}$$

P. Auer et al., Finite time analysis of the multiarmed bandit problem, 2002

Upper Confidence Bounds applied to Trees (UCT)

- Approach the selection of a node for traversal as a bandit problem.
- Maintain of a tree that represents the states you've seen.
- Track the average reward and the number of times each state has been visited.

- Main concept: Rather than relying on a pre-designed heuristic, simulate a random trajectory from each state multiple times to estimate its value.
  - When combined with UCB, provides a solid approximation of the quality of a game state.

Selection in UCT

- When selecting action a on state s, we apply this method:

$$\frac{\sum_{t=1}^{n(s,a)} r_t(s,a)}{n(s,a)} + c\sqrt{\frac{\log n(s)}{n(s,a)}}$$

- $n(s)$: number of times the state s has been visited
- $n(s,a)$: number of times action a has been applied on state s
- $r_t(s,a)$: reward from t-th choose of a on state s
- c: exploration hyperparameter

Expansion / Simulation / Backpropagation

- Expansion: always expand children nodes that are unvisited by adding it to the tree.
- Simulation: Estimate the value of the new node by randomly simulating until the end of the game (roll-out).
- Backpropagate the value to the ancestors of the node.
  - Update nodes counts and values (wins) for parents
- MCTS is anytime: accuracy improves with more computation

Limitations of MCTS

- Often a random rollout is not a great estimator for the value of a state
  - Learn to estimate the value of states
  - Learn a smarter policy for rollouts
- UCT expands every child of a state before going deeper
  - Learn which states are promising enough to expand
- UCT does not use prior knowledge at test time
  - Remember the results of simulations during training to speed up decision making at test time

Thinking Fast and Slow with Deep Learning and Tree Search

Thomas Anthony[1,✉], **Zheng Tian**[1], and **David Barber**[1,2]

[1]University College London
[2]Alan Turing Institute
✉thomas.anthony.14@ucl.ac.uk

Decomposing **Planning** and **Learning**

- Deep RL algorithms rely on a model not only to generalize plans, but to discover them too

- A novel RL algorithm which decomposes the problem into **separate planning and generalization tasks**.
  - Planning new policies is performed by tree search, while a deep neural network generalizes those plan
  - Subsequently, tree search is improved by using the neural network policy to guide search, bootstraping the (fast) NN policy to improve expert

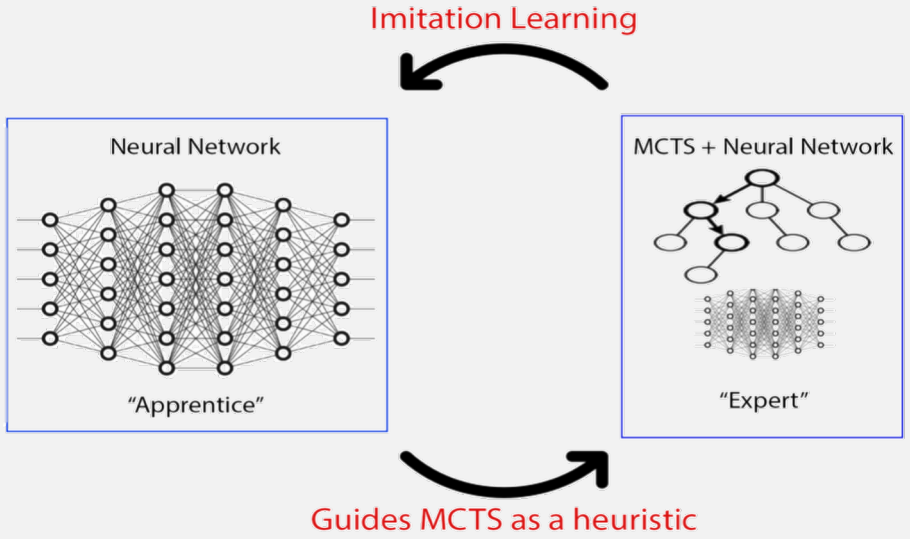
Expert Iteration (ExIt)

- **Expert Iteration** (ExIt) can be viewed as an extension of Imitation Learning (IL) where experts are not available.

- It **iteratively re-solve the IL problem**. Between each iteration, an expert improvement step is performed
  - Apprentice is implemented as a deep network, and the expert by a tree search algorithm
  - The (fast) apprentice policy is bootstrapped to increase the slow expert performance

Expert Iteration: Planner and Learner

- The role of the **expert** (**planner**) is to perform exploration
  - It is calculated using a tree search algorithm.
- The role of the **apprentice** (**learner**) is to generalize policies that the expert discovers
- The knowledge of learner is bootstrapped back into the planner
  - By using the NN policy to direct search effort towards promising moves, or by evaluating states encountered during search more quickly and accurately



Canonical Choice of Planner and Learner

- The expert (planner) is a tree search algorithm.
  - considers the exact dynamics of the game tree local to the state under consideration.
    - Similar to the lookahead human games players engage in when planning their moves.
  - By employing search, strong move sequences can be found potentially far away from the apprentice policy.

- The apprentice (learner) policy is a deep network that can be used to bias search towards promising moves, aid node evaluation, or both.
  - It is able to efficiently generalize across large state spaces

Algorithm 1 Expert Iteration

$\hat{\pi}_0$ = initial_policy()
π_0^* = build_expert($\hat{\pi}_0$)
for i = 1; i $\leq$ max_iterations; i++ **do**
    S_i = sample_self_play($\hat{\pi}_{i-1}$)
    $D_i = \{(s, \text{imitation\_learning\_target}(\pi^*_{i-1}(s)))|s \in S_i\}$
    $\hat{\pi}_i$ = train_policy(D_i)
    π_i^* = build_expert($\hat{\pi}_i$)
end for

$\pi^* \equiv \pi_{MCST}$

Imitation Learning from MCTS

- Finally, the move at the root s selected by MCTS is:

$$a^* = \underset{a}{\operatorname{argmax}}\, n(s,a)$$

- The loss at state s is given by the chosen-action targets (CAT) or tree-policy targets (TPT):

$$\mathcal{L}_{CAT} = -\log[\pi_\theta(a^*|s)]$$

$$\mathcal{L}_{TPT} = -\sum_a \frac{n(s,a)}{n(s)} \log[\pi_\theta(a|s)]$$

- Why not train the policy to pick just the optimal (MCTS) action?
  - Some states have several good actions.

Learning the Policy Network

- Run MCTS for n iterations on a state s
- Define the target policy: $\pi_{MCTS}(a|s) = \frac{n(s,a)}{n(s)}$

$$\hat{\pi} = \underset{\theta}{\operatorname{argmin}} - \sum_{a} \pi_{MCTS}(a|s) \log \pi_{\theta}(a|s)$$

Expert Improvement using the Policy Network

- Using neural network policy to bias the MCTS policy towards moves we believe to be stronger.
  - When a node is expanded, we evaluate the apprentice policy $\hat{\pi}$ at that state, and store it

$$\mathrm{UCT}_{\mathrm{P-NN}}(s,a) = \mathrm{UCT}(s,a) + w_a \frac{\hat{\pi}(a|s)}{n(s,a)+1}$$

- A policy network allows us to narrow the search

Value Network

- A value network acts to reduce the required search depth
  - compared to using inaccurate rollout-based value estimation.
- A KL loss between $V_\theta(s)$ and the sampled (binary) result z to train $V_\theta(.)$:

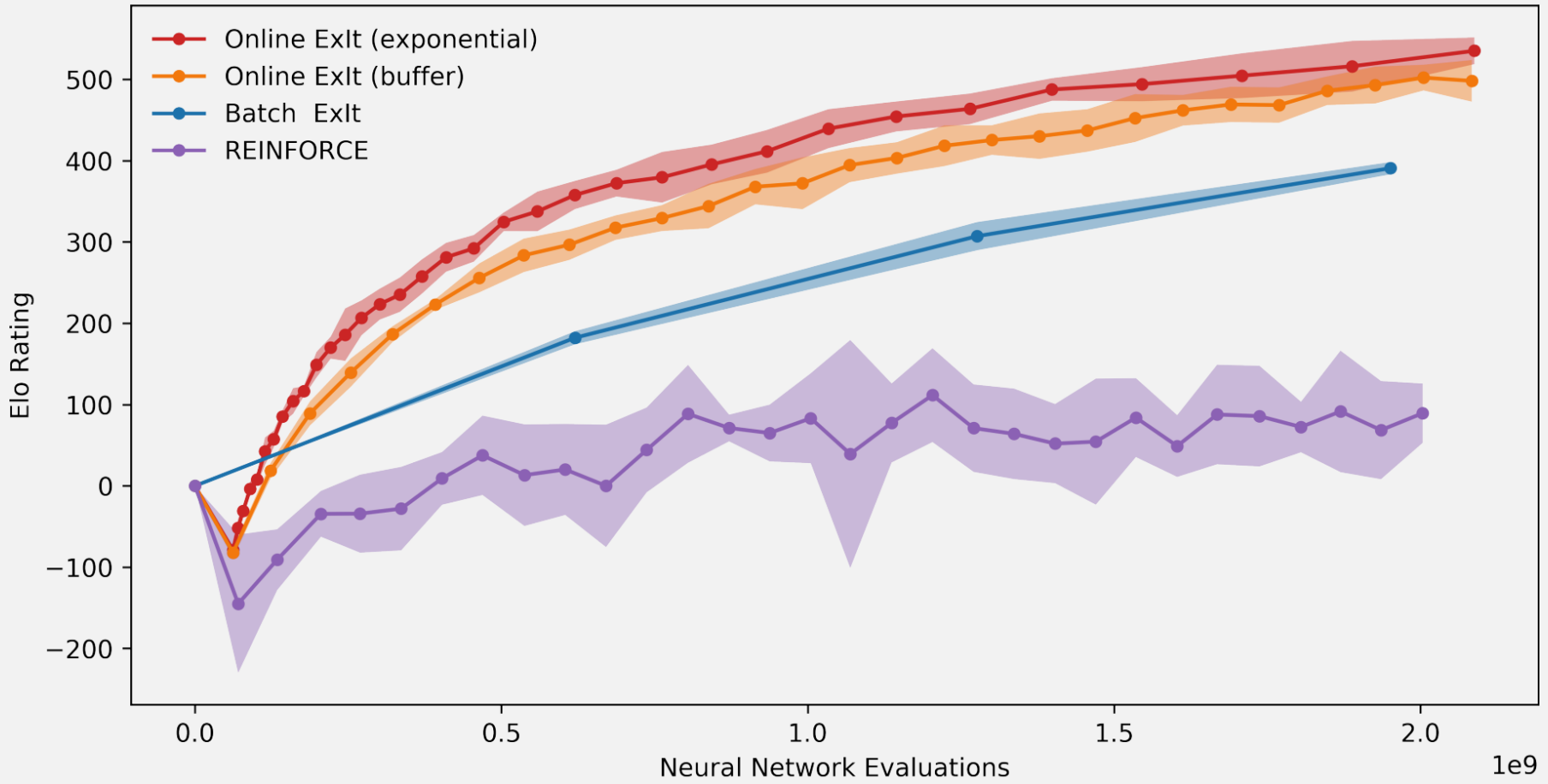
$$\mathcal{L}_V = -z \log V_\theta(s) - (1 - z) \log\big(1 - V_\theta(s)\big)$$

- Whenever a leaf is expanded, $V_\theta(s)$ is estimated.
  - This is backed up through the tree to the root in the same way as rollout results are
    - each edge stores the average of all evaluations made in simulations passing through it.
    - In the tree policy, the value is estimated as a weighted average of the network estimate and the rollout estimate.

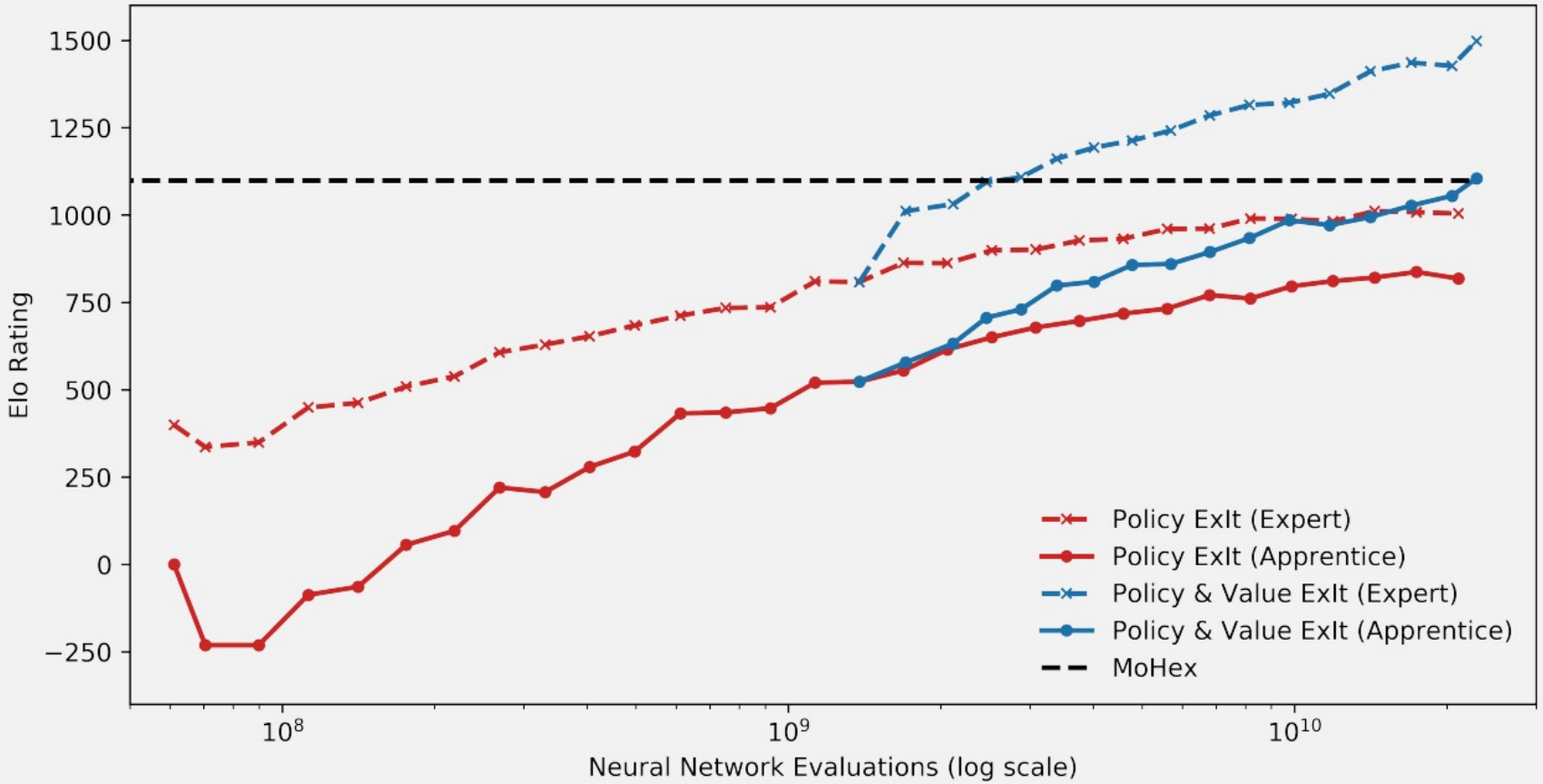
A Multi-task Network

- To accelerate the tree search and regularize value prediction, a multitask network with separate output heads for the apprentice policy and value prediction is used.

$$(\hat{\pi}(.|s), v(s)) = f_\theta(s)$$



T. Anthony et al., Thinking Fast and Slow with Deep Learning and Tree Search, NIPS 2017



T. Anthony et al., Thinking Fast and Slow with Deep Learning and Tree Search, NIPS 2017

ExIt: Summary

- An NN policy is used to improve the performance of the tree search by providing fast ‘intuitions’ to guide search

- Expert iteration alternates between two high-level operations:
  - Data collection: a base policy is combined with a search procedure to produce samples
  - Policy improvement: supervised learning on these expert samples to improve the base policy towards the expert policy.

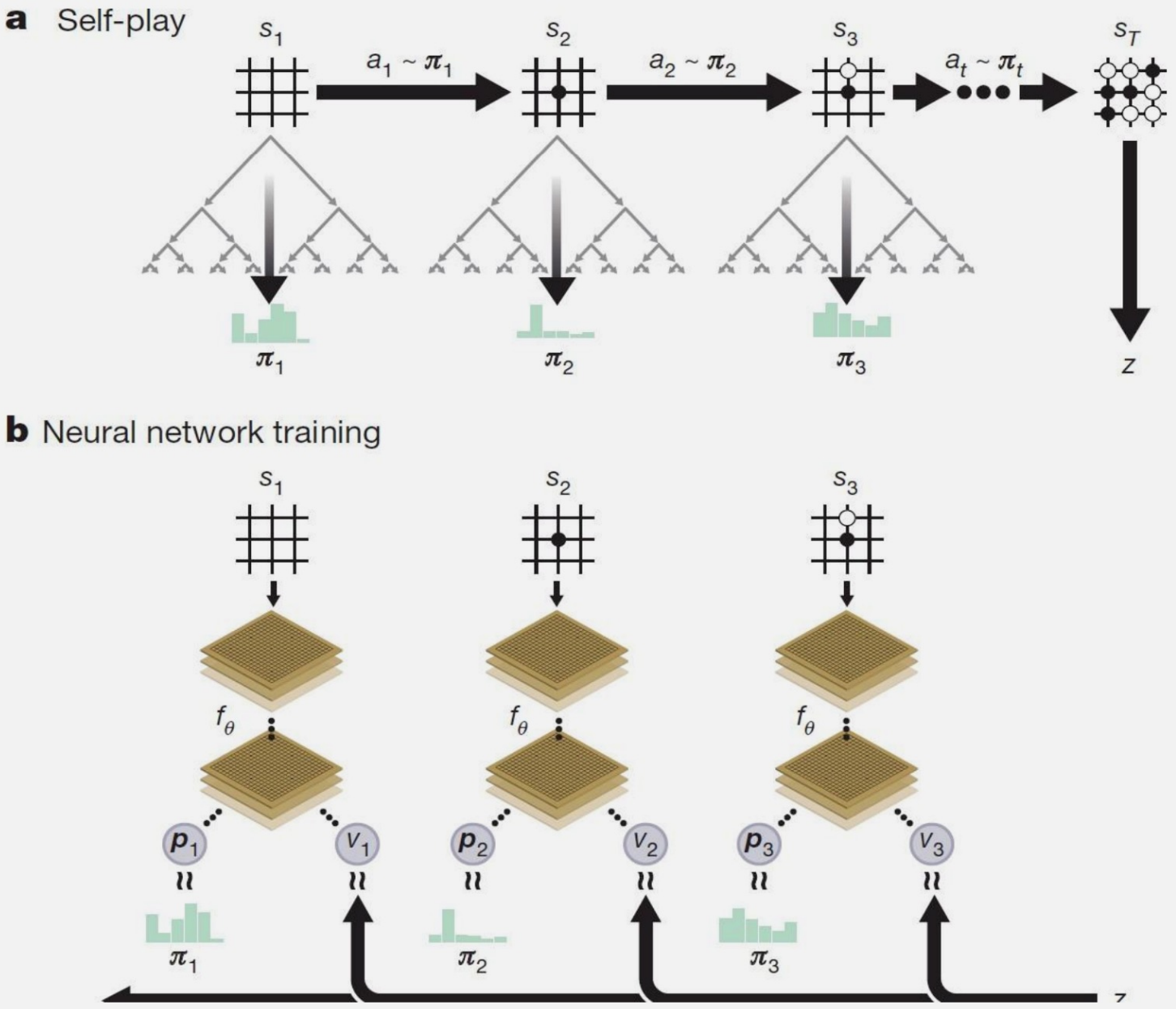
AlphaZero

- Starting from random play, given no domain knowledge except the game rules
- AlphaZero augments the deep policy with a tree search.
- Self-play games between the augmented network and itself can be used as a source of experience to train the network.
- This process progressively bootstraps the network from a random initialization up to superhuman play

D. Silver et al., Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017

AlphaZero: MCTS

- Games are played by selecting moves for both players by MCTS, $a_t \sim \pi_t$
- A game is played by repeatedly running MCTS from a state and choosing a move, until a terminal position is encountered
- The search returns a vector $\boldsymbol{\pi}$ representing a probability distribution over moves



$\pi(a|s) \propto n(s,a)^{1/\tau}$, where τ is a temperature parameter

D. Silver, Mastering the game of Go without human knowledge, Nature 2017.

AlphaZero: Policy Improvement Objective

- The policy model is found as:

$$(\boldsymbol{p}, v) = f_{\boldsymbol{\theta}}(\boldsymbol{s})$$

v: predicted outcome
z: the game outcome (-1/1/0)
$\boldsymbol{p}$: policy vector
$\boldsymbol{\pi}$: the MCTS probabilities

- AlphaZero are trained by self-play, starting from randomly initialized θ
  - get training examples in the form (s_t, π_t, z_t) through self play

- Parameters $\boldsymbol{\theta}$ are updated as:

$$\mathcal{L} = (z - v)^2 - \boldsymbol{\pi}^T \log \boldsymbol{p} + c\|\boldsymbol{\theta}\|_2^2 \qquad v \approx E[z|s]$$

- AlphaZero simply maintains a single neural network that is updated continually, rather than waiting for an iteration to complete.

MCTS

- Each simulation traverses the tree by selecting the edge with:

$$(P(\cdot\,|s), V(s)) = f_\theta(s)$$

$$Q(s,a) + c\frac{P(a|s)}{1+n(s,a)}$$

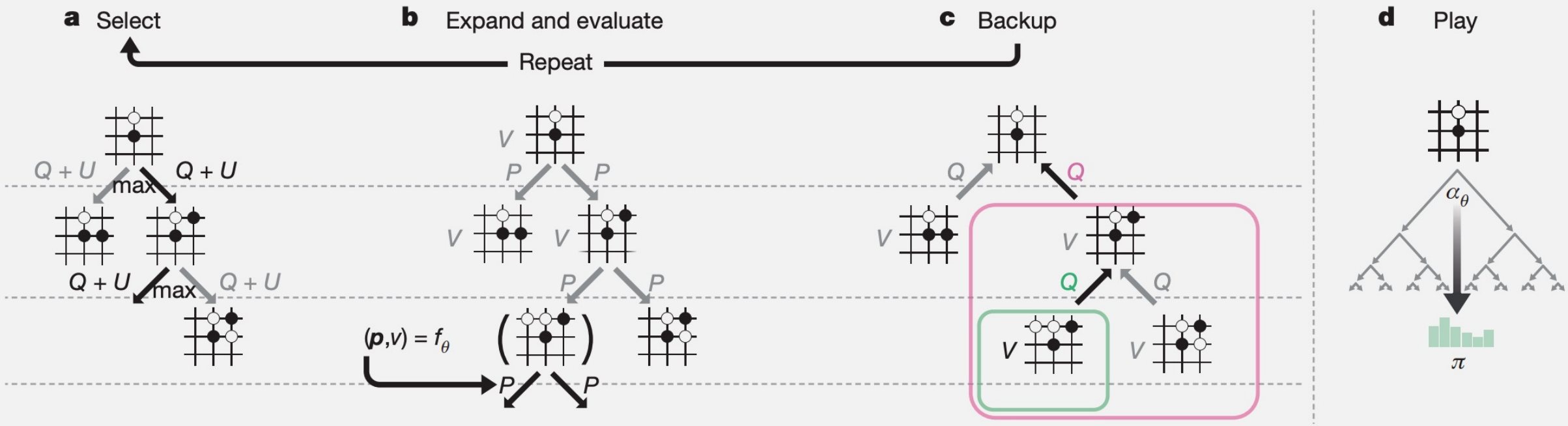
until a leaf node s′ is encountered.

The leaf is expanded and evaluated only once by the network to generate $(P(\cdot\,|s'), V(s')) = f_\theta(s')$.

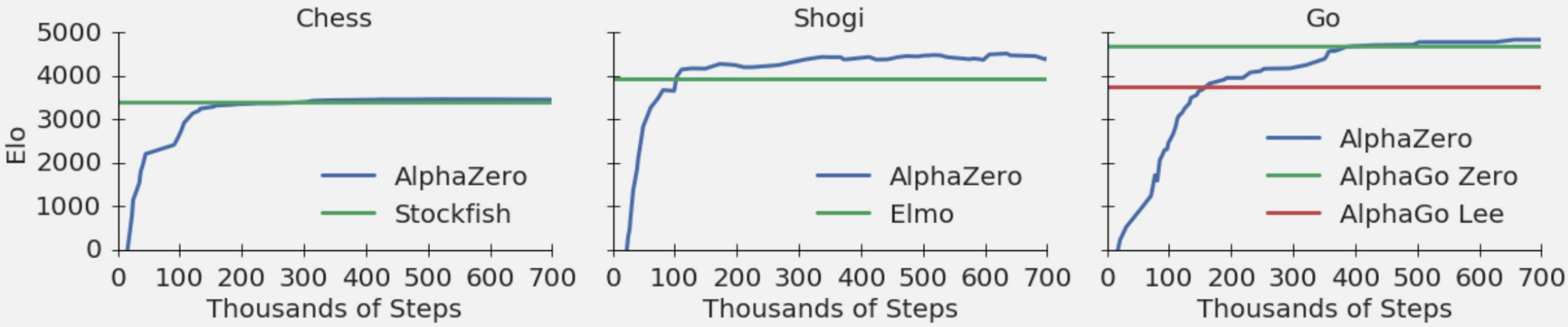
Improving MCTS with the Learned Value

- Evaluate positions with the value network instead of rollouts.

- Some variants (e.g., ExIt) use a combination of a rollout (using the policy network) and the value network.
  - Rollouts are usually more expensive than value network computations.

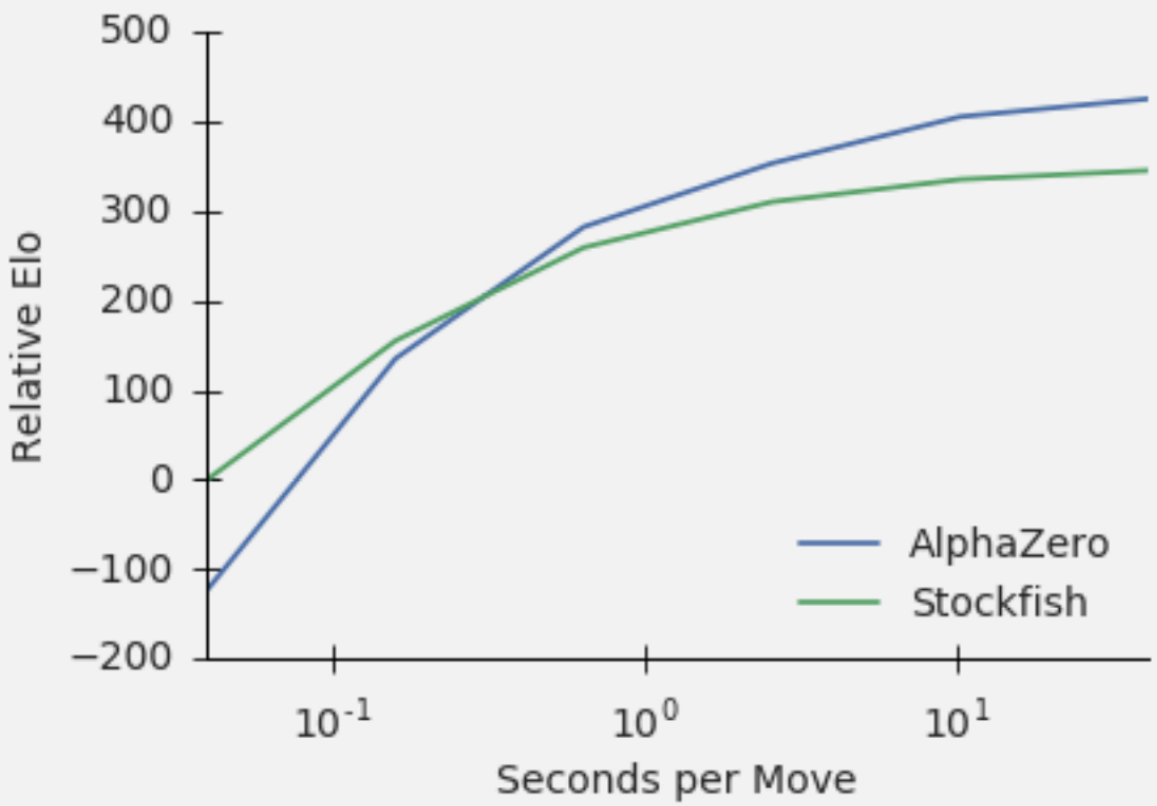


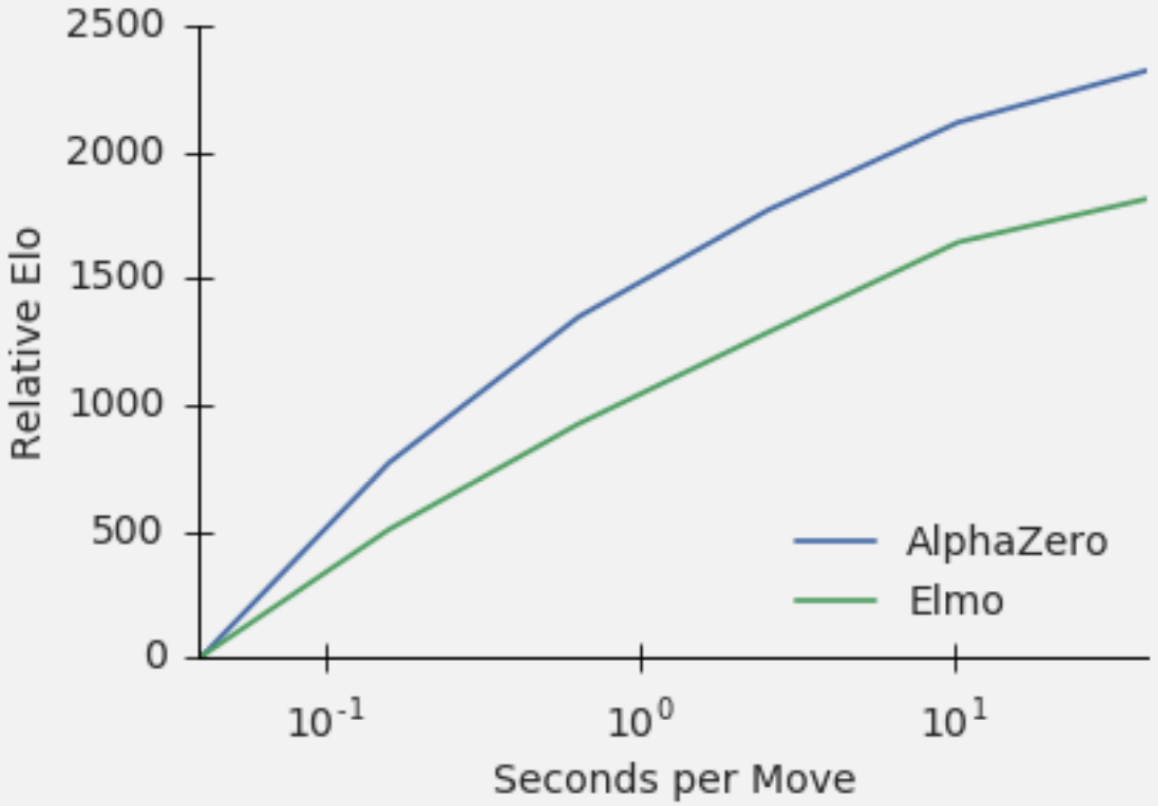
D. Silver, Mastering the game of Go without human knowledge, Nature 2017.



D. Silver et al., Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017

Chess and Shogi: Results





D. Silver et al., Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017

AlphaZero: Summary

- New SOTA algorithm for Go, Shogi, and Chess
- Trained solely through self-play + Monte-Carlo Tree Search
- Trained using maximum likelihood estimation (MLE) to predict policy and reward, without using reinforcement learning for updates!

Learning to search: Summary

- Policy Network $\pi_\theta(a|s)$
  - Probability distribution over the moves
  - Used to focus the search towards good moves
  - Can replace the random policy during rollouts
- Value Network $V_\theta(s)$
  - Predicts the value of any given game state
  - An alternative to rollout simulation in MCTS
- Data is collected from self-play games
- Policy and Value networks are either trained after each iteration (Expert Iteration) or continuously (AlphaZero)

LLMs & MCTS

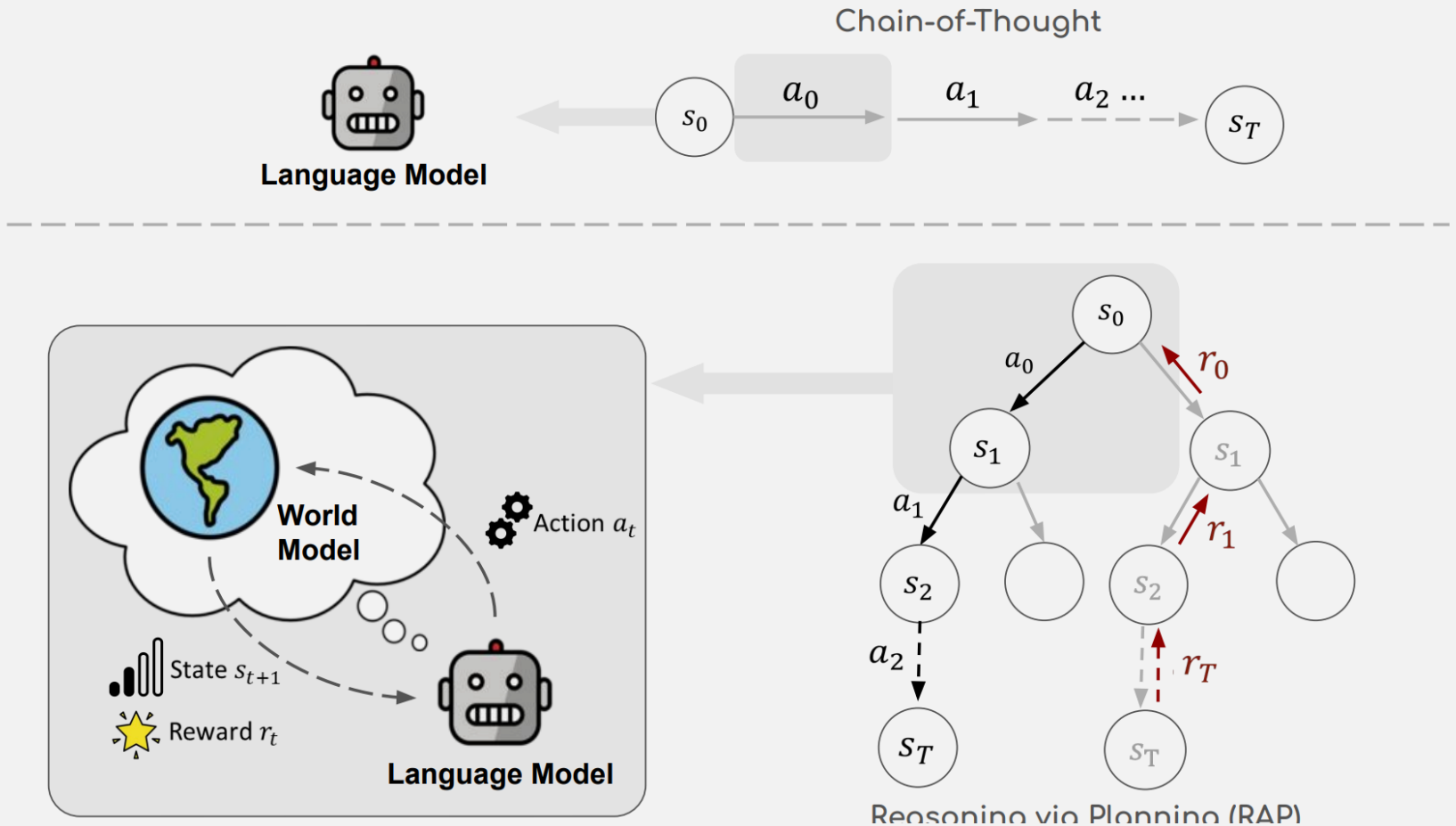
- MCTS-based planning effectively maintains a proper balance between exploration (of unvisited reasoning traces) and exploitation (of the best reasoning steps identified so far).
- How can we combine MCTS and LLMs?
- How about MCTS, LLMs, and RL?

Integration of MCTS with LLMs: Challenges

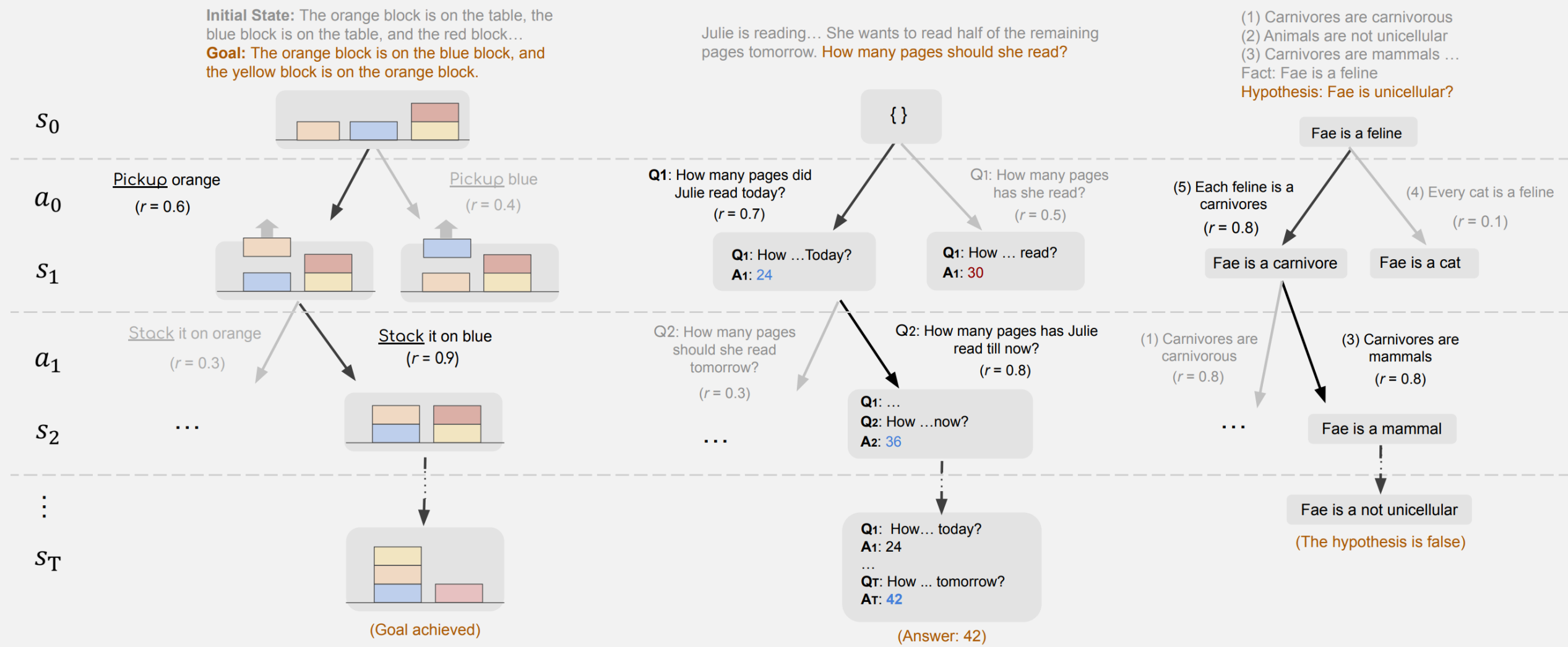
- The integration of MCTS with LLMs has several challenges:
  - Limited Data: High-quality annotated data for LLMs is generally scarce.
    - Furthermore, how to construct of synthetic data for LLMs training, similar to AlphaZero's self-play data, remains unclear.
  - Search inefficiency: The vast number of potential token combinations
  - Imperfect Feedback: In contrast to the clear win/loss feedback, feedback in natural language tasks is often subjective and nuanced

Tian et al., Toward Self-Improvement of LLMs via Imagination, Searching, and Criticizing, 2025

Reasoning via Planning (RAP)



Hao et al., Reasoning with Language Model is Planning with World Model, 2023



Hao et al., Reasoning with Language Model is Planning with World Model, 2023

Language Model as a World Model

- build the world model by repurposing the LLM with prompting
  - **Actions**: the LLM generates an action space by sampling from its generative distribution $a_t \sim p(a|s_t, c)$, where c is a proper prompt (e.g., in-context demonstrations)
  - **Transition model**: the same LLM is repurposed to obtain a state transition distribution $p(s_{t+1}|s_t, a_t, c')$

Hao et al., Reasoning with Language Model is Planning with World Model, 2023

Reward Design

- **Likelihood of the action**: log probability of the action as a reward. This reward reflects the "instinct" of LLMs as an agent
- **Confidence of the state**: The confidence of the state (i.e., answers in this case) as a reward.
  - e.g., in math reasoning, given an action (i.e., a sub-question), the world model predicts the next state by answering the sub-question.
- **Self-evaluation by the LLM**
  - Requests LLM to criticize itself with the question "Is this reasoning step correct?"
- **Task-specific heuristics**
  - e.g., for Blocks-world, we compare the predicted current state of blocks with the goal to calculate a reward

Hao et al., Reasoning with Language Model is Planning with World Model, 2023

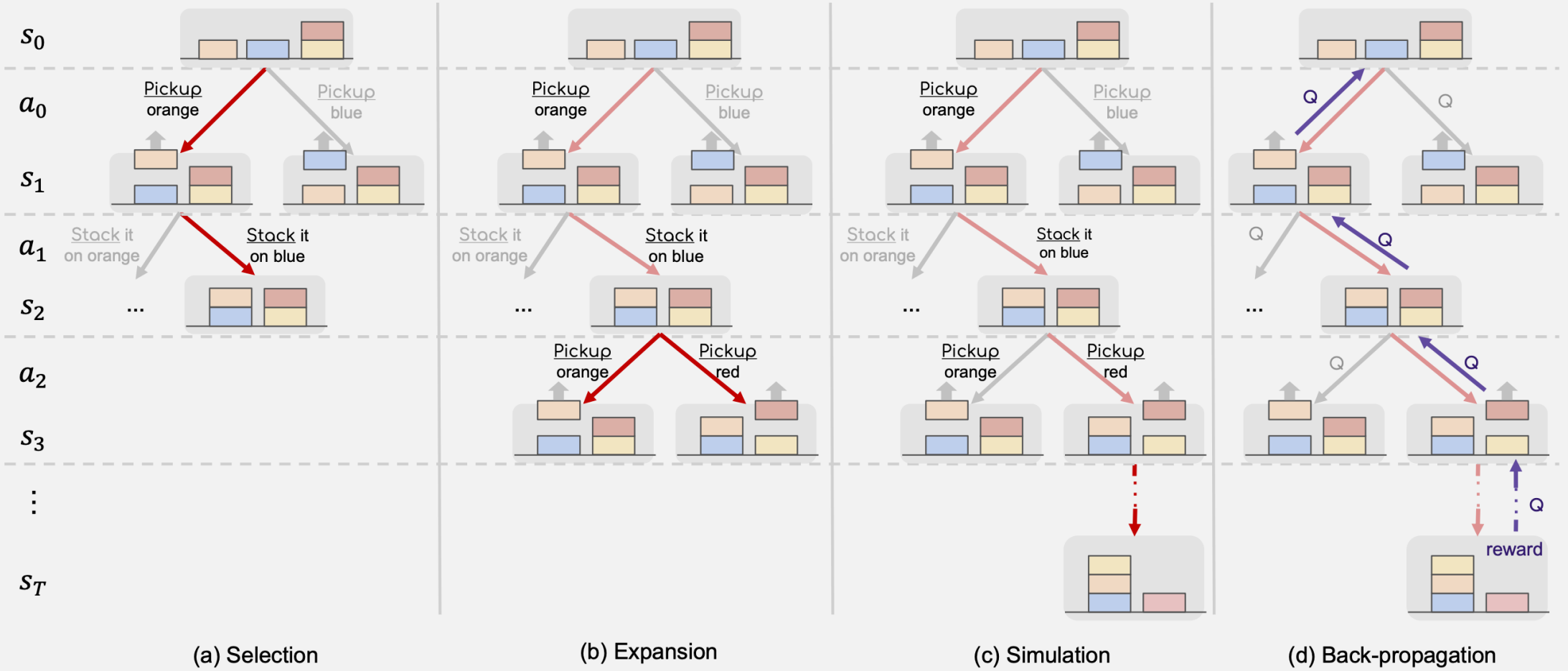
Planning with Monte Carlo Tree Search

- Q(s,a) estimates the expected future reward of taking action a in state s

$$a^* = \arg\max_{a \in A(s)} \left[Q(s,a) + w\sqrt{\frac{\ln N(s)}{N(c(s,a))}} \right]$$

- Aggregation: RAP could produce multiple traces and answers from different MCTS iterations, which will be aggregated to produce the final answer

Hao et al., Reasoning with Language Model is Planning with World Model, 2023

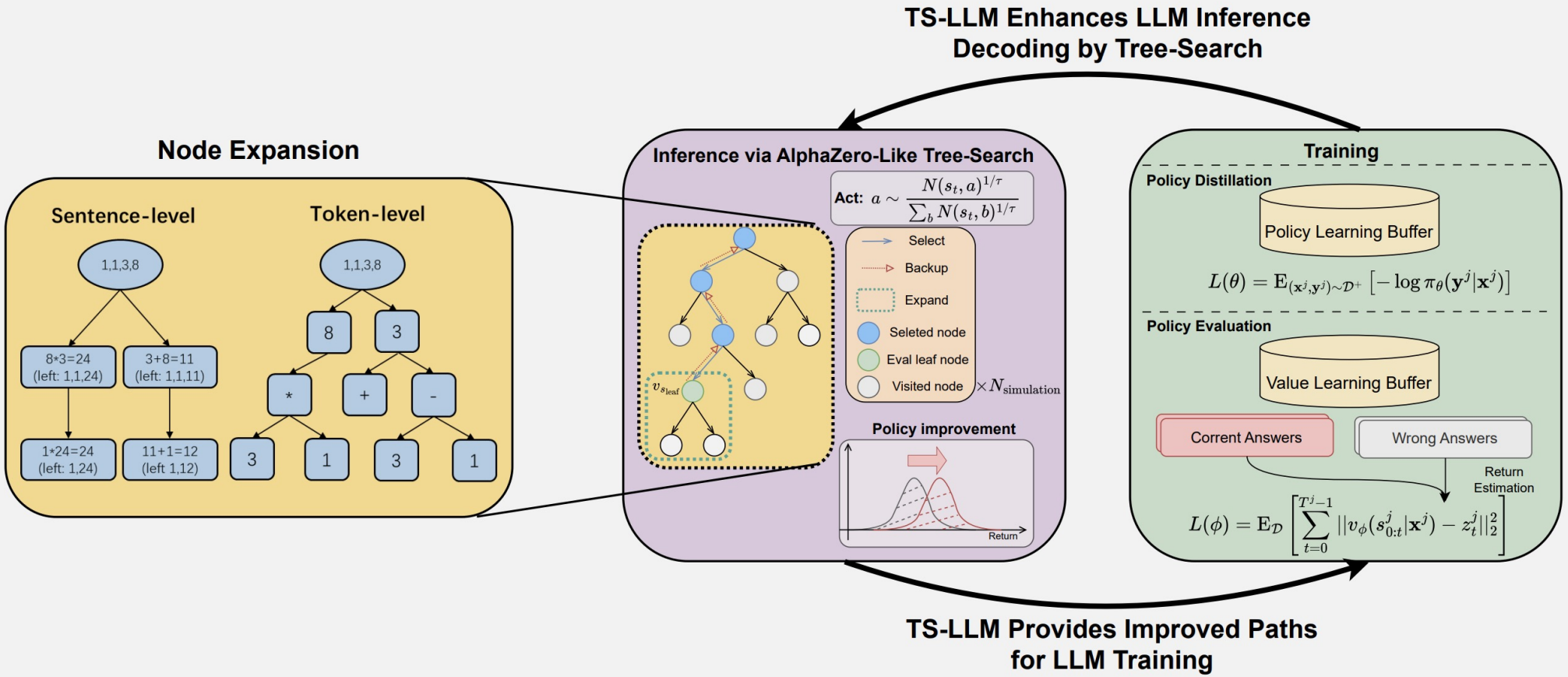


Hao et al., Reasoning with Language Model is Planning with World Model, 2023

Which problem does remain?

- First, the value functions in the tree-search algorithms are obtained by prompting LLMs.
  - As a result, such algorithms lack general applicability and heavily rely on both well-designed prompts and the robust capabilities of advanced LLMs.
  - Also, such prompt-based self-evaluation is not always reliable
- restricting their capabilities to relatively simple and shallow tasks.

AlphaZero-Like Tree-Search can Guide LLM



X. Feng et al., AlphaZero-Like Tree-Search can Guide Large Language Model Decoding and Training, 2023

ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search

Dan Zhang[1*†], **Sining Zhoubian**[1*], **Ziniu Hu**[2*], **Yisong Yue**[2], **Yuxiao Dong**[1], **Jie Tang**[1]
[1]The Knowledge Engineering Group (KEG), Tsinghua University;
[2]California Institute of Technology
{zd21,zbsn21}@mails.tsinghua.edu.cn

https://rest-mcts.github.io/

Research Gap

- Models can generate wrong or useless intermediate reasoning steps, while finding the correct solution by chance
  - Thus, self-training dataset can often contain many false positives
- One way to tackle this issue is to use a value function or reward model to verify reasoning traces for correctness
- Training a reliable reward model to verify every step generally depends on dense human-generated annotations

Reward Model

- MCTS requires a value function:
  - ORM r_ϕ to prune branches, evaluate final solutions, and backup value.
    - It is costly and inefficient.
  - Recent work (X. Feng et al., 2023) suggests using a learned LLM value function in MCTS so the backup process can happen in the intermediate step
    - It greatly improves search efficiency but still relies on an ORM to select the final answer.

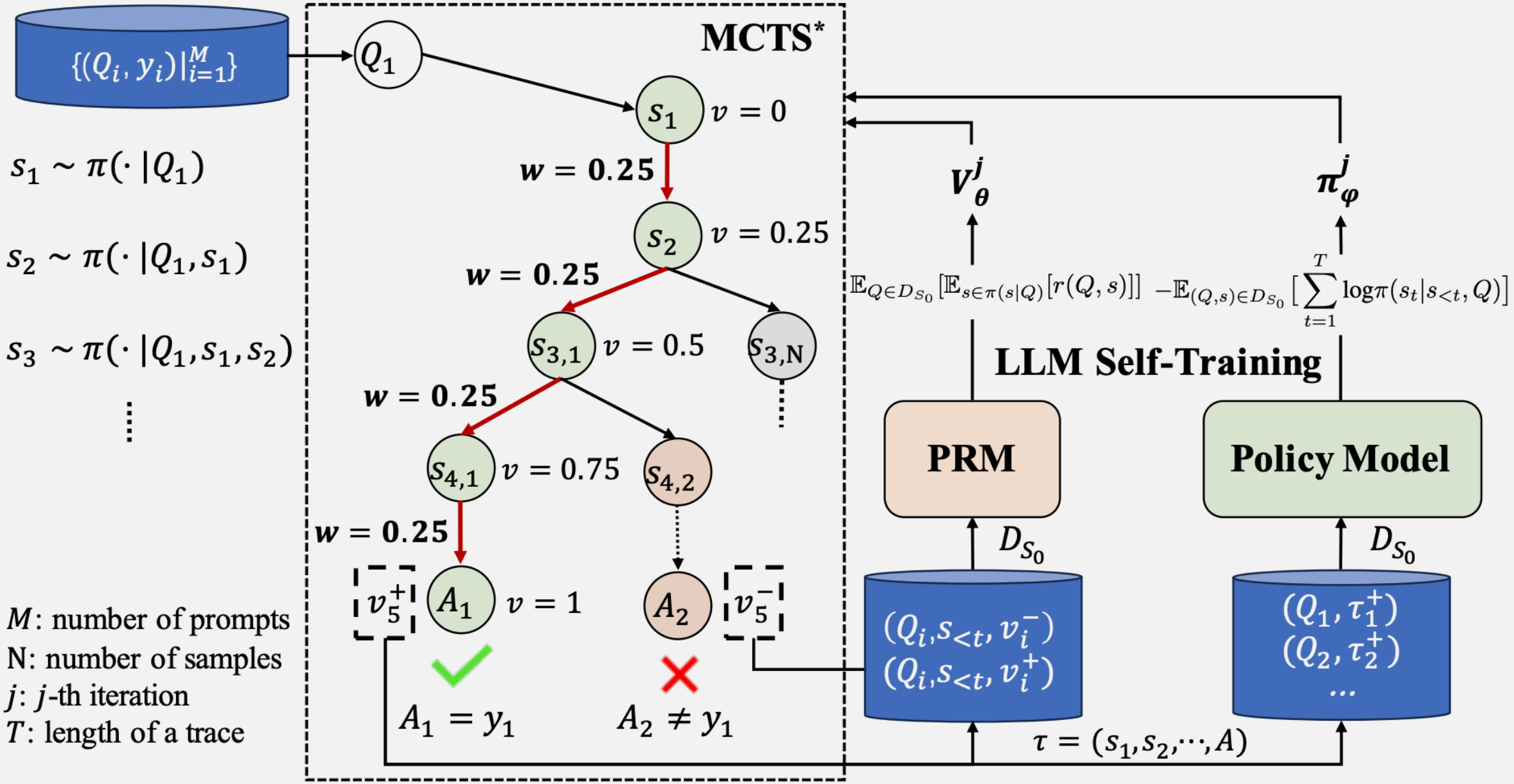
ReST-MCTS*: Main Idea

- Automatically acquiring high-quality reasoning traces and effectively process reward signals for verification and LLM self-training
- Training LLMs using model-based RL training
- A modified MCTS algorithm as the reasoning policy, denoted MCTS*, guided by a trained per-step process reward (value) model

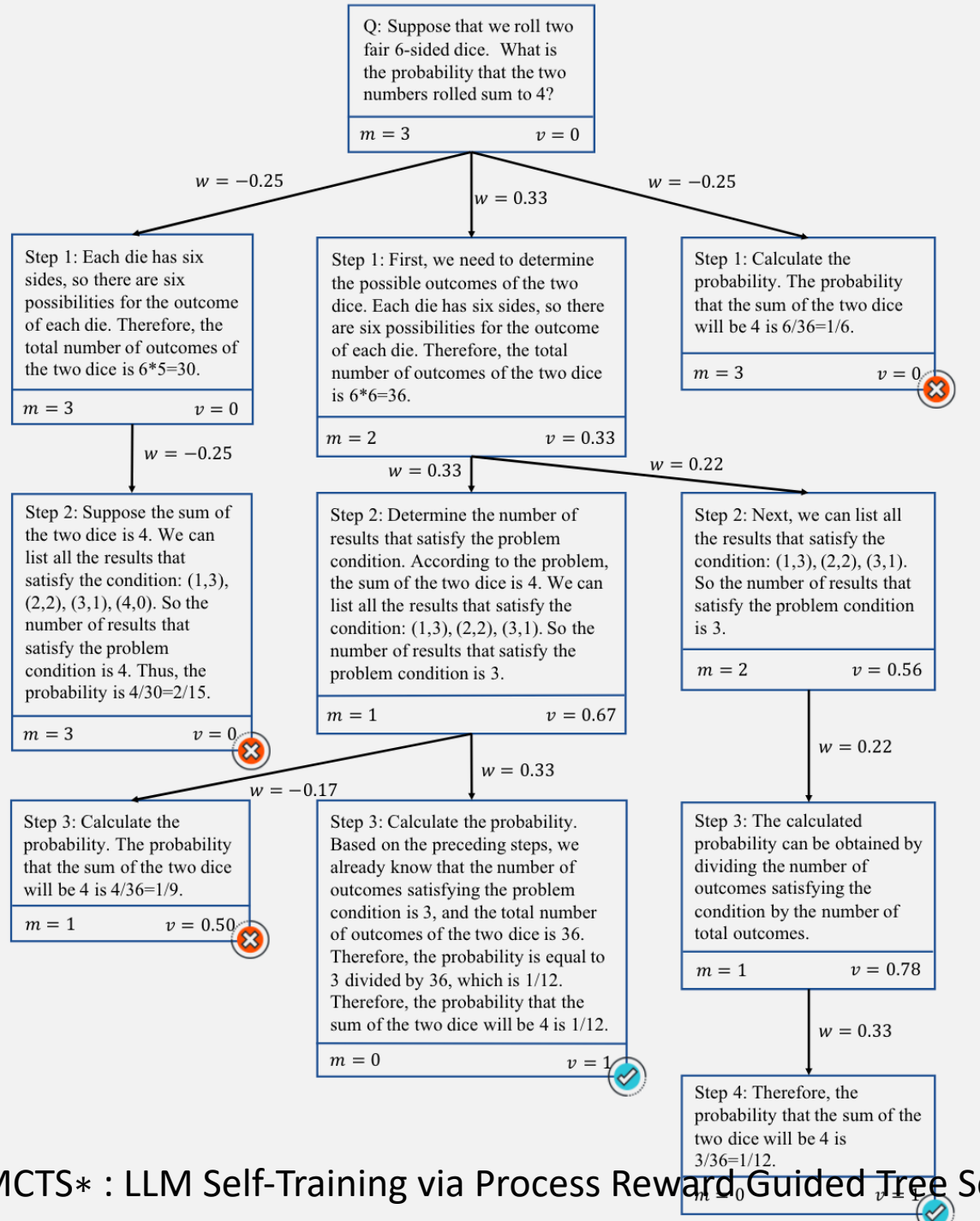
ReST-MCTS*

- Automatically generating per-step labels for training reward models
  - by a sufficient number of rollouts
  - effectively provides high quality samples, without requiring human intervention (like Math-Shepherd).

- Self-Training: This process can repeat for multiple iterations:
  - First step: sampling multiple reasoning traces using π (tree-structured traces)
  - Second step: the reasoning traces are used to fine-tune π.



D. Zhang, ReST-MCTS∗ : LLM Self-Training via Process Reward Guided Tree Search, NeurIPS 2024



D. Zhang, ReST-MCTS∗ : LLM Self-Training via Process Reward Guided Tree Search, NeurIPS 2024

Self-training pipeline

- Based on MCTS*‘s reasoning traces, **self-improvement** is performed on the reasoning **policy** and **PRM**
  - Instruction Generation: initialization starts from an original dataset D_0 for the training PRM, V_θ.
  - Collect process reward data for PRM
  - Collect reasoning traces for policy model
  - Mutual self-training for process reward model and policy model

Self-training pipeline

- **Instruction Generation**: initialization starts from an original dataset D_0 for the training PRM, V_θ.
  - Fine-grained dataset for science and math where each question is paired with a correct step-by-step solution.
  - The generated single steps by employing an LLM policy (ChatGLM2) that is basically incompetent for reasoning tasks of this difficulty make new partial solutions that are regarded as incorrect.
- Collect process reward data for PRM
- Collect reasoning traces for policy model
- Mutual self-training for process reward model and policy model

Self-training pipeline

- Instruction Generation
- **Collect process reward data for PRM**: the target quality value of partial solutions of nodes near a correct reasoning path on the pruned search tree is derived.
- Collect reasoning traces for policy model
- Mutual self-training for process reward model and policy model

Self-training pipeline

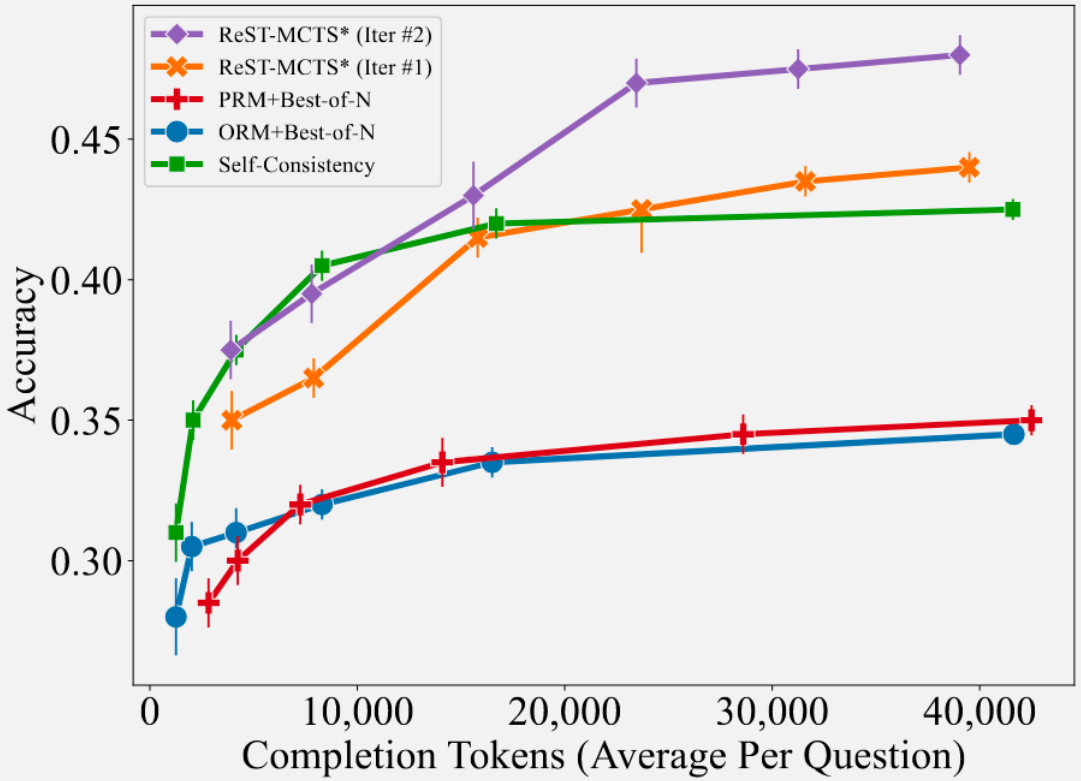
- Instruction Generation
- Collect process reward data for PRM
- **Collect reasoning traces for policy model**
  - First, all the unfinished branches (branches that do not reach a final answer) are pruned.
  - Then, other traces' final answers are verified according to their correctness through simple string matching or LLM judging
  - The correct ones are selected for policy improvement
- Mutual self-training for process reward model and policy model

Self-training pipeline

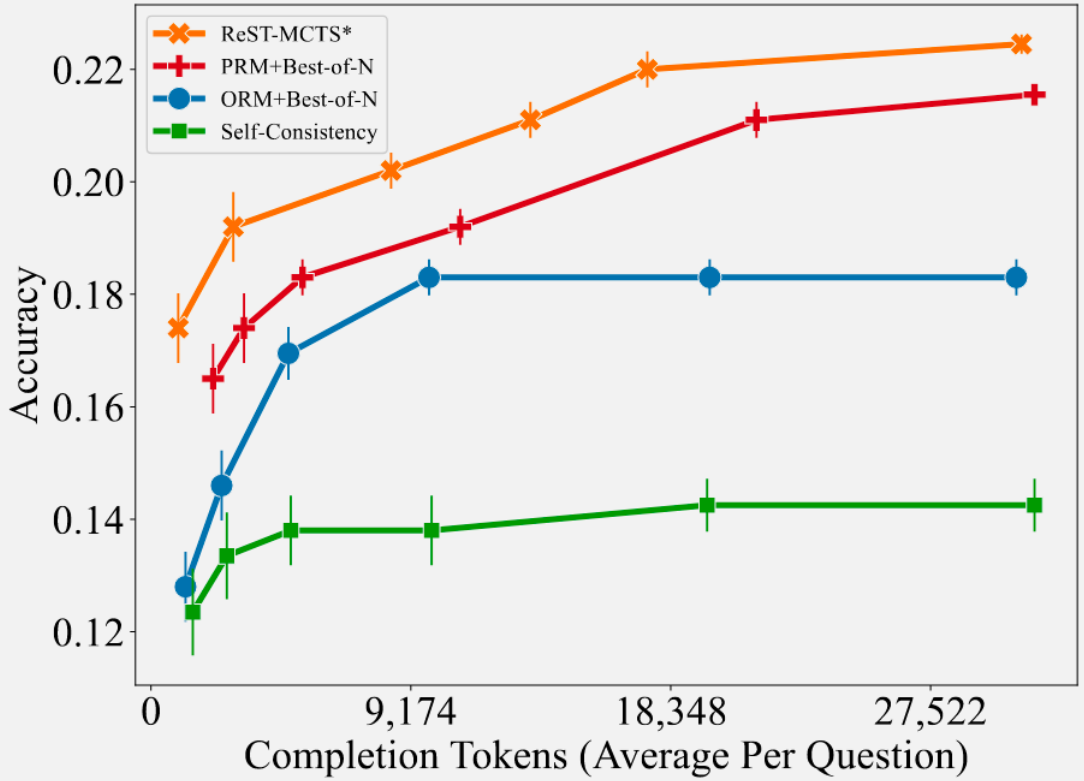
- Instruction Generation
- Collect process reward data for PRM
- Collect reasoning traces for policy model
- **Mutual self-training for process reward model and policy model**
  - π is used to perform MCTS* and generate solutions
  - In the i-th ($i = 1, 2, \ldots$) iteration
    - V_θ is trained with D_{V_i-1} to obtain V_i
    - π_{S_i-1} is trained on D_{G_i} to generate new generator π_{S_i}.
    - At the same time, D_{G_i} drives the update of V_i to V_{i+1}.

PRM Guided Tree MCTS*

- MCTS*: comprises four main stages in each iteration
  - node selection
  - thought expansion
  - greedy MC rollout
  - value backpropagation.
- For each tree node C, we have $C = (p_C, n_C, v_C)$
  - p_C: a series of thoughts or steps as a partial solution
  - n_C: number of visits
  - v_C: quality value



(a) Self-training of value model on MATH.



(b) Comparison of value model on SciBench.

D. Zhang, ReST-MCTS∗ : LLM Self-Training via Process Reward Guided Tree Search, NeurIPS 2024

Models	Dataset	SC	ORM	SC+ORM	MS	SC + MS	SC + ReST-MCTS* (Value)
Mistral-7B: MetaMATH	GSM8K	83.9	86.2	86.6	87.1	86.3	**87.5**
	MATH500	35.1	36.4	38.0	37.3	38.3	**39.0**

Models	Subject	Chemistry				Physics			Math			All
	Method	atkins	chemmc	quan	matter	fund	class	thermo	diff	stat	calc	Ave.
GLM4	CoT	11.21	23.07	8.82	4.08	19.44	2.12	**7.46**	10.00	12.00	28.57	12.68
	ToT	11.21	23.07	8.82	**12.24**	**22.22**	**6.38**	5.97	**12.00**	25.33	**30.95**	15.82
	ReST-MCTS*	**13.08**	**28.20**	**14.70**	8.16	**22.22**	4.25	**7.46**	**12.00**	**26.66**	**30.95**	**16.77**
GPT-3.5-turbo	CoT	5.60	7.69	5.88	**6.12**	6.94	2.12	**2.98**	4.00	16.00	**11.90**	6.92
	ToT	**8.41**	**12.82**	**11.76**	**6.12**	**11.11**	0.00	0.00	**10.00**	18.66	9.52	8.44
	ReST-MCTS*	5.60	**12.82**	**11.76**	**6.12**	6.94	**8.51**	**2.98**	**10.00**	**24.00**	**11.90**	**10.06**
LLaMA2-13B-Chat	CoT	**2.80**	2.56	**2.94**	2.04	2.77	**2.12**	0.00	2.00	2.66	**2.38**	2.23
	ToT	0.93	**5.12**	**2.94**	**4.08**	2.77	0.00	**1.49**	0.00	4.00	**2.38**	2.37
	ReST-MCTS*	0.93	**5.12**	**2.94**	2.04	**4.16**	**2.12**	0.00	**4.00**	**5.53**	**2.38**	**2.90**

D. Zhang, ReST-MCTS∗ : LLM Self-Training via Process Reward Guided Tree Search, NeurIPS 2024

Algorithm 1: Mutual self-training ReST-MCTS* for value model and policy model.

Input: base LLM π, original dataset for policy model D_{S_0}, original dataset for value model D_0, new problem set D_G, number of solutions N, j-th solution A_j, correct solution a^*, value model V_θ, weighted value function w, quality value function v, number of iterations T.

$\pi_{S_0} \leftarrow \text{SFT}(\pi, D_{S_0})$ // fine-tune generator
$D_{V_0} \leftarrow \text{generate\_value\_data}(D_0, w, v)$ // initialize train set for value model
$V_0 \leftarrow \text{train\_value\_model}(V_\theta, D_{V_0})$ // initialize value model
for $i = 1$ to T **do**
  $D_{G_i} \leftarrow \text{generate\_policy\_data}(\pi_{S_{i-1}}, V_{i-1}$ guided MCTS*, $D_G, N)$ // generate synthetic data for policy model
  for $j = 1$ to N **do**
    $D_{G_i(A_j=a^*)} \leftarrow \text{label\_correctness}(D_{G_i})$ // match and select correct solutions
  end for
  $\pi_{S_i} \leftarrow \text{SFT}(\pi_{S_{i-1}}, D_{G_i(A_j=a^*)|_{j=1}^N})$ // self-training policy model
  $D_{V_i} \leftarrow \text{extract\_value\_data}(D_{G_i})$ // collect process reward and extract value data
  $V_i \leftarrow \text{train\_value\_model}(V_{i-1}, D_{V_i})$ // self-training value model
end for

Output: π_{S_T}, V_T

D. Zhang, ReST-MCTS∗ : LLM Self-Training via Process Reward Guided Tree Search, NeurIPS 2024

Summary

- Search (or sampling) + LLMs
  - Verifiers
- SFT or RL as post-training on LLMs
- Planning along Learning