Two papers:

- Reasoning with Language Model is Planning with World Model
- RL Planning and AI Planning: A Primer and Survey (Preliminary Report)

Reasoning with Language Model is Planning with World Model

Radowan Mahmud Redoy (snf4za)

Rishov Paul (vst2hb)

Introduction to Problem

LLMs have remarkable reasoning capabilities with techniques such as

- CoT (chain-of-thought)
- Least-to-most prompting

Creating action plans to move a block to a target state

GPT-3 success rate 1%

78% Success rate 

Introduction to Problem

To make an action plan towards a goal, planning involves following process

Exploring various alternative courses of actions

Assessing the likely outcomes by rolling out possible future scenarios

Iteratively refining the plan based on the assessment

LLMs struggle with complex tasks that require multiple steps of ***Math, Logical or Commonsense reasoning***

Introduction to Problem

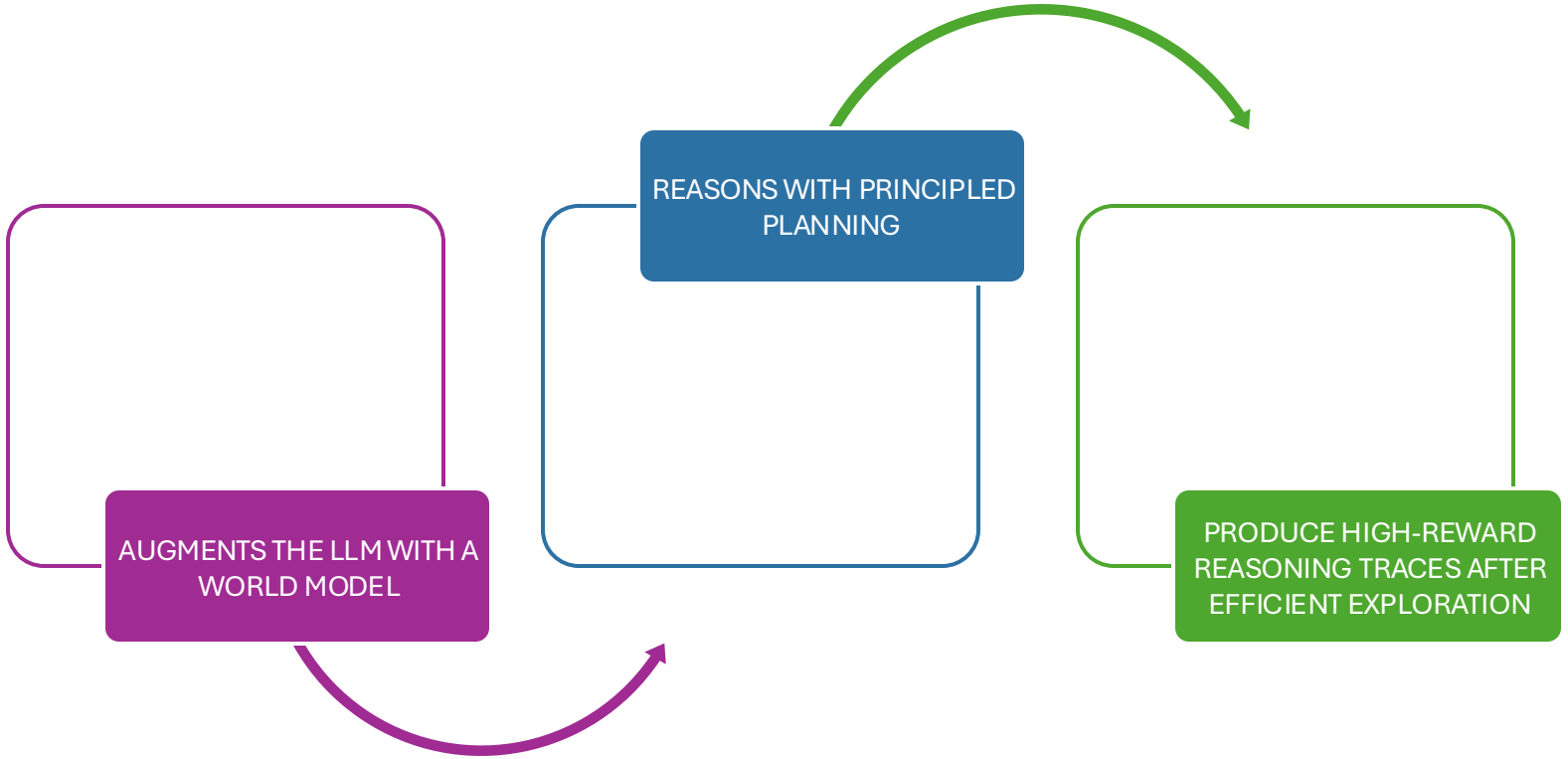
key limitations of the current reasoning with LLMs,

Lack of an internal world knowledge to simulate the state of the world

Absence of a reward mechanism to assess and guide the reasoning towards the desired state

Incapability of balancing exploration vs. exploitation to efficiently explore vast reasoning space.

Reasoning via Planning (RAP)



RAP (Approach)

Build the world model by repurposing the LLM with prompting

Introduce the rewards for assessing each state during reasoning

Guided by the world model and rewards, plan with Monte Carlo Tree Search and explore reasoning space to find optimal reasoning traces

Finally, when multiple promising reasoning traces are acquired during planning, introduce an aggregation method

World Model

A world model predicts the next state of the reasoning after applying an action to the current state

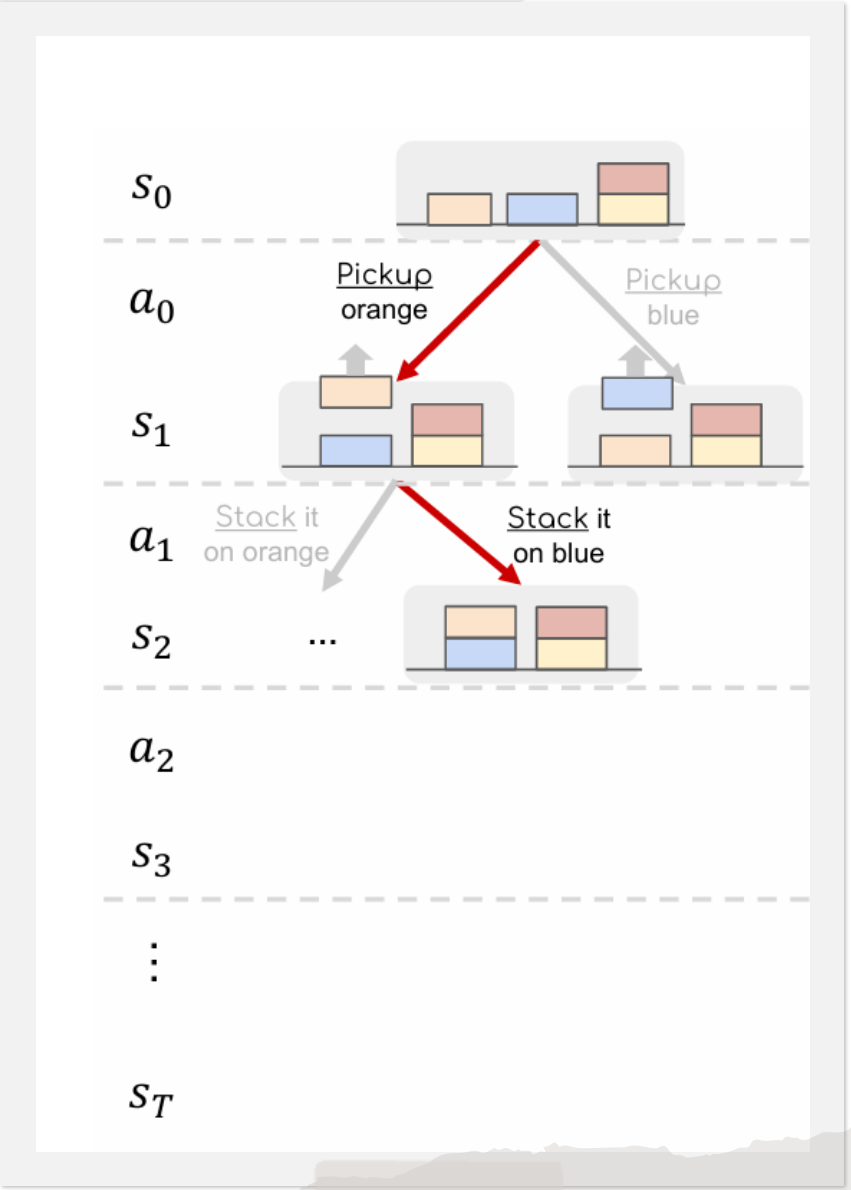
World	State	Action
Blocks World	Configuration of blocks	Moving a block (e.g., "pickup the orange block")
Math Reasoning	Values of intermediate variables	Subquestion that derives new values
Logical Reasoning	Fact being focused on	Choosing a rule for the next deduction

Example: **NVIDIA Cosmos World Foundation Models** A family of pretrained multimodal models that developers can use out-of-the-box for world generation and reasoning,

Language Model as World Model

With the definition of state and action, the reasoning process can be described as a **Markov decision process (MDP)**

- Taking s0, the LLM generates an action space by sampling from its generative distribution at ~ p(a|st, c), where c is a proper prompt .
- Once an action is chosen, the world model then predicts the next state st+1 of the reasoning.



Action plan generation prompt

```
stack the orange block on top of the
    red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the
    orange block is clear, the yellow
    block is clear, the hand is empty,
    the blue block is on top of the red
    block, the orange block is on top
    of the blue block, the red block is
    on the table and the yellow block
    is on the table.
My goal is to have that the blue block
    is on top of the red block and the
    yellow block is on top of the
    orange block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the
    orange block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the
    red block is clear, the blue block
    is clear, the orange block is
    clear, the hand is empty, the blue
    block is on top of the yellow
    block, the red block is on the
    table, the orange block is on the
    table and the yellow block is on
    the table.
My goal is to have that the blue block
    is on top of the orange block and
    the yellow block is on top of the
    red block.

My plan is as follows:

[PLAN]
unstack the blue block from on top of
    the yellow block
stack the blue block on top of the
    orange block
pick up the yellow block
stack the yellow block on top of the
    red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the
    red block is clear, the blue block
    is clear, the yellow block is
    clear, the hand is empty, the
    yellow block is on top of the
    orange block, the red block is on
    the table, the blue block is on the
    table and the orange block is on
    the table.
My goal is to have that the orange
    block is on top of the blue block
    and the yellow block is on top of
    the red block.

My plan is as follows:

[PLAN]
unstack the yellow block from on top of
    the orange block
stack the yellow block on top of the
    red block
pick up the orange block
stack the orange block on top of the
    blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that,
    <initial_state>
My goal is to have that <goals>.

My plan is as follows:

[PLAN]
```

Plan Generation: Task setup

- **Environment:** *Blocksworld* benchmark (Valmeekam et al., 2022)
  - Goal: Rearrange blocks into target stacks.
- **State Definition:**
  - Current orientation of blocks.
- **Action Definition:**
  - 4 verbs: **STACK**, **UNSTACK**, **PUT**, **PICKUP** + objects.
  - Valid actions generated based on current state & domain rules.

Initial State: The orange block is on the table, the blue block is on the table, and the red block…
Goal: The orange block is on the blue block, and the yellow block is on the orange block.

Plan Generation: Task Setup

- **State Transition:**
  - LLM predicts state changes after each action.
  - State updated by adding/removing block conditions that are no longer true.
- **Termination:**
  - Ends when goal conditions met or depth limit reached.
- **Rewards:**
  - **r1:** LLM log-probability of action (intuitive, near-goal).
  - **r2:** Heuristic reward based on goal conditions met.
  - Super high reward when all goal conditions are satisfied.

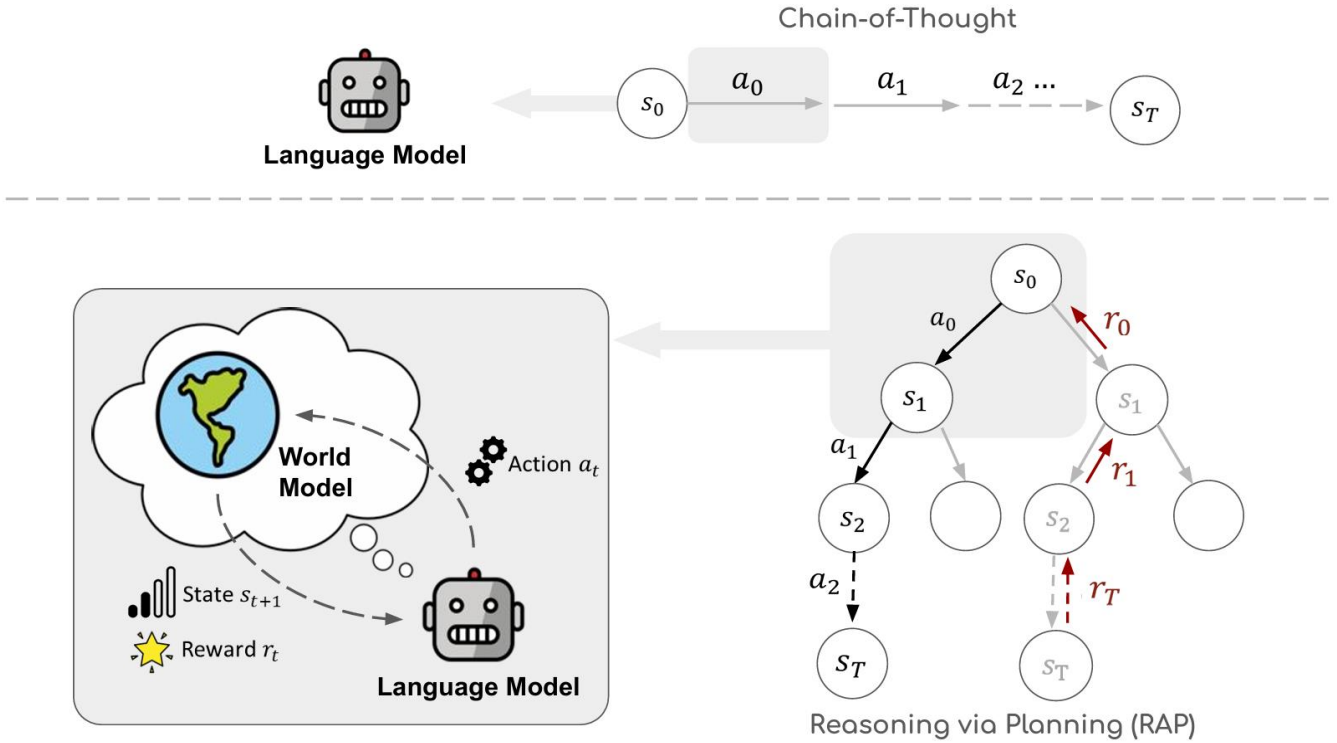
Next state prediction prompt

```
I am playing with a set of blocks where
    I need to arrange the blocks into
    stacks. Here are the actions I can
    do

Pick up a block
Unstack a block from on top of another
    block
Put down a block
Stack a block on top of another block

I have the following restrictions on my
    actions:
I can only pick up or unstack one block
    at a time.
```

```
I can only pick up or unstack a block
    if my hand is empty.
I can only pick up a block if the block
    is on the table and the block is
    clear. A block is clear if the
    block has no other blocks on top of
    it and if the block is not picked
    up.
I can only unstack a block from on top
    of another block if the block I am
    unstacking was really on top of the
    other block.
I can only unstack a block from on top
    of another block if the block I am
    unstacking is clear. Once I pick up
    or unstack a block, I am holding
    the block.
I can only put down a block that I am
    holding.
I can only stack a block on top of
    another block if I am holding the
    block being stacked.
I can only stack a block on top of
    another block if the block onto
    which I am stacking the block is
    clear. Once I put down or stack a
    block, my hand becomes empty.

After being given an initial state and
    an action, give the new state after
    performing the action.
```

```
[SCENARIO 1]
[STATE 0] I have that, the white block
    is clear, the cyan block is clear,
    the brown block is clear, the hand
    is empty, the white block is on top
    of the purple block, the purple
    block is on the table, the cyan
    block is on the table and the brown
    block is on the table.
[ACTION] Pick up the brown block.
[CHANGE] The hand was empty and is now
    holding the brown block, the brown
    block was on the table and is now
    in the hand, and the brown block is
    no longer clear.
[STATE 1] I have that, the white block
    is clear, the cyan block is clear,
    the brown block is in the hand, the
    hand is holding the brown block,
    the white block is on top of the
    purple block, the purple block is
    on the table and the cyan block is
    on the table.

[SCENARIO 2]
[STATE 0] I have that, the purple block
    is clear, the cyan block is clear,
    the white block is clear, the hand
    is empty, the white block is on top
    of the brown block, the purple
    block is on the table, the cyan
    block is on the table and the brown
    block is on the table.
[ACTION] Pick up the cyan block.
[CHANGE] The hand was empty and is now
    holding the cyan block, the cyan
    block was on the table and is now
    in the hand, and the cyan block is
    no longer clear.
[STATE 1] I have that, the cyan block
    is in the hand, the white block is
    clear, the purple block is clear,
    the hand is holding the cyan block,
    the white block is on top of the
    brown block, the purple block is on
    the table and the brown block is on
    the table.

[SCENARIO 3]
[STATE 0] <state>
[ACTION] <action>
[CHANGE]
```

Traditional Method vs RAP



Case Study: Comparison of COT and RAP

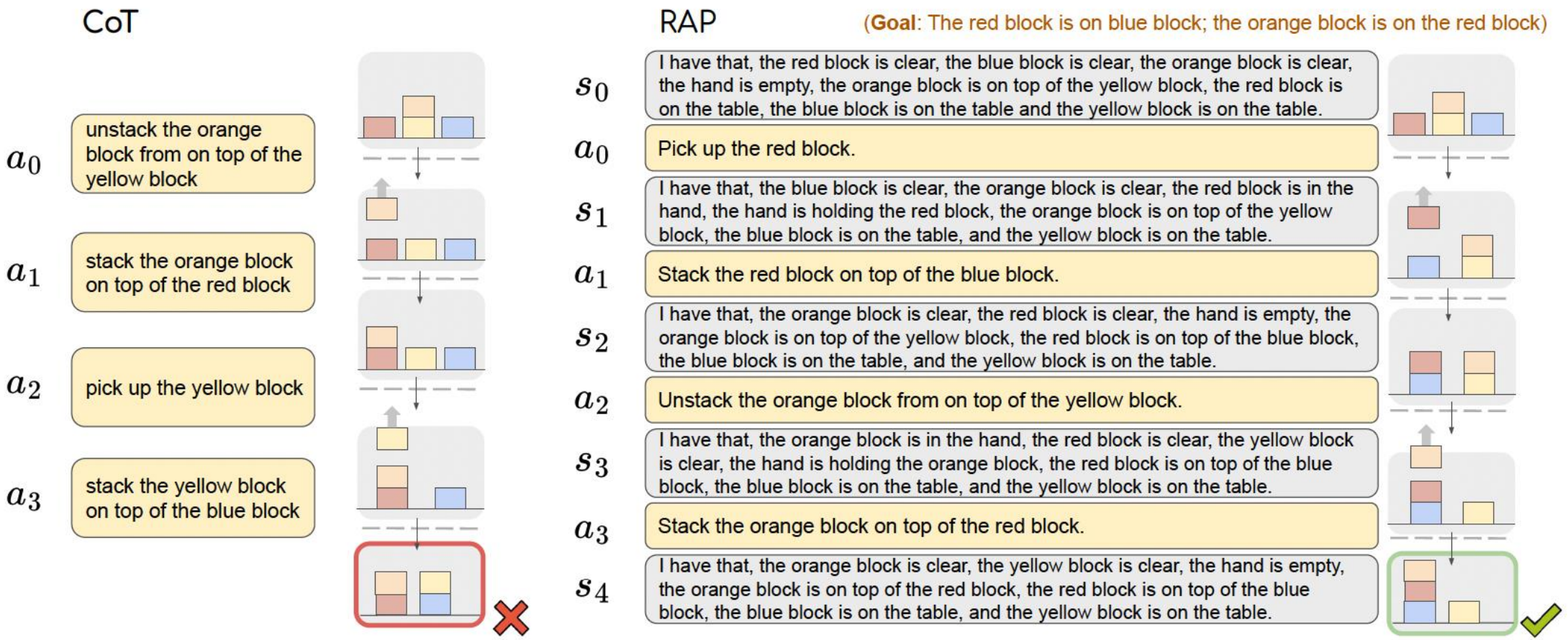


Figure 4: Comparing reasoning traces in Blocksworld from CoT (left) and RAP (right).

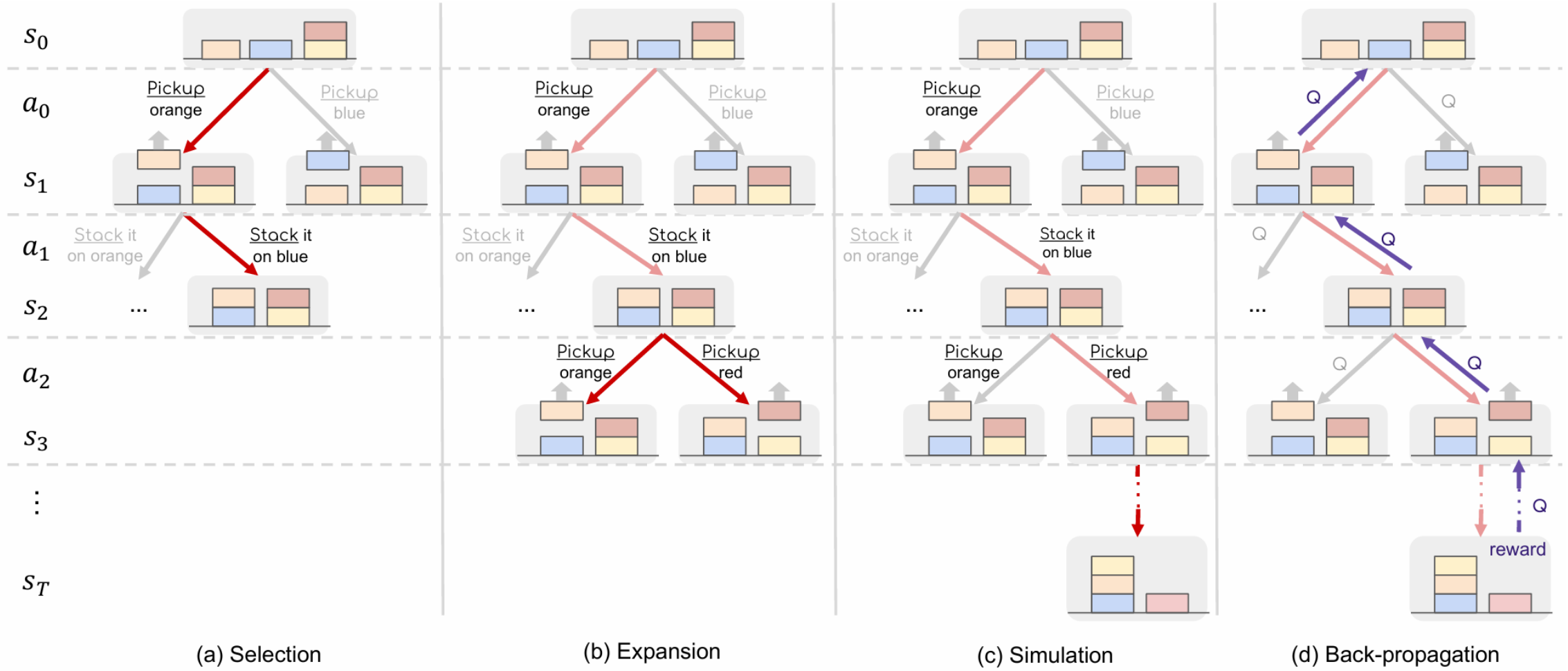
Planning with Monte Carlo Tree Search

Once equipped with the world model and rewards LLMs can reason with any planning algorithms.

Monte Carlo Tree Search (MCTS)

A powerful planning algorithm that strategically explores the space of reasoning trees and strikes a proper balance between exploration and exploitation to find high-reward reasoning traces efficiently.

Planning with Monte Carlo Tree Search



Reward Design

Likelihood of the action

- Log probability of the action as a reward
- Reflects the "in stinct" of LLMs as an agent

Confidence of the state

- Draw multiple sample answers from the world model
- Use the proportion of the most frequent answer as the confidence.

Self-evaluation by the LLM

- It's beneficial to allow the LLM to criticize itself with the question
- The reward evaluates LLM's own estimation of the correctness of reasoning.

Task-specific heuristics

- RAP also allows us to flexibly plugin other task-specific heuristics into the reward function.

Experiments and Results

Demonstrate the flexibility and effectiveness of our **RAP framework** by applying it to a wide range of problems

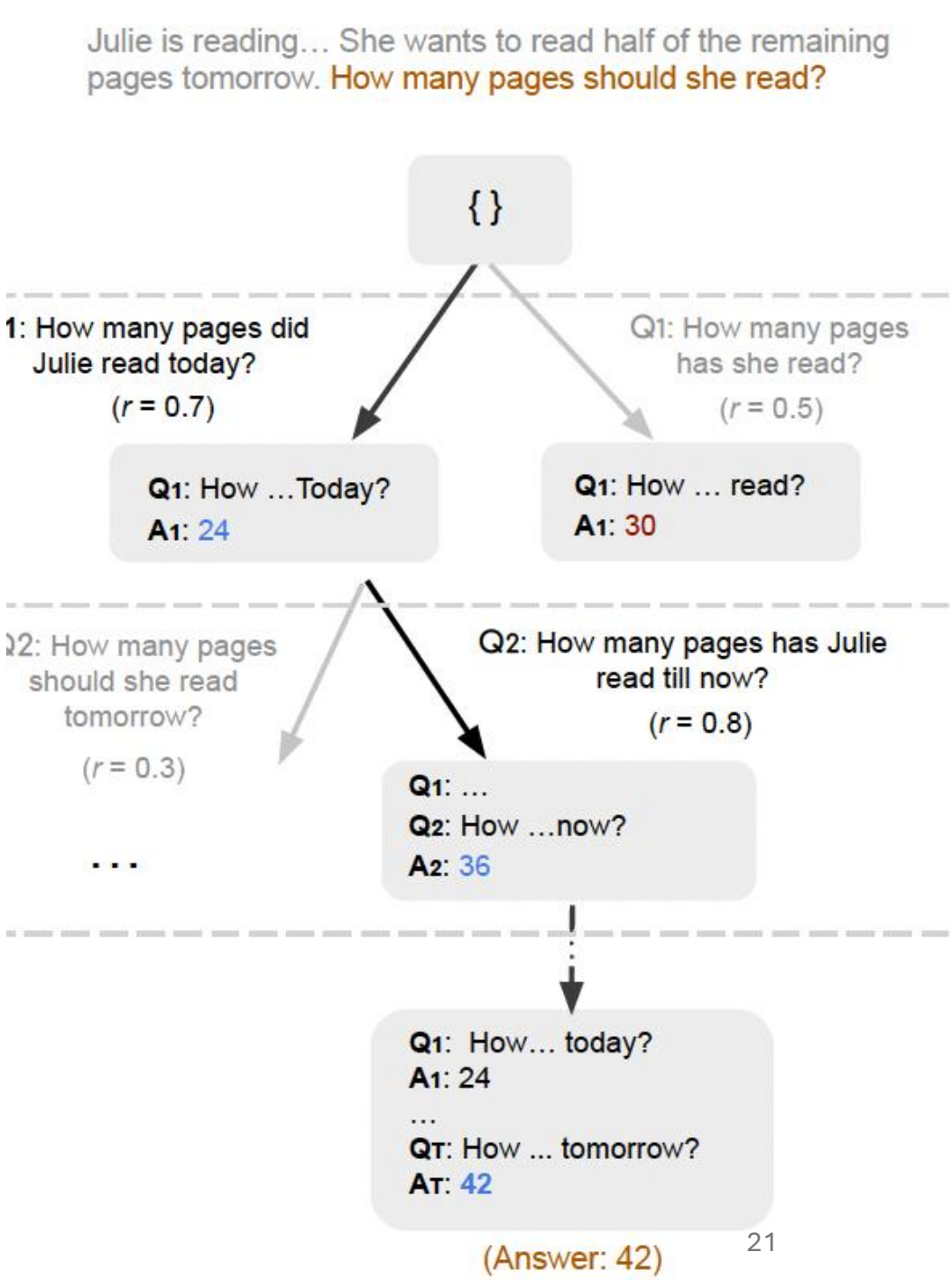
Results of Plan Generation

Method	2-step	4-step	6-step
CoT	0.17	0.02	0.00
CoT - pass@10	0.23	0.07	0.00
CoT (GPT-4)	0.50	0.63	0.40
RAP$^{(10)}$	1.00	0.86	0.26
RAP$^{(20)}$	**1.00**	**0.88**	**0.42**

Table 1: Results on Blocksworld. RAP$^{(10)}$ and RAP$^{(20)}$ refer to our method where the iteration number is set to 10 and 20, respectively. "pass@10" means 10 plans are sampled for each test case, and the test case is regarded as solved if at least one plan is correct. All other settings including RAP, only evaluate a single plan.

Math Reasoning: Task setup

- Dataset:
  - **GSM8k** - consists of grade school math word problems
- ***Input:*** *Description + Final Question*
- RAP decompose the final question into a sequence of smaller sub-questions
- **State**: Tracks the values of intermediate variables
- **Action**: incremental sub-question proposal about a unknown intermediate variable
- **World Model Response:** *Answers sub-question & updates state with new intermediate value*
- *RAP reviews updated state to inform next sub-question*



Math Reasoning: reward process

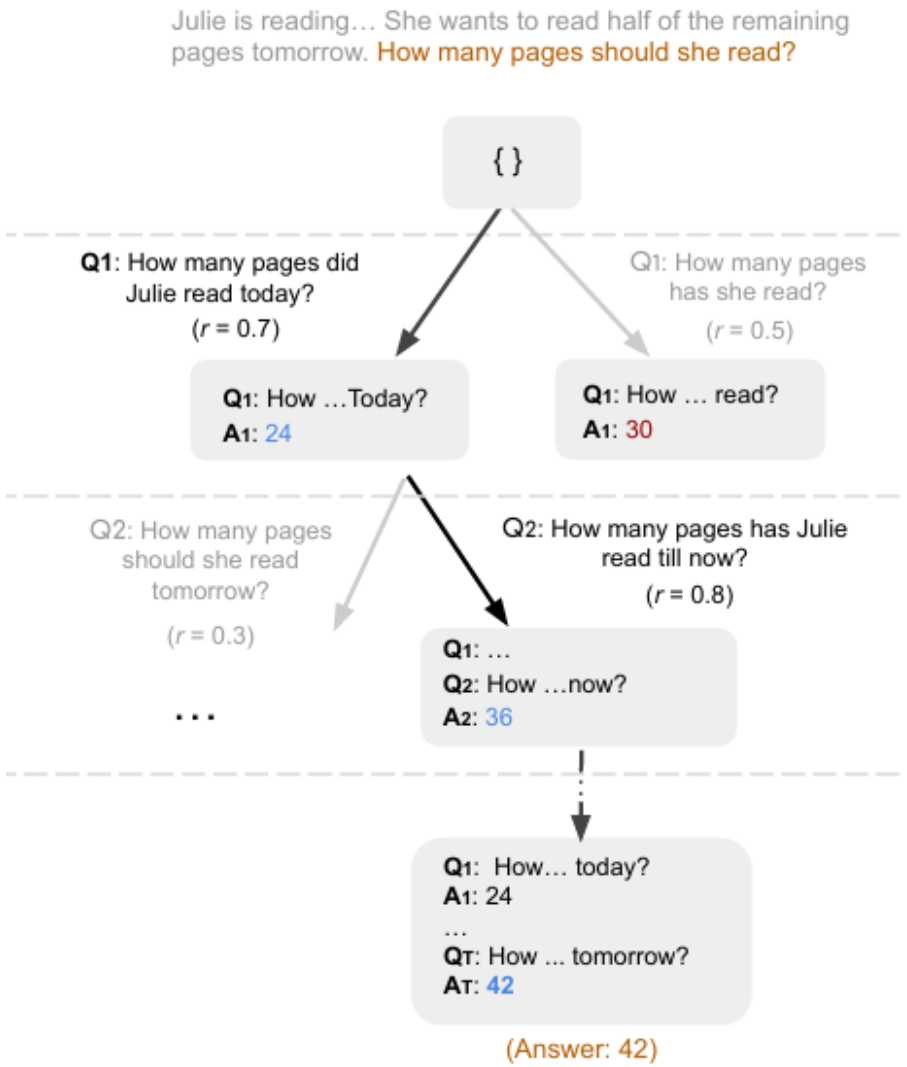
- **Reward (rt):**
  - Combines *LLM self-evaluation* & *state confidence*.
  - **$rt = r_{t,1}^{\alpha} * r_{t,2}^{(1-\alpha)}$**
  - encourages more relevant and useful sub-questions
- **Q-Value:** Considers future steps reward averages to optimize reasoning path

$$Q^*(s_t, a_t) = \max_{s_t, a_t, r_t, \ldots, s_l, a_l, r_l, s_{l+1}} \operatorname{avg}(r_t, \ldots, r_l).$$

RAP-Aggregation

- For problems, such as math reasoning where only the final answer is required, RAP could produce multiple traces and answers from different MCTS iterations, which will be aggregated to produce the final answer.

Note that problems like **plan generation or logical inference** require a complete reasoning trace as output; thus, **RAP Aggregation will not be applied**.



Math Reasoning: Results

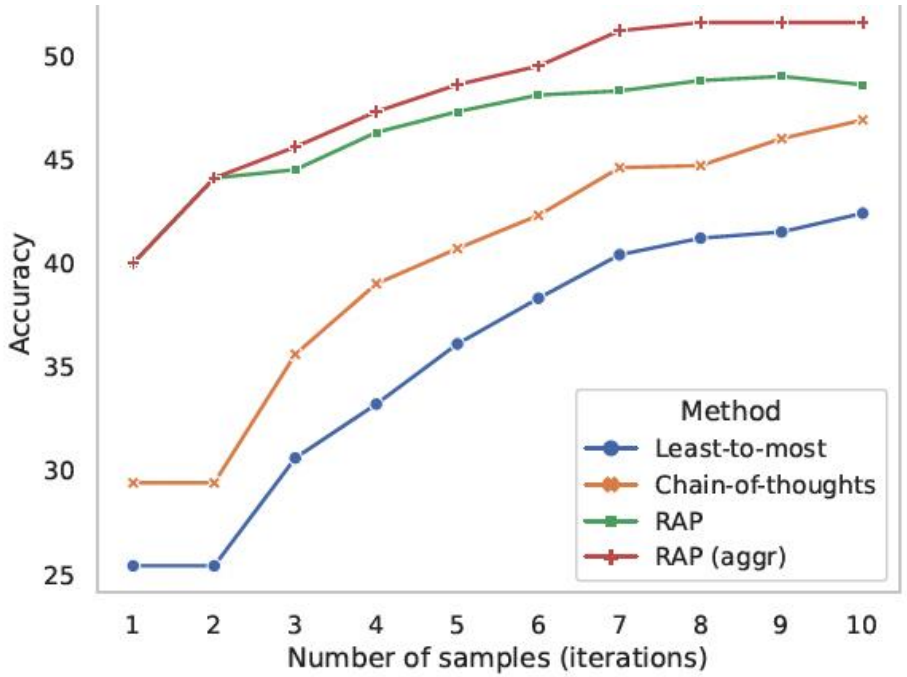


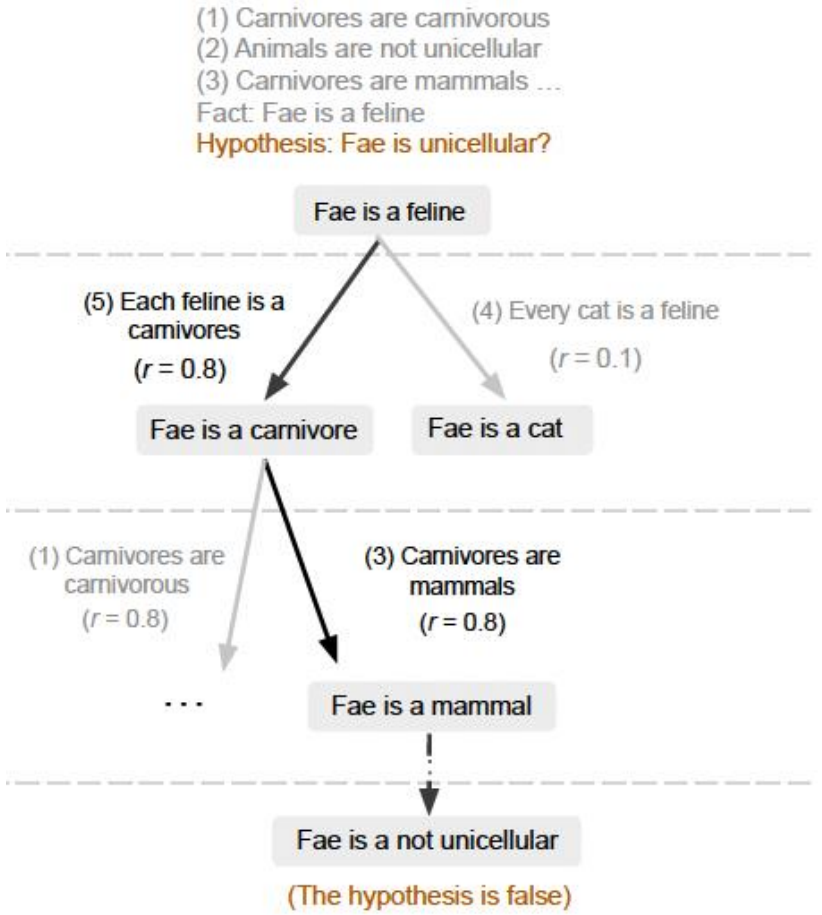
Figure 5: Results on GSM-8K, with different numbers of sampled paths or iterations.

Method	Accuracy (%)
Chain-of-Thought	29.4
+ $SC^{(10)}$	46.8
Least-to-Most	25.5
+ $SC^{(10)}$	42.5
$RAP^{(1)}$	40.0
$RAP^{(10)}$	48.6
+ aggr	**51.6**

Table 2: Results on GSM8k. The superscripts indicat the number of samples or iterations.

Logical Reasoning with RAP: Task Setup

- **Dataset:** PrOntoQA (Saparov & He, 2022)
- **Objective:**
  - Verify if a hypothesis fact is **true or false**.
  - Provide a **step-by-step proof** supporting the answer.
- **Problem Setup:**
  - Input: Set of facts + logical rules.
  - Output: Final answer **(True/False)** + logical proof.
- **RAP Framework Setup:**
  - **State:** Current fact in focus
  - **Action:** Select a rule from the fact set.



Logical Reasoning: Process and Reward

Reasoning Process:

World Model: Executes one-hop reasoning (apply selected rule).

Produces new fact → updates state.

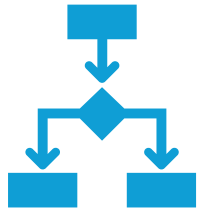
Reward Design:

Self-Evaluation: LLM scores the quality of reasoning steps.

Few-shot examples provided to LLM for better self-assessment.

Q-value Update:

- Uses average rewards of future steps (similar to GSM8k reward formulation).



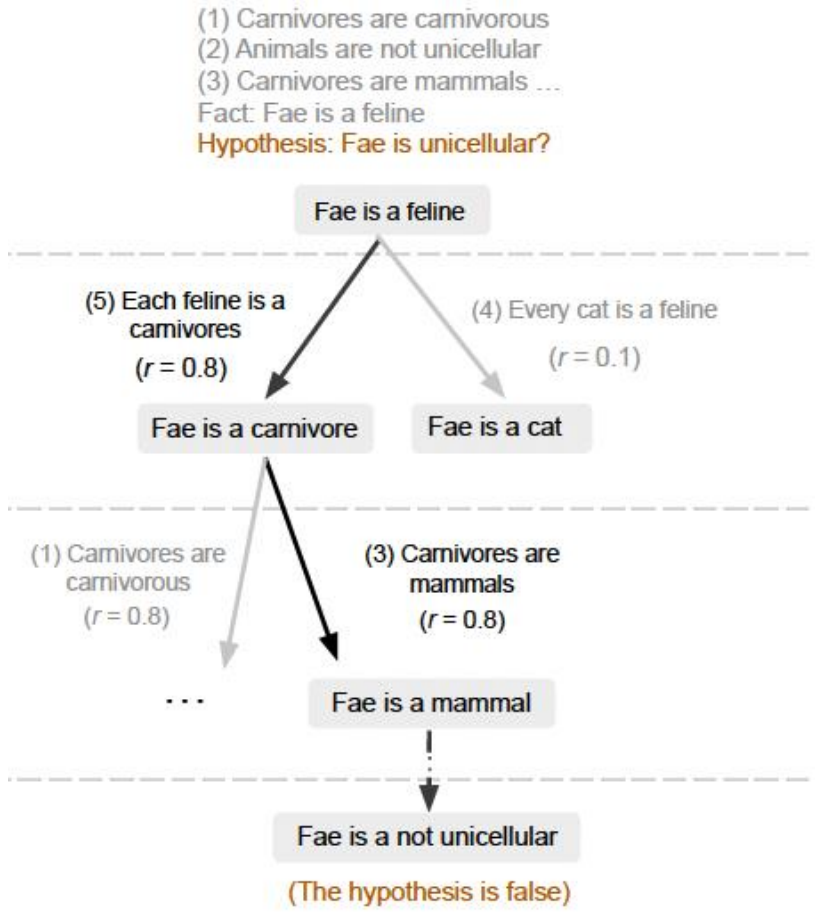
Outcome:

Supports multi-step reasoning with detailed proof generation.

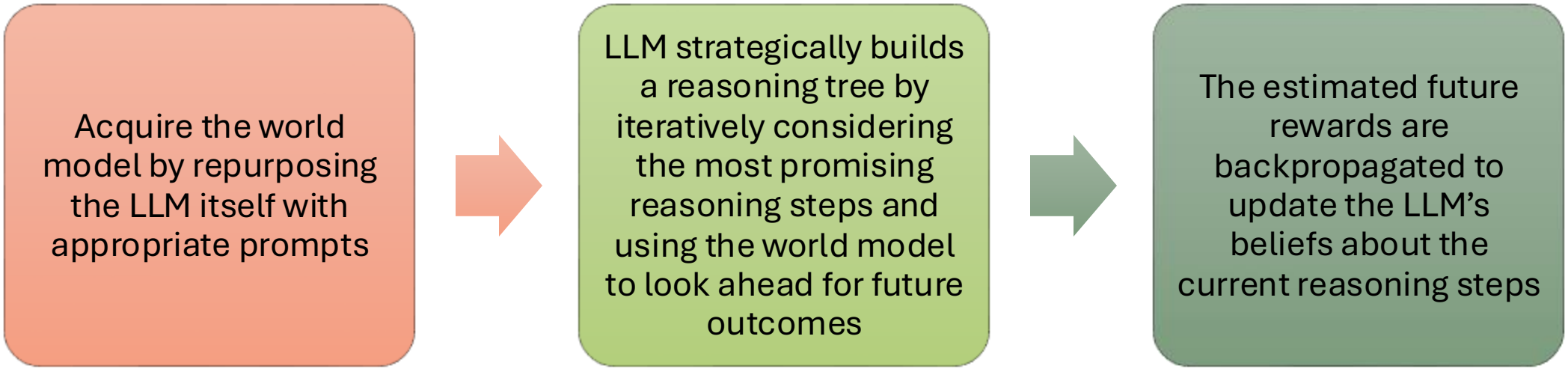
Logical Reasoning: Results

Method	Pred Acc	Proof Acc
CoT	87.8	64.8
CoT + SC	89.8	-
RAP (Ours)	**94.2**	**78.8**

Table 3: Results on ProntoQA.



RAP (Overview)



RAP enables LLMs to strategically **plan a coherent reasoning trace** for solving a wide range of reasoning tasks.

Conclusion

- RAP bridges the gap between human-like planning and LLM reasoning — setting a new benchmark for intelligent problem-solving
- Integrates **Monte Carlo Tree Search (MCTS)** to balance **exploration** and **exploitation**.
- Outperforms several CoT-based reasoning methods.
- In some cases, surpasses even **GPT-4** on challenging reasoning tasks

RL Planning and AI Planning: A Primer and Survey (Preliminary Report)

Frist Half: Swakshar Deb (swd9tc)

What is Planning?

- Planning is a sequence of actions (a_1, a_2, …, a_n) that lead us to the desired goal.

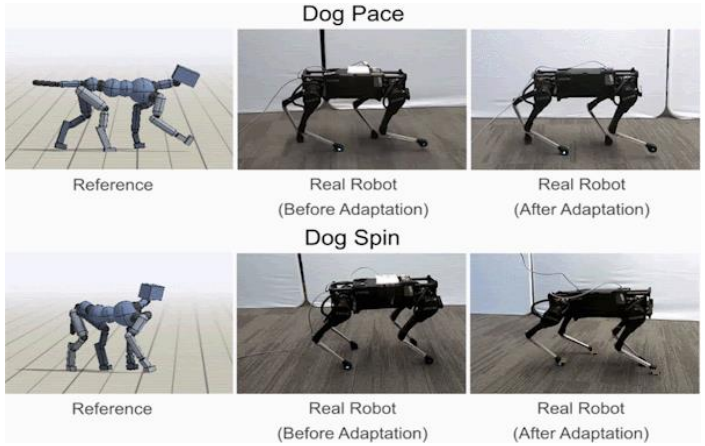
R. Epstein, C. E. Kirshnit, R. P. Lanza & L. C. Rubin, **Insight' in the pigeon: antecedents and determinants of an intelligent performance**, Nature, 1984

Why Planning is Important?

1) To handle complex task

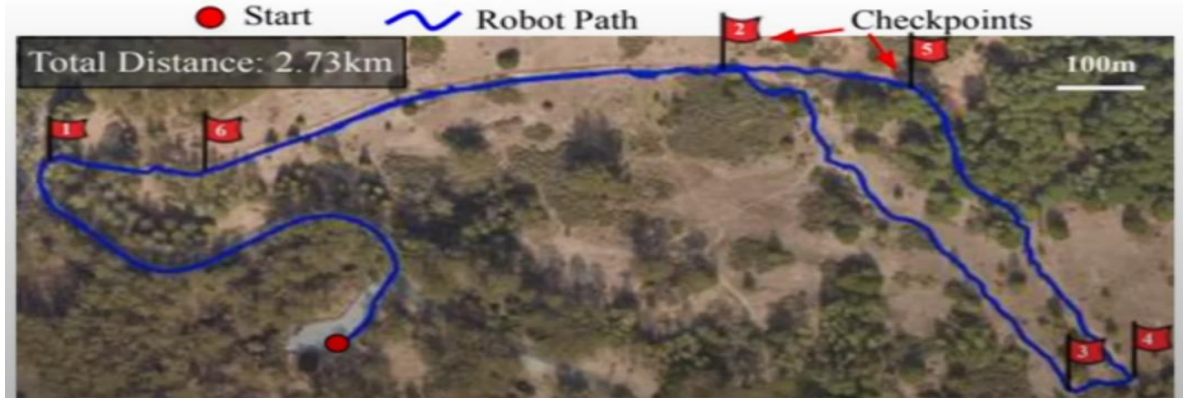
2) Can provide fine grained motor skills

Wu et al., **Discrete Policy: Learning Disentangled Action Space for Multi-Task Robotic Manipulation**, 2025



Peng et al., **Learning Agile Robotic Locomotion Skills by Imitating Animals,** 2020

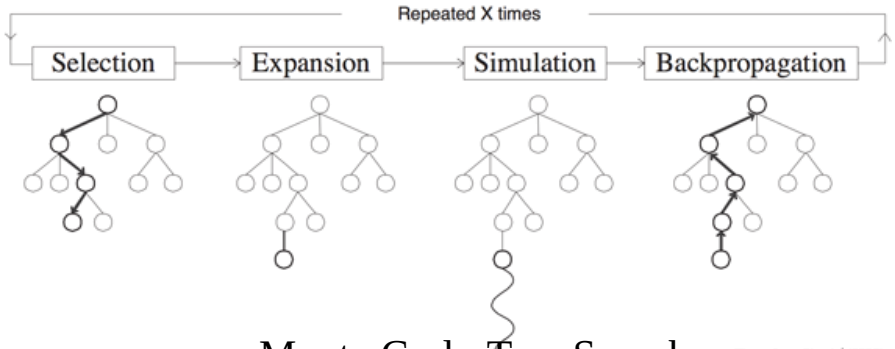
3) Planning should give the optimal path



Classical Planning vs Reinforcement Learning (RL)

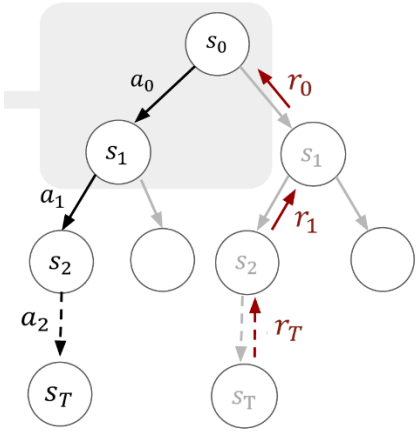
1) Classical Planning:
   tree search-based planning

   e.g., A* search, Greedy best first search, Monte Carlo search etc



Monte Carlo Tree Search

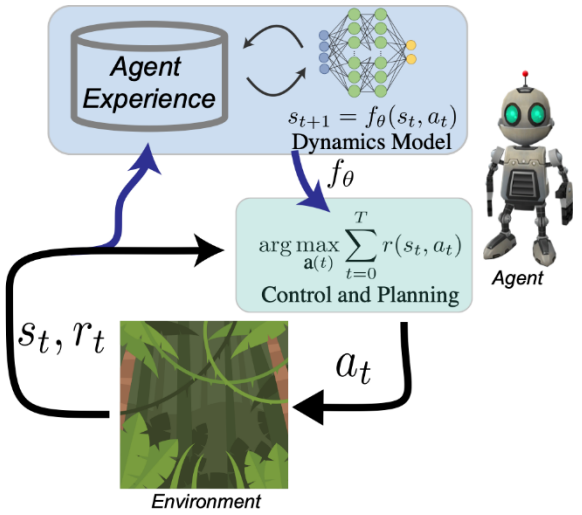
Figure from Chaslot (2006)



Reasoning via Planning (RAP)

2) Reinforcement Learning based Planning

3) AI Planner (logic based)



Planning and Representations: State

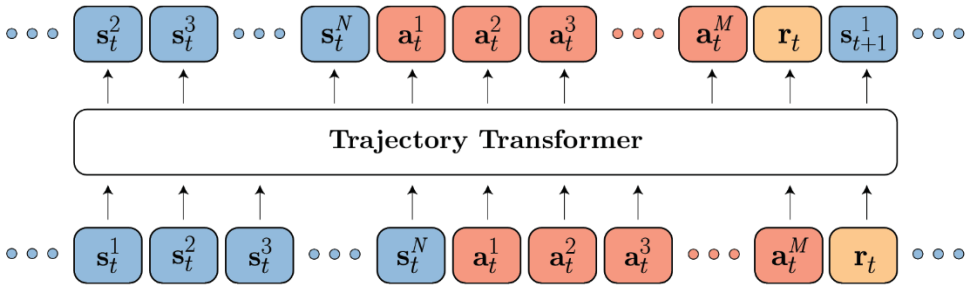
- State refers to the current situation or configuration that an agent perceives in its environment.

Shah et. al., **Rapid Exploration for Open-World Navigation with Latent Goal Models,** CoRL, 2021

(2) RL Planning: Trajectory Transformer

Trajectory transformer: fit transformer to high probability trajectory that has higher reward.

Step 1: Trajectory Modelling



initial trajectory:

$$\boldsymbol{\tau} = \left(\mathbf{s}_1, \mathbf{a}_1, r_1, \mathbf{s}_2, \mathbf{a}_2, r_2, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T\right).$$

tokenized trajectory:

$$\boldsymbol{\tau} = \left(\ldots, s_t^1, s_t^2, \ldots, s_t^N, a_t^1, a_t^2, \ldots, a_t^M, r_t, \ldots\right) \qquad t = 1, \ldots, T.$$

every dimension of every state $\mathbf{s}^i$, action $\mathbf{a}^i$, reward $\mathbf{r}_i$ is a token

Step 2: Planning

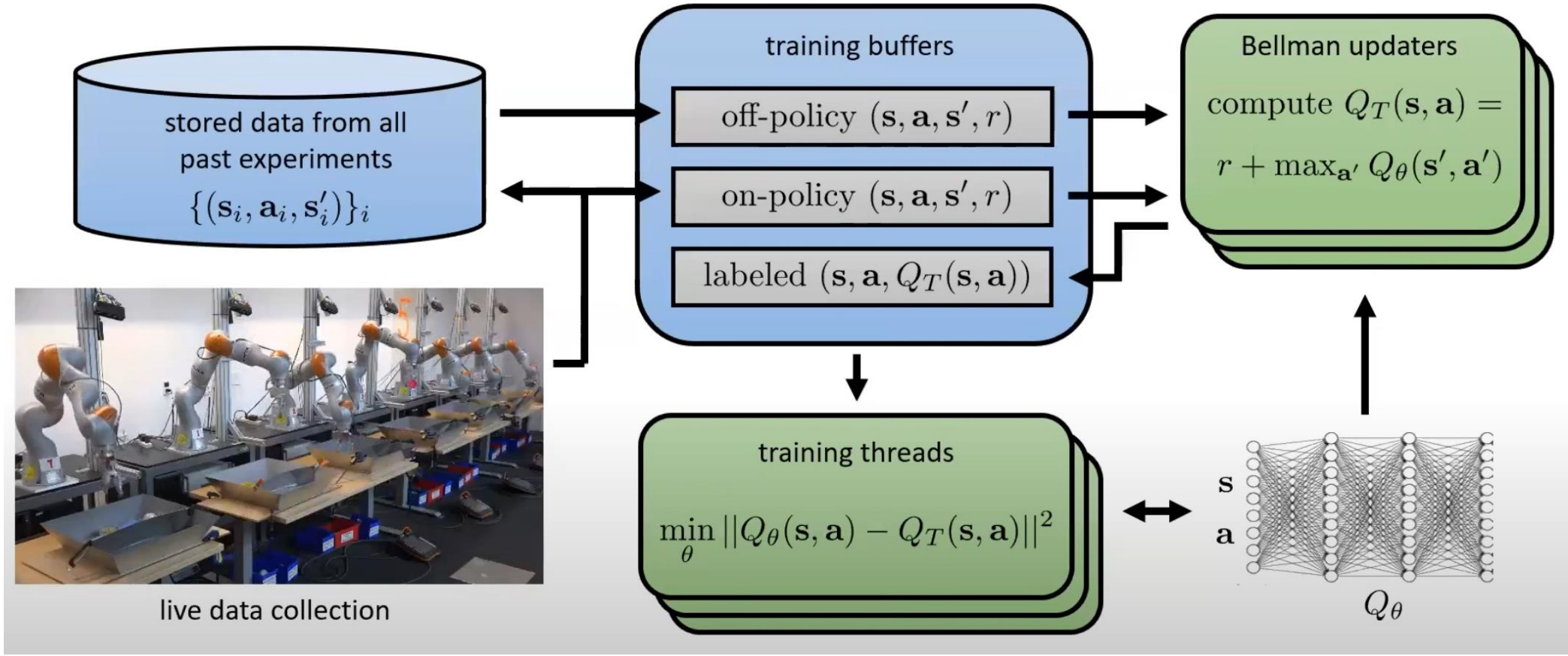
$$\mathcal{L}(\tau) = \sum_{t=1}^{T} \left(\sum_{i=1}^{N} \log P_\theta\left(s_t^i \mid \mathbf{s}_t^{<i}, \boldsymbol{\tau}_{<t}\right) + \sum_{j=1}^{M} \log P_\theta\left(a_t^j \mid \mathbf{a}_t^{<j}, \mathbf{s}_t, \boldsymbol{\tau}_{<t}\right) + \log P_\theta\left(r_t \mid \mathbf{a}_t, \mathbf{s}_t, \boldsymbol{\tau}_{<t}\right) \right),$$

Learn the high probability trajectory

Janner et al., **Offline Reinforcement Learning as One Big Sequence Modeling Problem**, Neurips, 2021

Demonstration in the Simulated Environment

RL Planning: QT-OPT



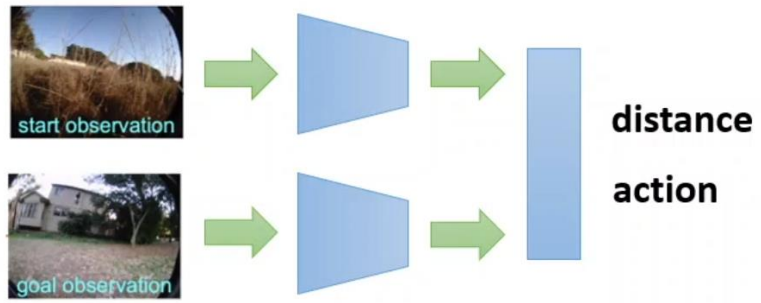
Kalashnikov et al., **QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation**, 2018

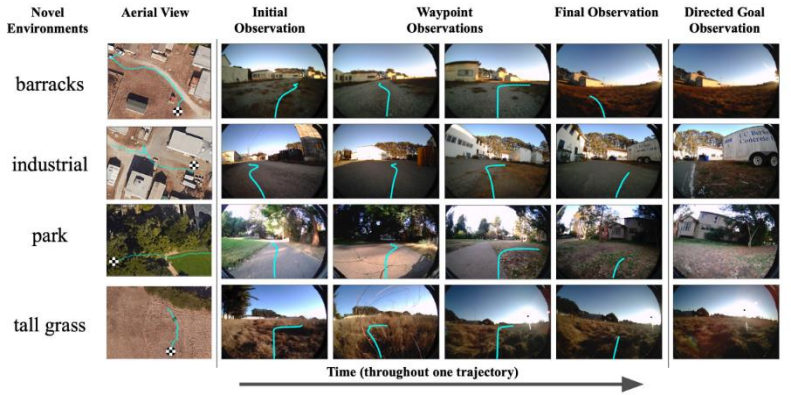
This slide is taken from https://www.youtube.com/watch?v=5Tdndu275AY&t=1202s

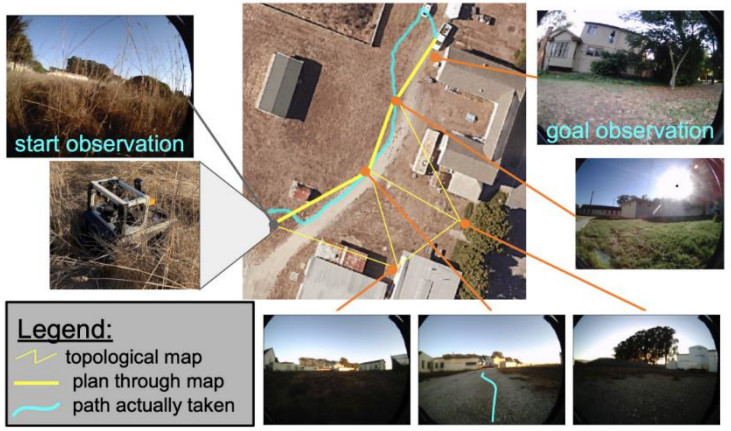
Does it works?

Method	Dataset	Test
QT-Opt (ours)	580k off-policy + 28k on-policy	**96%**
Levine et al. [27]	900k grasps from Levine et al. [27]	78%
QT-Opt (ours)	580k off-policy grasps only	87%
Levine et al. [27]	400k grasps from our dataset	67%

ViNG: Learning Open-World Navigation with Visual Goals





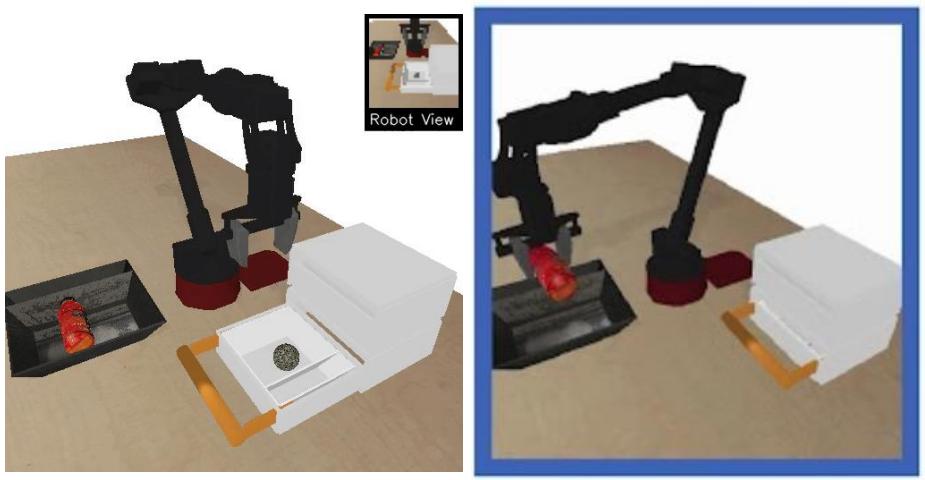


Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, Sergey Levine, **ViNG: Learning Open-World Navigation with Visual Goals**, Neurips, 2021

Real World demonstration of ViNG

COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning

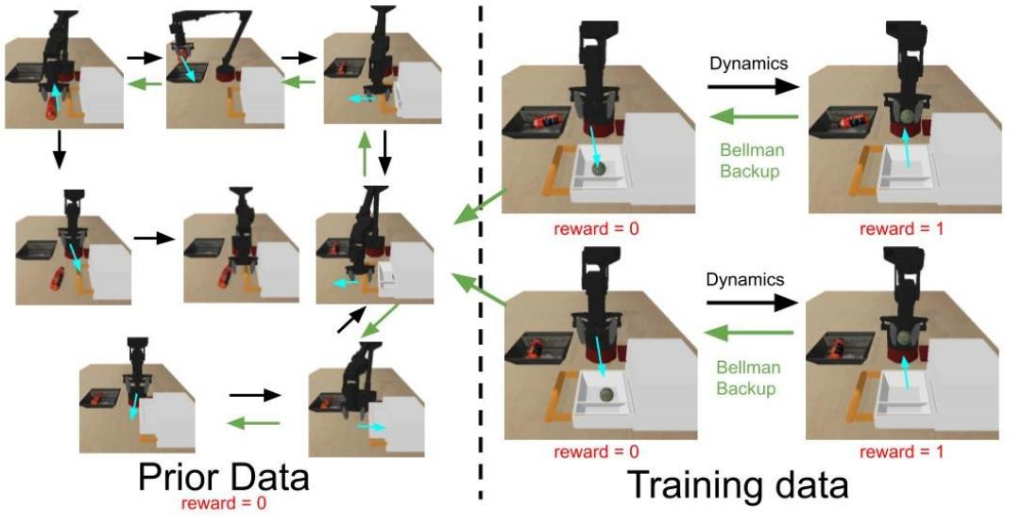
RL policies typically do no generalize well to initial condition that is not seen during training



Training time

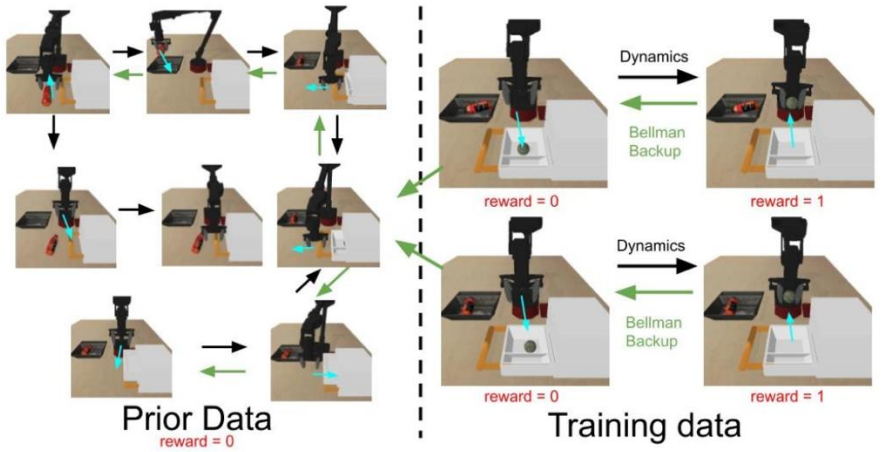
New initial condition

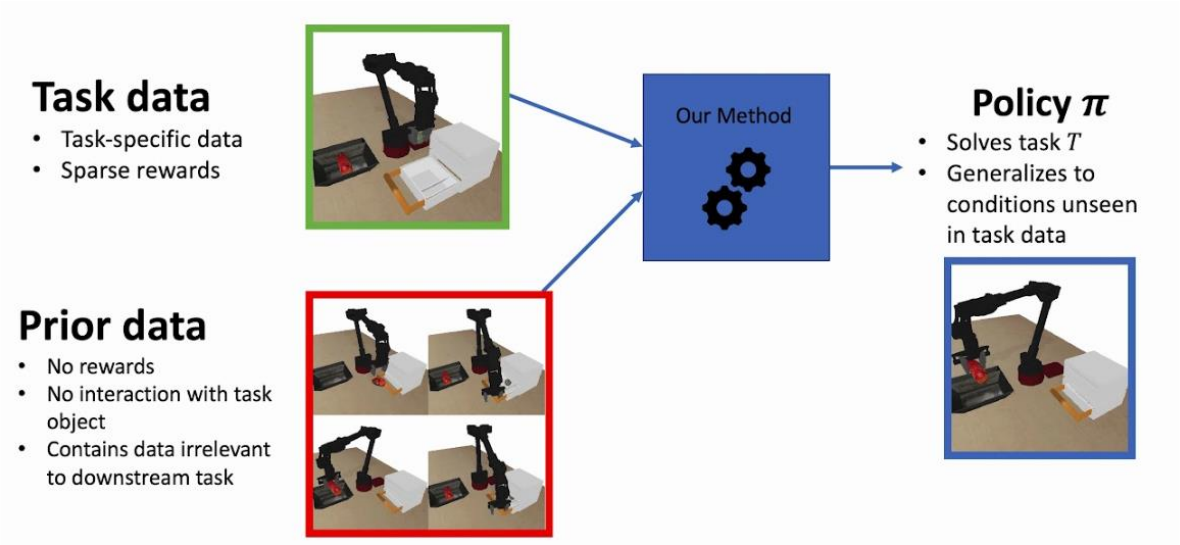
Can we use previously collected, unlabeled data to extend the learned skill?



Singh et al., **COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning**, CoRL, 2020

Problem Setup

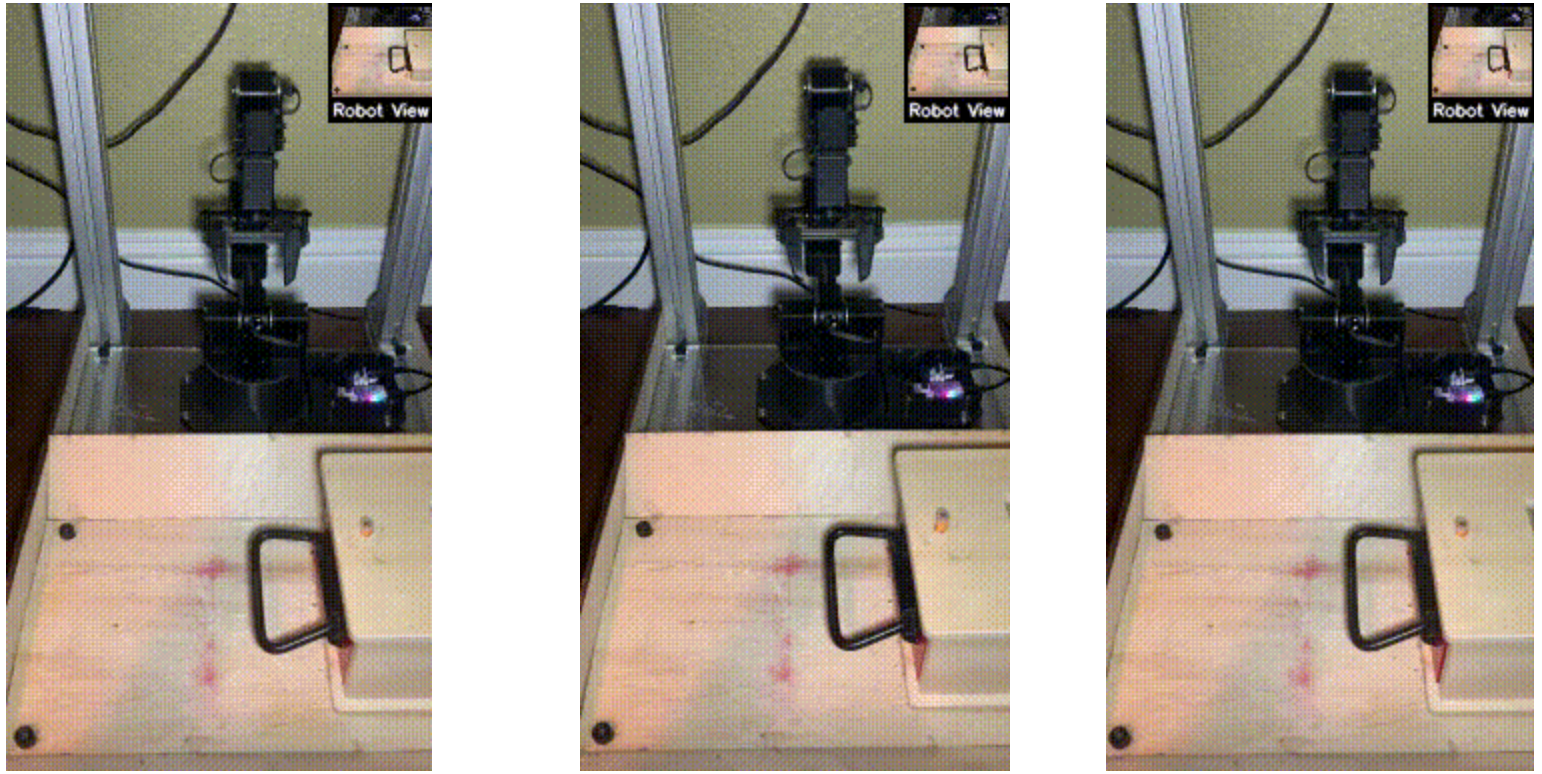




- Reward = 0
- Propagate the learned Q values to prior data

- Sparse 0, 1 reward
- Learn Q function in the training dataset

Real World Evaluation of CoG



Real World Evaluation

(3) AI Planning, logic based

Example of logic based state representation: used for multiple objects

$$\{\texttt{in(dog, bedroom)}, \texttt{hungry(dog)}, \texttt{brown(dog)}, \texttt{dirty(bedroom)}, \texttt{likes(dog, bedroom)}, \texttt{can\_make(dog, sandwich)}, \texttt{in(bed, bedroom)}\}.$$

AP Representations: State

- **Object:** Constant symbols representing entities (e.g., dog, bedroom).
- **Predicate:** A logical statement that represent relationships or properties between objects.
- **Grounding:** Assigning objects to predicate arguments to form propositions.

***in*(dog, bedroom)** → Means the dog is in the bedroom

- A **state** is a set of propositions representing all predicates.

State = **{**in(dog, bedroom), hungry(dog), brown(dog), dirty(bedroom), likes(dog, bedroom), can_make(dog, sandwich), in(bed, bedroom)**}**

AP Representations: Action Schema

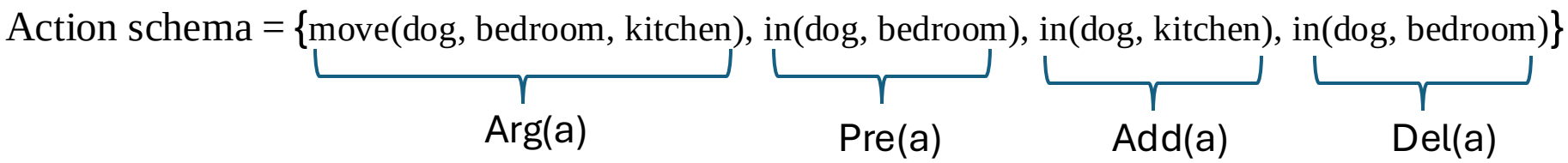
- A planning action is represented as a **tuple**:

$$\text{action schema} = \langle \text{arg}(a), \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$$

- Components:
  - **arg(a):** Arguments for the action.
  - **pre(a):** Preconditions that must be true for execution.
  - **add(a):** Positive effects (predicates that become true after executing the action).
  - **del(a):** Negative effects (predicates that become false after executing the action).

Continue...

- Syntax: action schema = ⟨arg(a), pre(a), add(a), del(a)⟩
- Example:



Benefits of Structure: Attention

- AP represents states as **databases of facts**, enabling **queries for transitions** and goal determination
- Actions **only affect relevant facts**, leaving irrelevant details unchanged.
- Example:

State = **{**in(dog, bedroom), hungry(dog), brown(dog), dirty(bedroom), likes(dog, bedroom), can_make(dog, sandwich), in(bed, bedroom)**}**

Action: Move(dog, bedroom, kitchen)

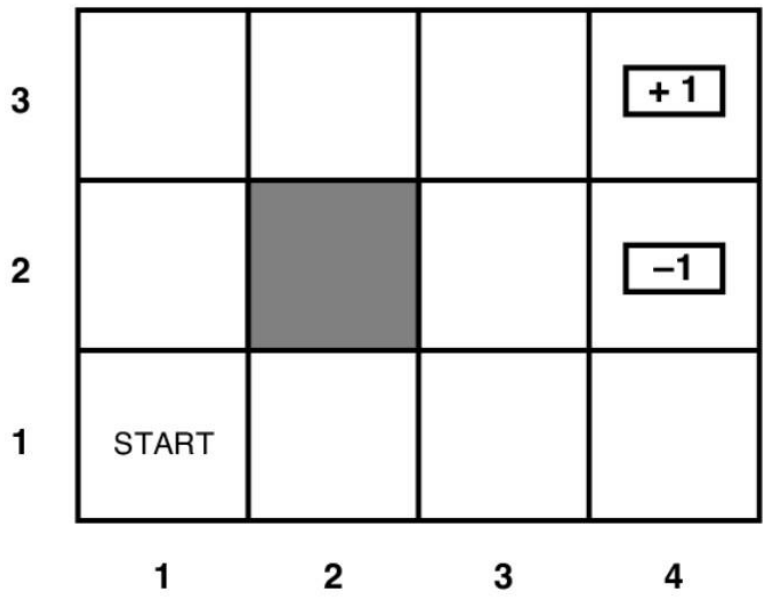
New state ={**in(dog, kitchen)**,hungry(dog),brown(dog),dirty(bedroom),likes(dog, bedroom),can_make(dog, sandwich),in(bed, bedroom)}

Benefits of Structure: Automation

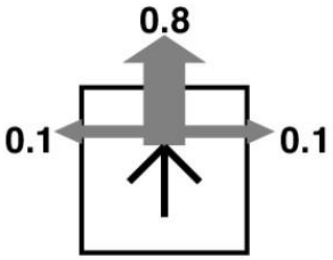
- AP can be viewed as a declarative programming paradigm where problems are specified as a program in a planning language
- Do not have to write a solver for a problem and only a model of the problem.
- Examples: A* search, Greedy Best First Search, Monte Carlo Tree Search.

Planning Extensions: Uncertainty

- **Probabilistic planning:** Allowing for probabilistic transitions.
- **Nondeterministic planning:** Probabilities for transitions are not given or unknown.
- **Conformant planning:** Computing plans in partially observable environments



Transition model:



R(s) = -0.04 for every non-terminal state

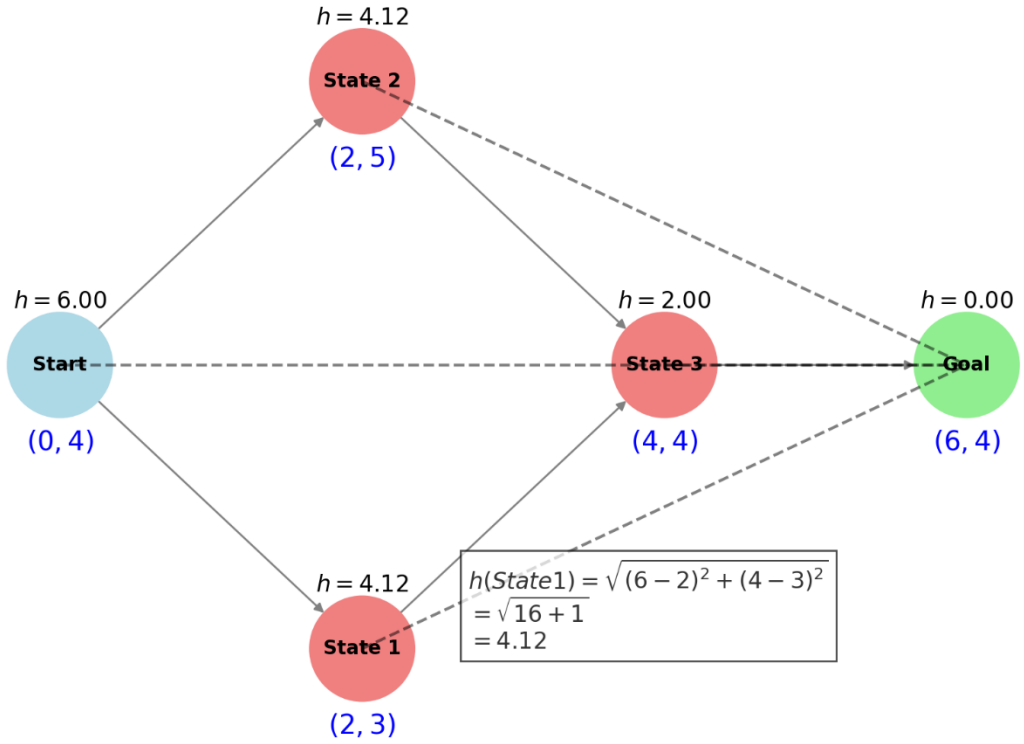
Planning Extensions: Time and Numerics

- **Numeric Planning** introduces functions and numeric expressions, allowing the representation of numeric state variables capturing e.g. resources, physical properties, and plan metrics.

- **Temporal Planning** extends the defnition of actions and plans to allow for durative and concurrent actions, as well as deadlines and temporal synchronisation.

Solving Planning Task: Heuristic Functions

- Functions that estimate the cost from a given state to the goal.
- Helps guide search algorithms like A* and Greedy Best-First Search.
- **Examples:** Manhattan Distance, Euclidean distance, Number of misplaced tiles (used in puzzles)

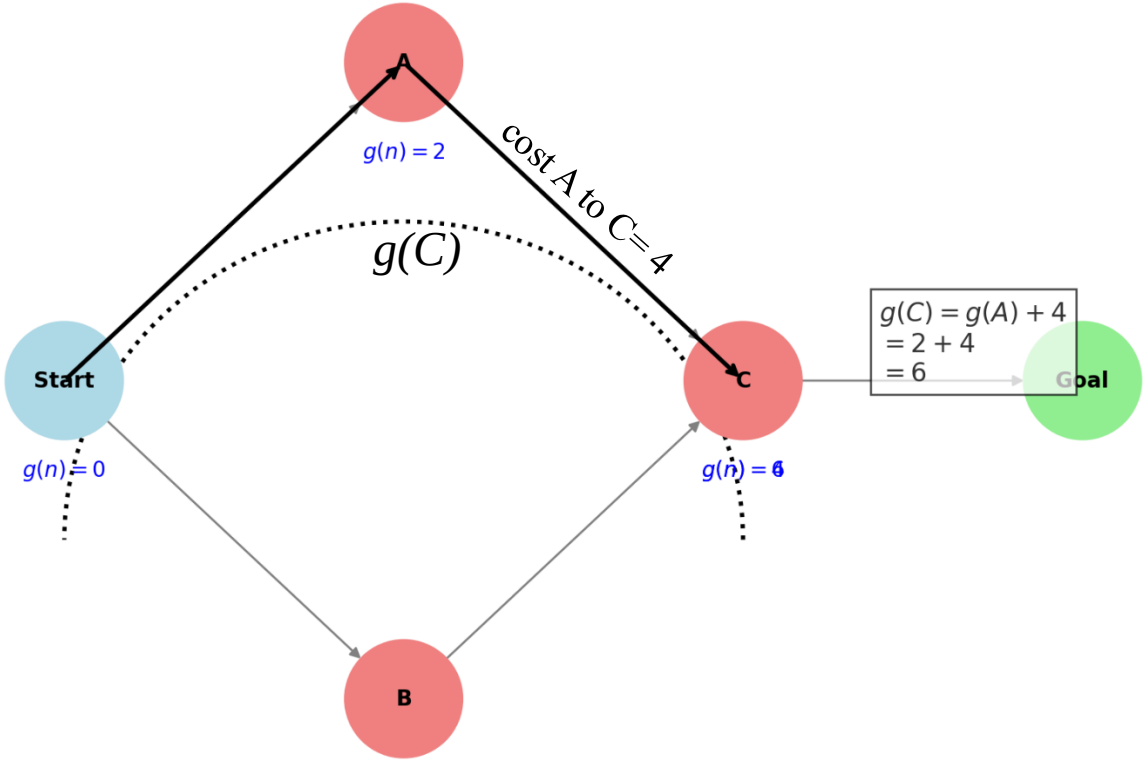


Solving Planning Task: A* Algorithm

- Select the node with lowest

$$f(n) = g(n) + h(n)$$

Where, *g(n)* is **cost** of node n from the start and *h(n)* is the **heuristic value** of node n



Problem Decomposition: Subgoals

- **Subgoals as Decomposition**: They break down complex tasks into intermediate goals
- **Landmarking in Planning:** Represent essential actions or state required for the solution

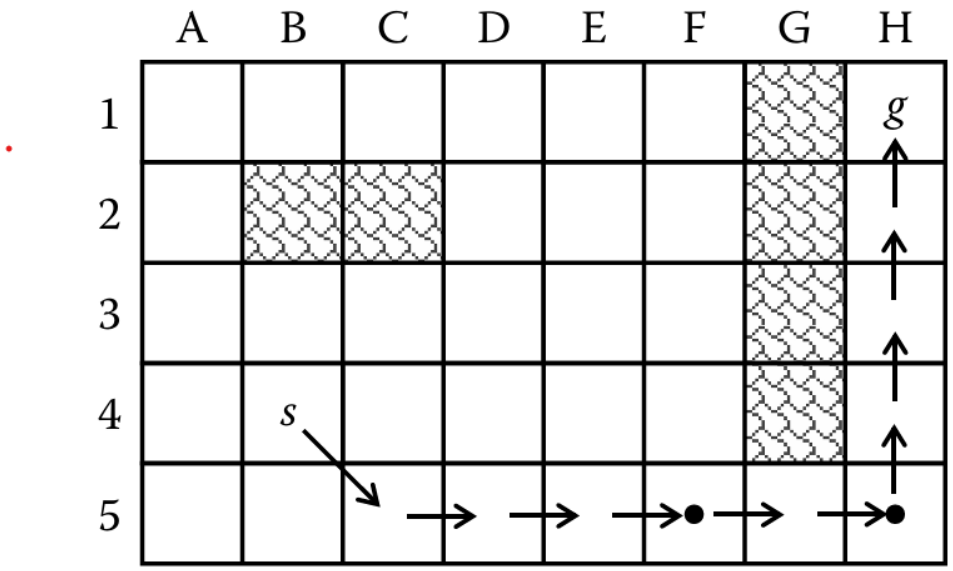
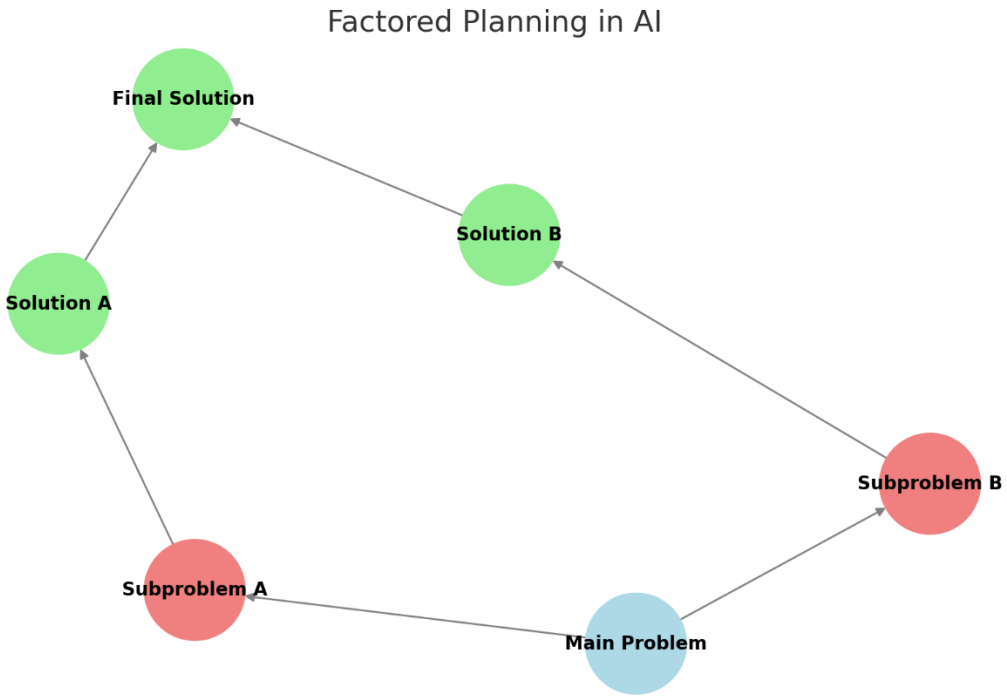


Fig: '*s*' is the start state, '*g*' is the goal state and • is the subgoals

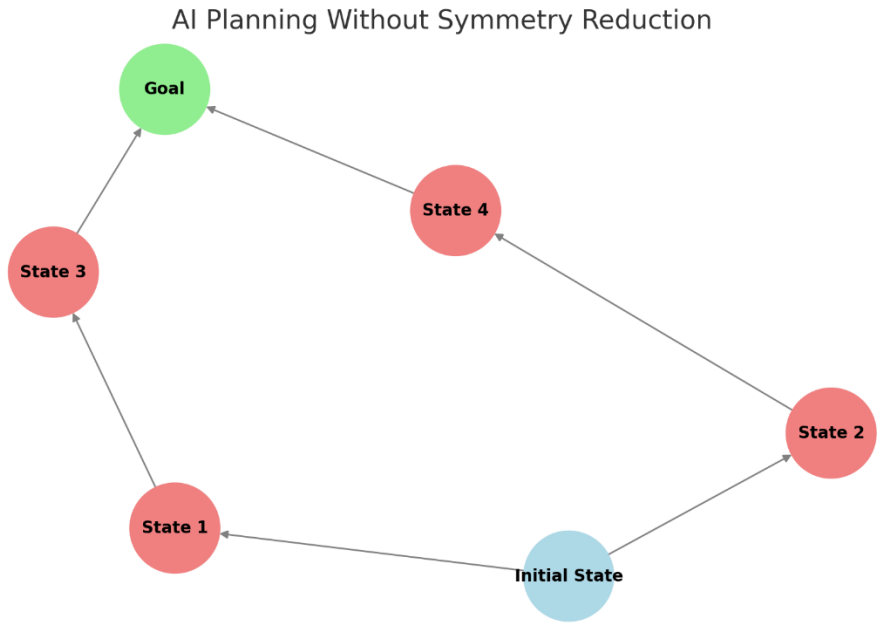
Problem Decomposition: Factored Planning

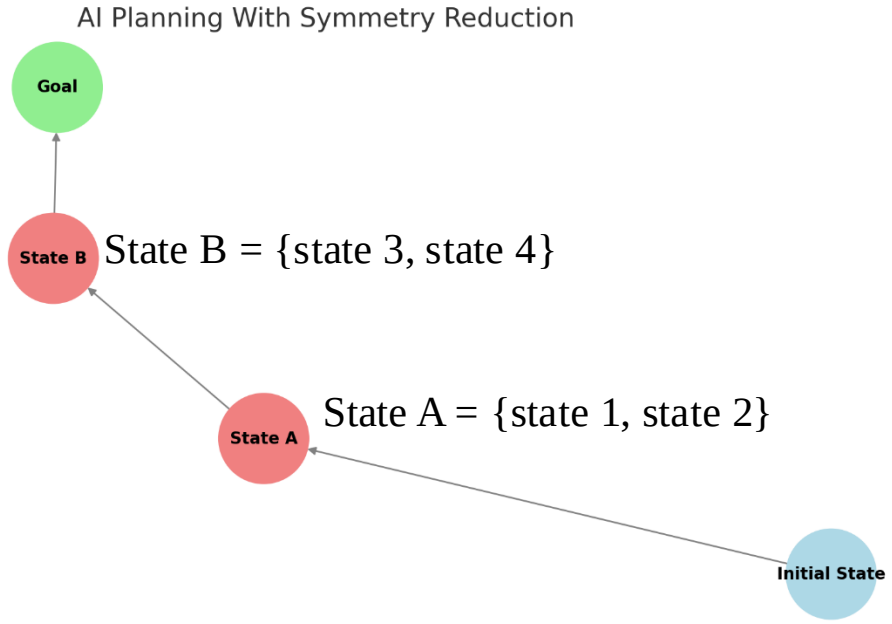
- Decomposes large planning problems into smaller, independent subproblems.
- Each subproblem is solved separately and then combined for the final solution.
- Allows parallel solving of independent factors by multiple robots.



Problem Decomposition: Symmetries

- A form of problem decomposition that involves collapsing equivalent subproblems to generate an easier task to solve
- Improves efficiency by avoiding duplicate exploration of symmetrical paths.





Second Half of the Paper AI Planning: A Primer and Survey(Preliminary Report)

Presented By : Md. Mahir Ashhab (ftm2nu)

Generalization in Planning

What is Generalization?

- The ability of AI planners to **solve unseen problems** by leveraging prior knowledge.
- planning approaches define **task-specific rules** that transfer across domains.
- Different from **RL,** where generalization is often tied to reward learning,

Formal Definition of Generalization in Planning

Problem Tuple: ⟨D, T_train, T_test⟩
D: A domain
T_train: Training tasks
T_test: Unseen test tasks
Learning involves constructing **Generalized Planning Knowledge (GPK)** for efficiently solving T_test.

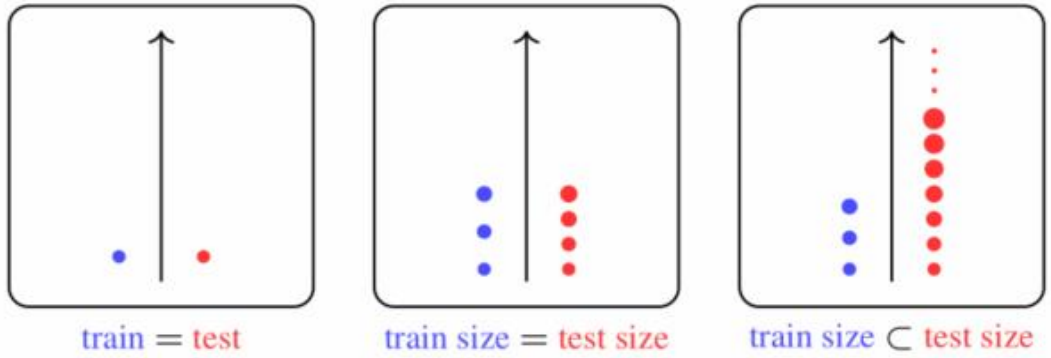


Figure 1: Generalisation setups for decision-making. AP approaches incorporating learning often handle the most general case (right) involving arbitrarily large test problems.

Learning Structure and from Structure

- Modern approaches integrate **learning methods** to **automatically infer structure**.
- Different from **classical AI planning**, which relies on explicitly defined **state-transition models**,
- Two key perspectives:
  A. **Learning structured representations from unstructured data** (e.g., learning symbolic planning models from raw state-action traces).
  B. **Leveraging structured planning knowledge to enhance learning efficiency** (e.g., using heuristic functions to guide learning).
- **Key Question:**
  - *How can AI planning methods be extended to handle complex, real-world tasks where models are unknown or changing?*

A: Learning Planning Models - STRUCTURE

Goal: Convert raw **state-action sequences** into structured planning models (**PDDL, RDDL**).

PDDL :

$$\{\texttt{in(dog, bedroom)}, \texttt{hungry(dog)}, \texttt{brown(dog)},$$
$$\texttt{dirty(bedroom)}, \texttt{likes(dog, bedroom)},$$
$$\texttt{can\_make(dog, sandwich)}, \texttt{in(bed, bedroom)}\}.$$

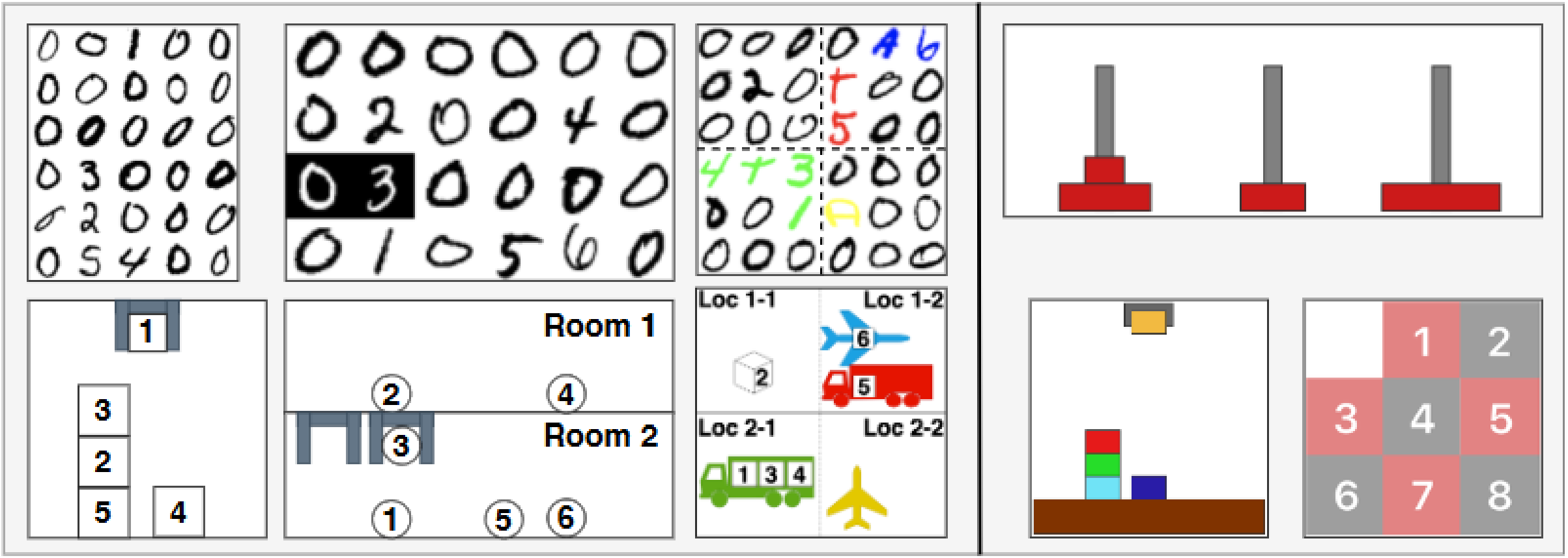
Types of Model Learning Approaches

1. **From Passive state-action traces**
   - Extracting action models from state transitions
   - Example: **LOCM2, ARMS** for deterministic planning

2. **Learning from Passive Images/Videos traces**
   - Convert **raw observations** into planning models
   - Example: Using **GNNs and Transformer-based encoders**

3. **Active Model Learning (Exploratory Learning)**
   - Incrementally refining models based on agent interactions
   - Example: **Incremental Learning Model (ILM)**

Neuro-Symbolic Learning of Lifted Action Models from Visual Traces



Neuro-Symbolic Learning of Lifted Action Models from Visual Traces

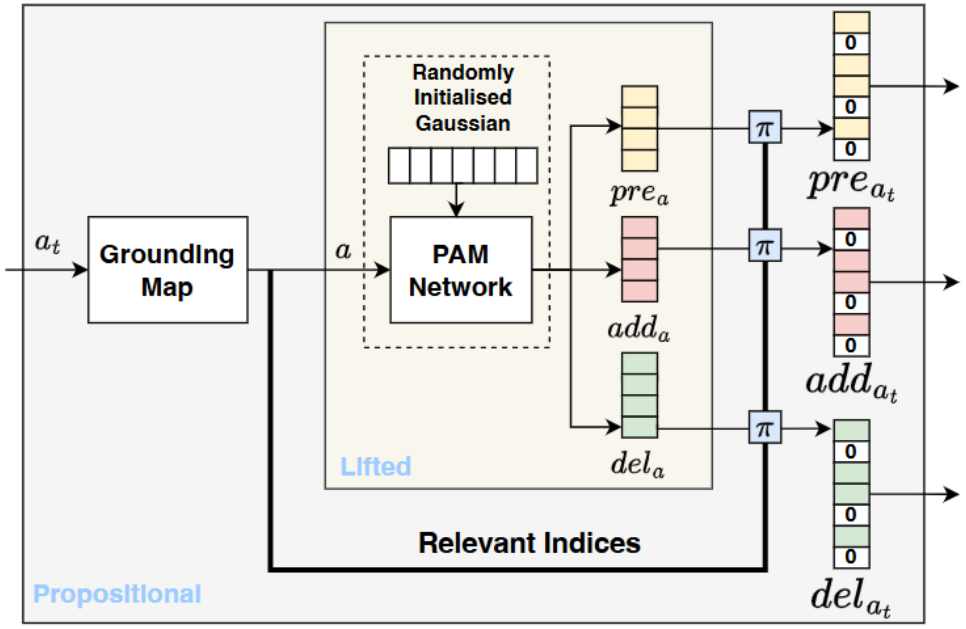
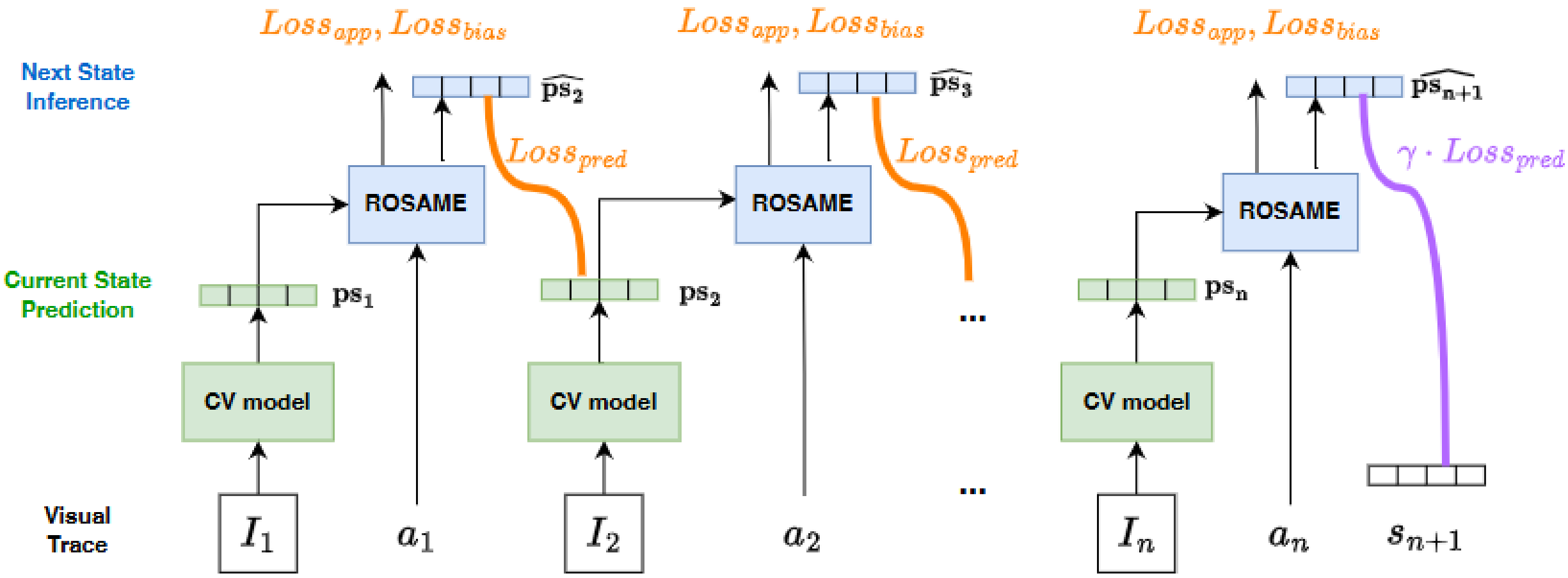


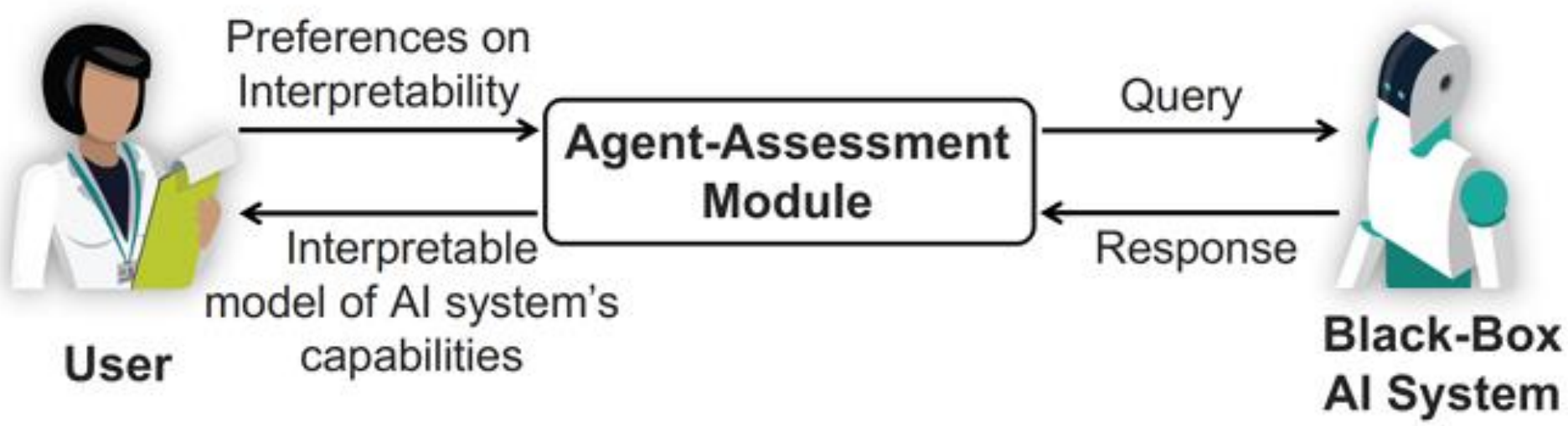
Figure 3: ROSAME architecture. The projection operation π maps the output of the PAM network to relevant indices in vectors of length $|P_I|$. Indices not mapped take value zero.

Definition 1 *A* Probabilistic Action Model *(PAM) is defined as a tuple of three functions* $\langle pre, add, del \rangle$*, where for an action schema* $a(\vec{x})$ *and a predicate* $p(\vec{y})$ *relevant to* $a(\vec{x})$*,* $pre(a(\vec{x}), p(\vec{y}))$*,* $add(a(\vec{x}), p(\vec{y}))$*, and* $del(a(\vec{x}), p(\vec{y}))$ *are probabilities of* $p(\vec{y})$ *being a precondition, an add effect, or a delete effect of* $a(\vec{x})$*.*

Neuro-Symbolic Learning of Lifted Action Models from Visual Traces



Asking the Right Questions: Learning Interpretable Action Models Through Query Answering



Asking the Right Questions: Learning Interpretable Action Models Through Query Answering

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, \mathbb{A}_H, \mathbb{P}^*, \mathbb{S}$
Output: poss_models
Initialize poss_models $= \{\phi\}$
for γ in some input pal ordering Γ **do**
    new_models $\leftarrow$ poss_models
    pruned_models$= \{\}$
    for each $\mathcal{M}'$ in new_models **do**
        for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
            $\mathcal{Q}, \mathcal{M}_i, \mathcal{M}_j \leftarrow$ generate_query$(\mathcal{M}', i, j, \gamma, \mathbb{S})$
            $\mathcal{M}_{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$
            pruned_models$\leftarrow$ pruned_models $\cup\, \mathcal{M}_{prune}$
        end for
    end for
    if pruned_models is $\emptyset$ **then**
        update_pal_ordering$(\Gamma, \mathbb{S})$
        continue
    end if
    poss_models $\leftarrow$ new_models $\times \{\gamma^+, \gamma^-, \gamma^{\emptyset}\} \setminus$ pruned_models
end for

B: Learning for Planning (L4P)

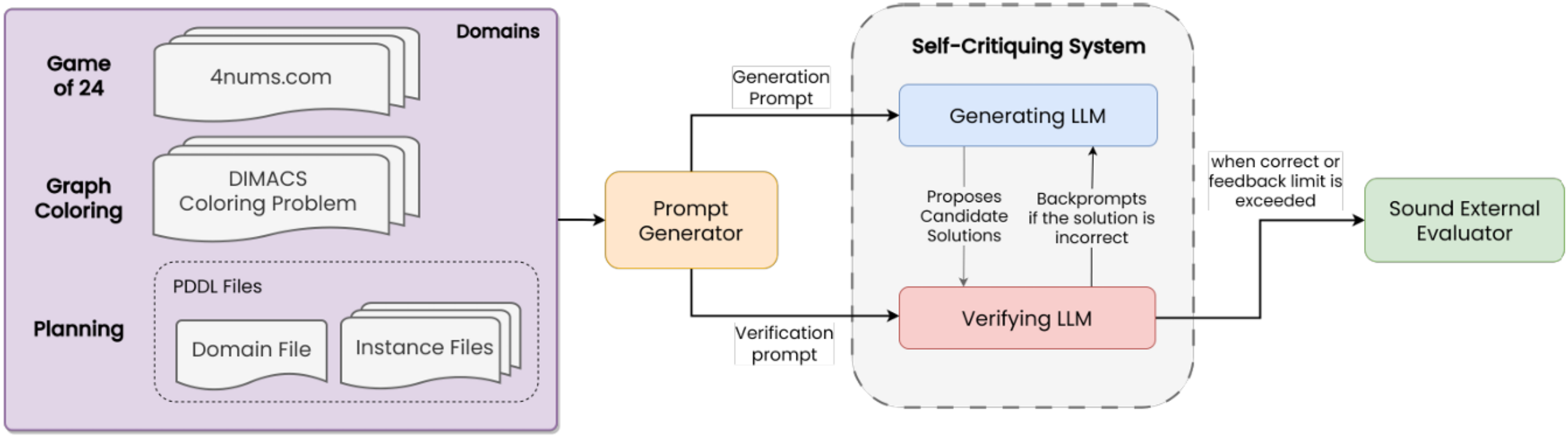
L4P focuses on **enhancing AI planners** using machine learning techniques.

Key Techniques:

1. **Graph Learning for Heuristic Search**
   - Using **Graph Neural Networks (GNNs)** to learn heuristic functions.
   - Example: **Toyer et al. (2018) – ASNets**

2. **Automatically Generating Reward Functions**
   - Reward function **learning via planning abstractions**.
   - Example: **RL approaches integrating symbolic knowledge**.

Low cost Reasoning: On the Self-Verification Limitations of Large Language Models on Reasoning and Planning Tasks



Automatic reward functions and labels: ASNets

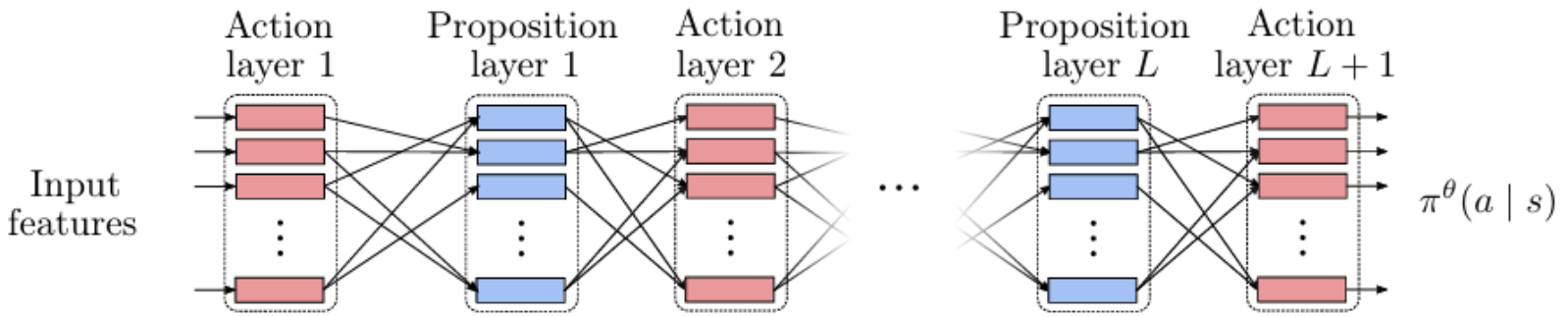
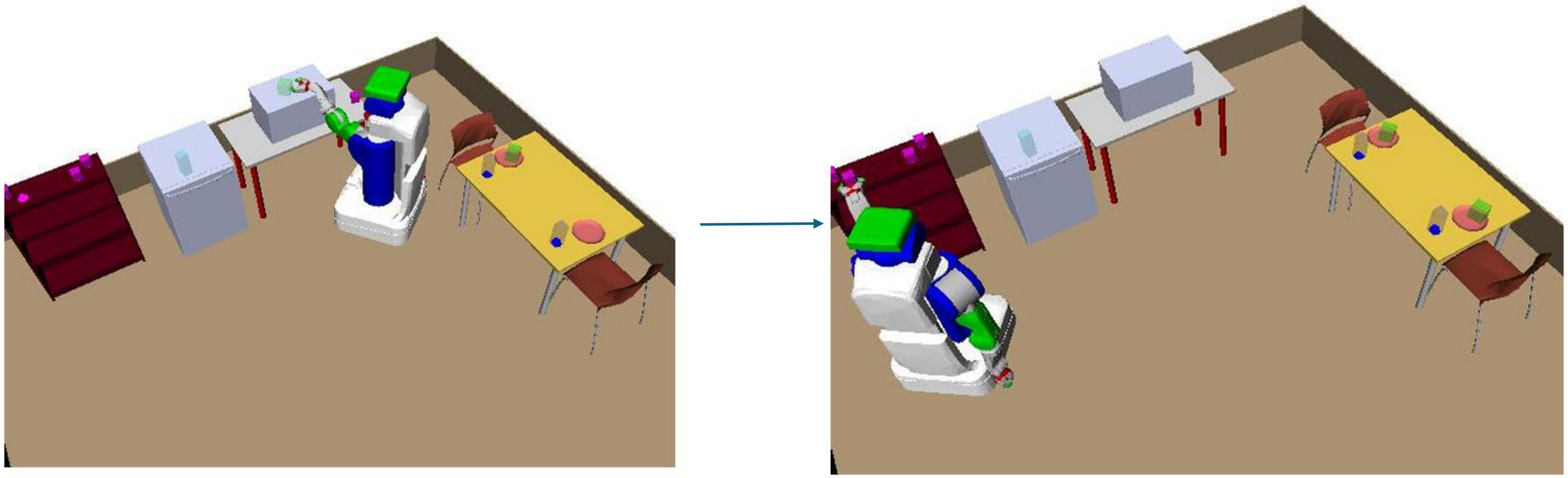


Figure 2: High-level overview of an ASNet. Each coloured rectangle represents an action module (red) or a proposition module (blue); these modules apply learnt transformations to input feature vectors in order to produce more expressive output feature vectors. Information flows from the input (left) to the output (right) along the black lines connecting modules in successive layers. For the sake of visual clarity, skip connections (described in the main text) are not depicted. Modules are grouped into L proposition layers and $L+1$ action layers. Throughout, we refer to such a network as an "L-layer ASNet".

Automatically Generate supervised Training Labels from Training Tasks with Domain Independent Planners



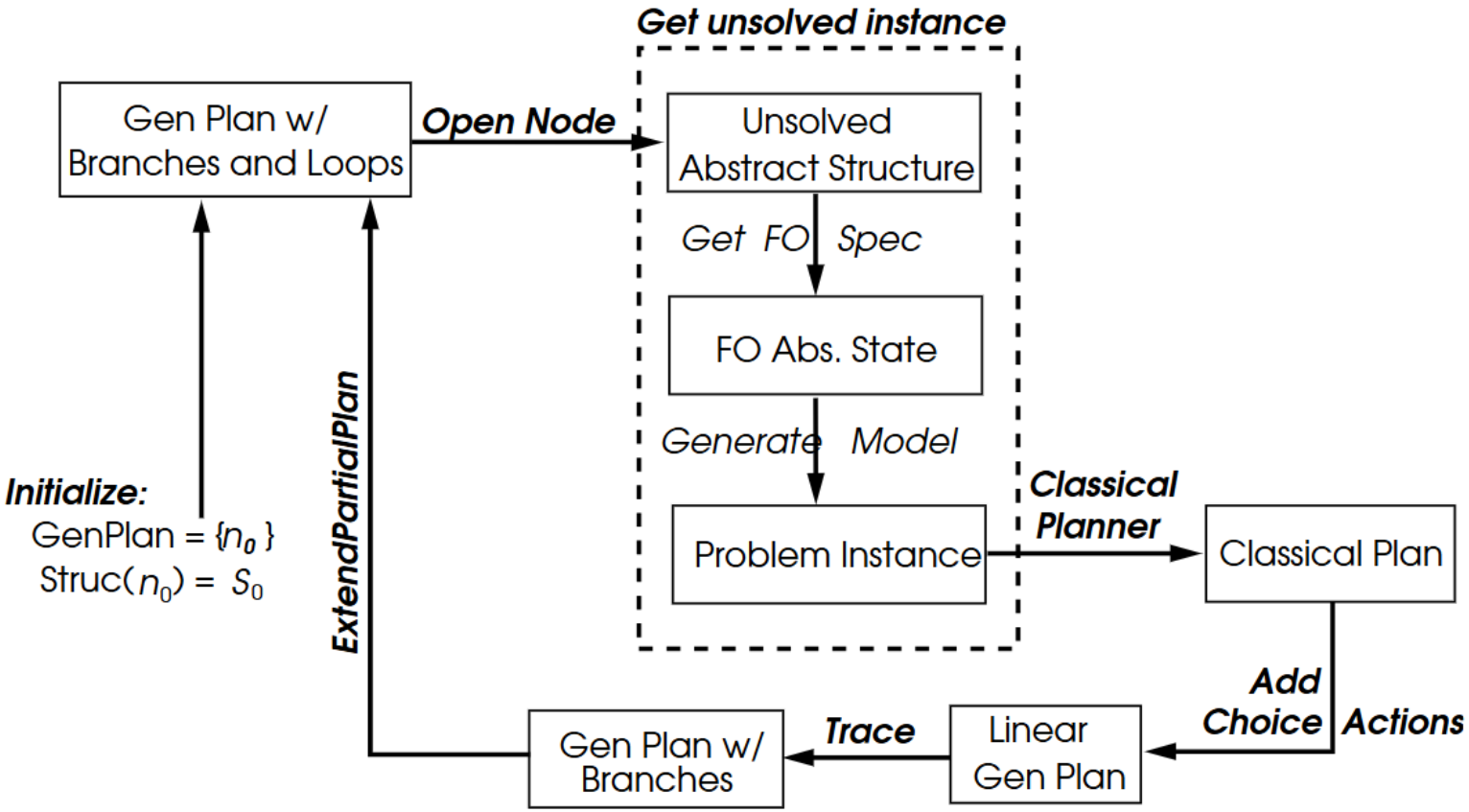
Generalized Planning (GP)

Objective
• Learn **reusable policies** for multiple planning tasks.

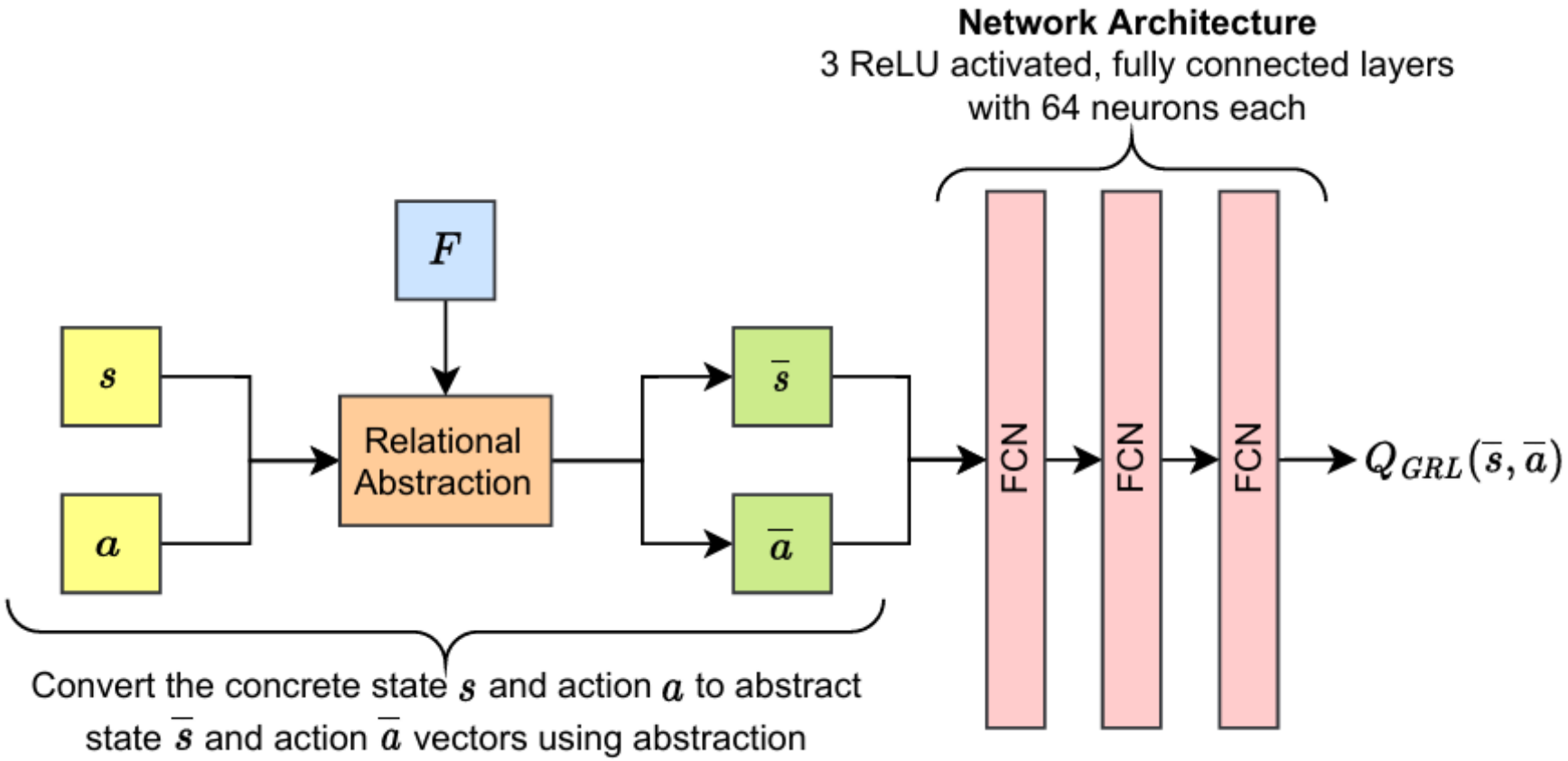
Challenges:
• Scalability to **large state spaces**
• **Verifying correctness** of learned policies

Solution Approaches:
1. **Program Synthesis-Based GP**
2. **Graph-Based GP**

Directed Search for Generalized Plans Using Classical Planners



Generalized Q-Function in RL problems



Key Takeaways from the Survey

- AI Planning is evolving to **incorporate learning-based techniques**.
- Structure **reduces complexity** in decision-making.
- **Bridging AI Planning and RL** is crucial for next-gen AI applications.

Challenges and Future Directions

Scalability

- How to handle **large, real-world dynamic environments**?

Generalization

- How to **efficiently transfer** learned models?

Integration with Generative AI

- How can **foundation models** help automate planning knowledge?

Final Thoughts and Open Questions

- AI Planning is **bridging traditional symbolic reasoning and deep learning**.
- Learning-based planning is **reshaping automation, robotics, and real-world AI**.

Open Discussion Question:

- *How can we create AI planners that learn and adapt like humans?*

Thank you