Introduction to Agentic AI

-- AI Agent Planning Design

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

- Chain-of-Thought and ReAct
- Post-training (Fine-tuning) Reasoning
  - DeekSeek R1
- In-context Reasoning Basics
  - Reflexion

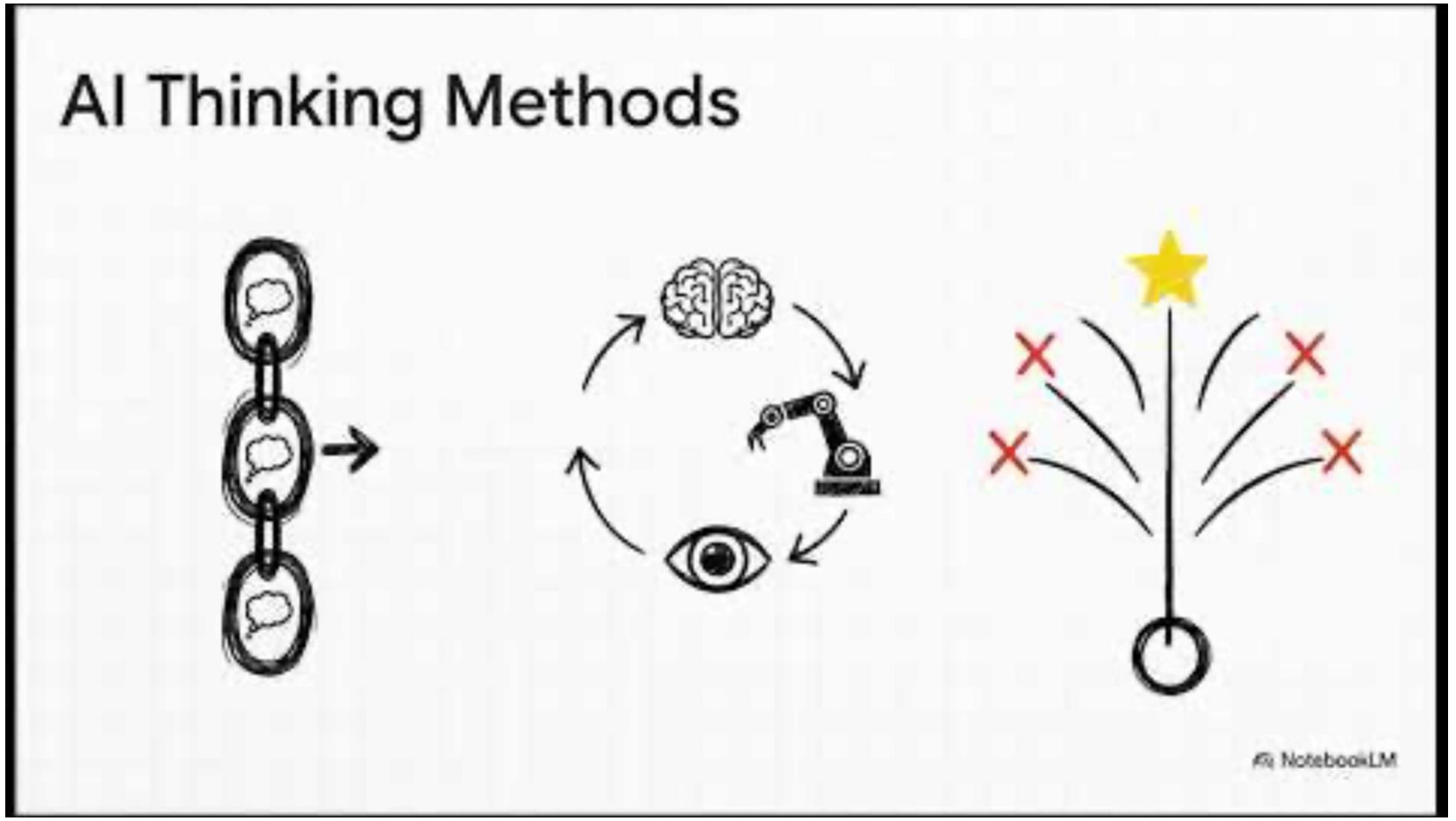
This Lecture

- Search for Planning
  - Monte Carlo Tree Search (MCTS)
- Workflow Design
- Process Formulation
  - Planning Domain Definition Language (PDDL)

AI Agent Planning

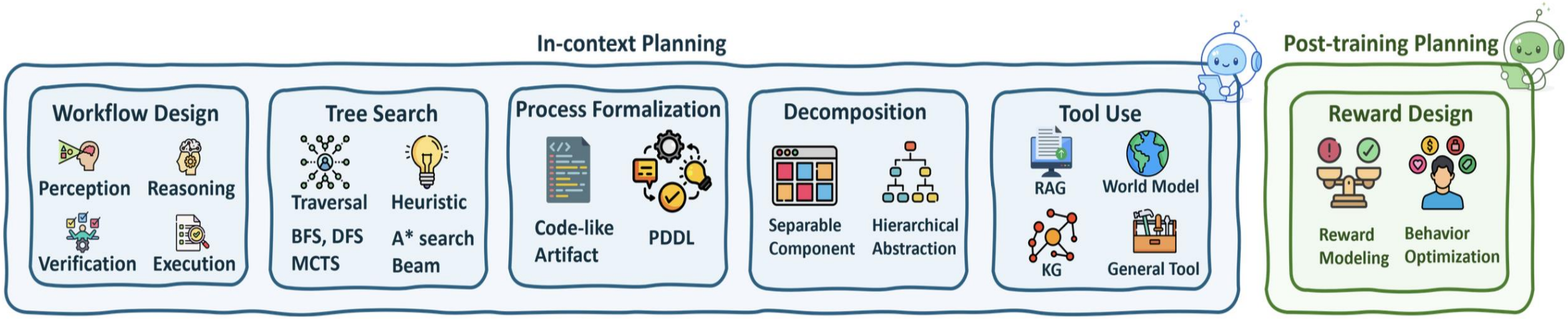
- A sequence of actions to achieve **a specific goal**
  - Decision-making, goal prioritization and action sequencing.
  - The goal serves as the guiding principle for the agent's decision-making process, determining the end state it seeks to achieve.
  - If goal is complex, agentic AI models can break it down into smaller, more manageable sub-goals (**task decomposition**).
- Accurate state representation
  - Structured understanding of the environment.
  - Modeling the current conditions, constraints and contextual factors that influence decision-making.
- Action sequencing
  - Structuring a logical and efficient set of steps that the agent must follow.

From Goal Decomposition to AI-Driven Execution



https://www.youtube.com/watch?v=_e6xJZMWqcs

Reasoning and Planning in Agentic AI



- **Reasoning** is the analytical process of understanding information
  - analyzing data, identifying patterns, and using logic to determine necessary actions (**output insights, justifications or recommendations**)
- **Planning** is the process of breaking a goal into sequential, actionable steps
  - taking the insights from reasoning and mapping out a sequence of actions, tasks, or subtasks to achieve a goal (**output a sequence of action**)

Agentic Reasoning for Large Language Models https://arxiv.org/abs/2601.12538

This Lecture

- Search for Planning
  - Monte Carlo Tree Search (MCTS)
- Workflow Design
- Process Formulation
  - Planning Domain Definition Language (PDDL)

Simulation Representation for Planning

Basic Components of a Simulation

- S = set of all states
  - propositions that are true: you are in a house, door is open, knife in drawer
- A = set of all actions
  - take knife from drawer, walk through door
- T = transition matrix T: (S, A)→S
  - (you are in a house & door is open, walk through door) → you are outside
  - There are pre-conditions that need to be met to perform a certain action, and post-conditions that are true after

You have a simulation, now what?

- You need to plan out a policy
  - Sequence of actions to get from start state to goal state
- A **plan** gives you a way of getting said policy
  - Can be expressed as constraints on actions you can perform
  - Search is a way of getting possible plans for a given specific goal
    - Monte-Carlo Tree Search (MCTS) interleave evaluation and policy improvement to solve complex planning problems by sampling future, unknown states, often used in games like Chess and Go.

Search Terminology

- **State Space Search** - each state is a node on the search tree, go from there
- **Planning Space Search** – searching through space of possible plans or constraints on actions
- **Satisficing –** looking longer and longer for a good enough solution
- **Optimal** – looking for the best possible solution there is
- Some Assumptions (but not necessary):
  - No environment stochasticity, exact post conditions always manifest once action is executed
  - No agent stochasticity, actions are always executed as planned

Forward Search

- Some deterministic implementations of forward search:
  - breadth-first search
  - depth-first search
  - best-first search (e.g., A*)
  - greedy search

- Breadth-first and best-first search are sound and complete. But they require too much memory
  - Sound: a plan generated by the search traces will guarantee a solution if executed. (a guarantee of correctness)
  - Complete: if a solution exists, then at least one of the search's traces will be a solution.

Forward Search Issues and Backward Search

Forward Search Issues

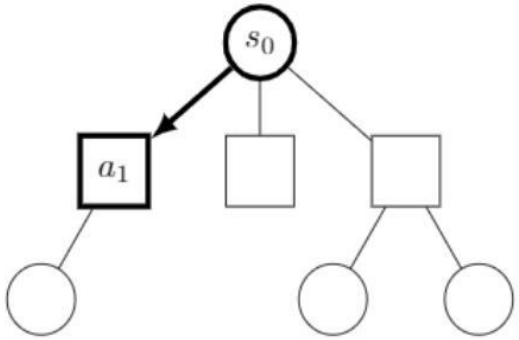
- Branching factor - lots of possible states and actions, deterministic searches waste time trying a bunch of unnecessary stuff.
- State and action spaces can blow up memory and compute costs.

Backward Search:

- we start at the goal and compute inverse state transitions
  - new set of subgoals = $T^{-1}(g, a)$
  - An action *a* is relevant for a goal *g*
- Generating predecessor states (inverting transition matrix) is hard

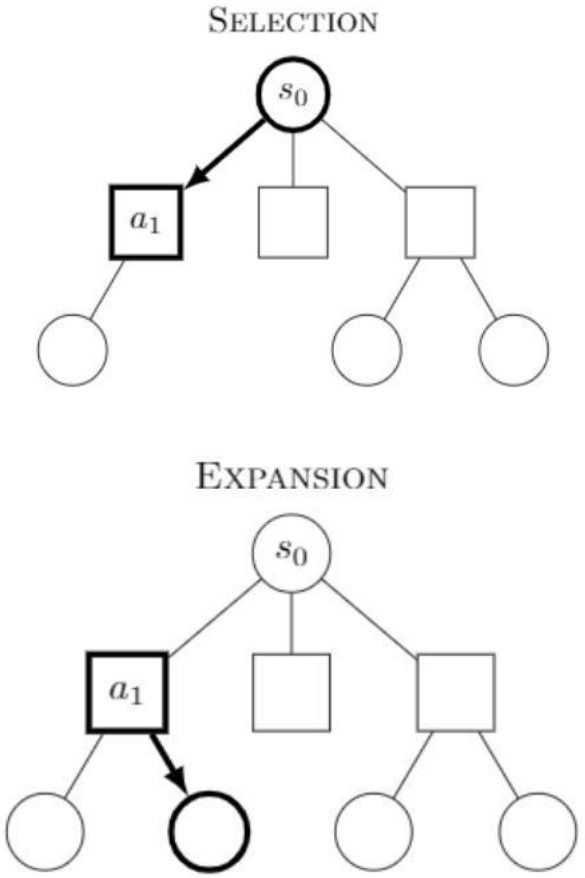
Monte Carlo Methods

- A set of methods that focus on learning better from simulated experiences collected by interacting with an environment
- When to use?
  - You have a way of easily simulating an environment, but it is too complex to solve deterministically with planning/search.
- For arbitrary problem with start state s0 and actions a_1
- All states have attributes:
  - Total simulation reward Q(s) and
  - Total no. of visits N(s)



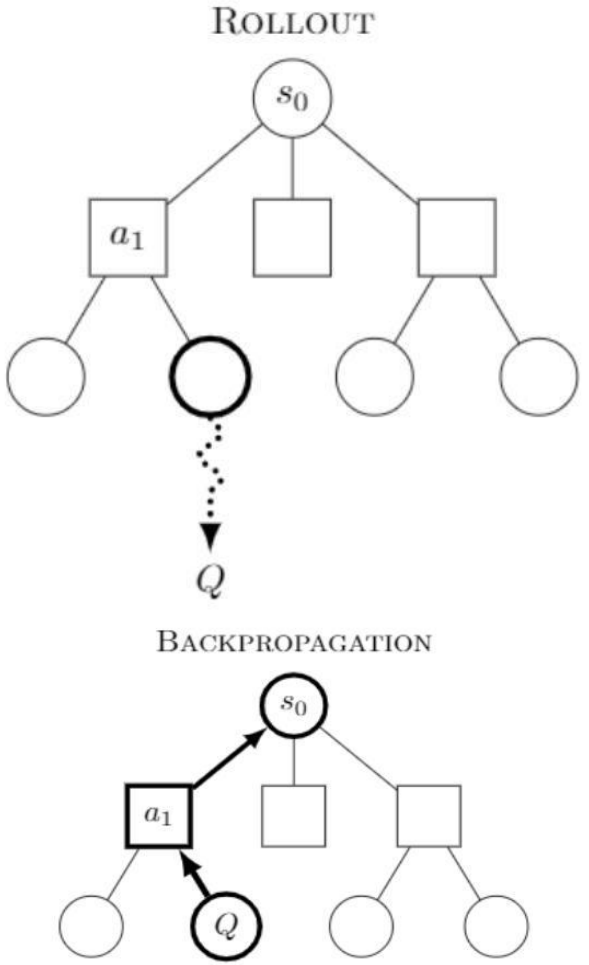
MCTS Part 1 – Selection and Expansion

- Selection
  - From the current state, pick an action to perform
  - For now, assume we pick randomly
  - Update N(s) as you pick a new state
- Expansion
  - Execute transition
  - If resultant state is a terminal state, observe result (reward)

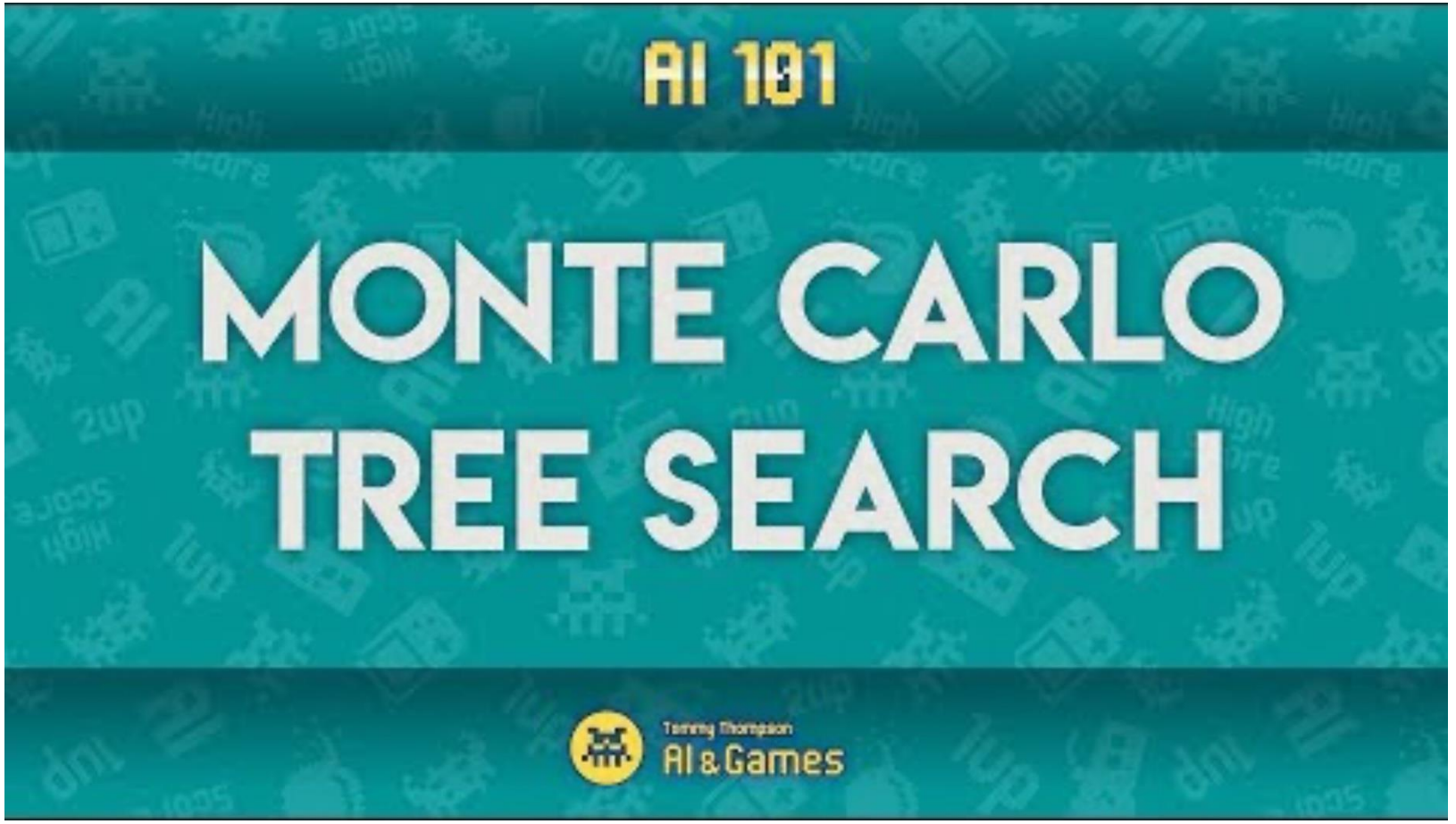


MCTS Part 2 - Rollout and Backpropagation

- Rollout
  - If it is not a terminal state, finish a playout until it is
  - For now, we will “cheat” and directly use our simulation for this
- Backpropagation
  - Add the reward of the simulated path to all node scores, this gives you Q(s) = Total_reward/visit_count



Monte Carlo Tree Search



https://www.youtube.com/watch?v=lhFXKNyA0QA

This Lecture

- Search for Planning
  - Monte Carlo Tree Search (MCTS)
- Workflow Design
- Process Formulation
  - Planning Domain Definition Language (PDDL)

Workflow Design

- Structuring the overall planning process into distinct stages such as perception, reasoning, execution, verification
  - Designing pipelines that decompose task solving into subtasks, often leveraging a plan-and-act framework.
  - Structured prompting to sequentialize tasks and guide reasoning progression.

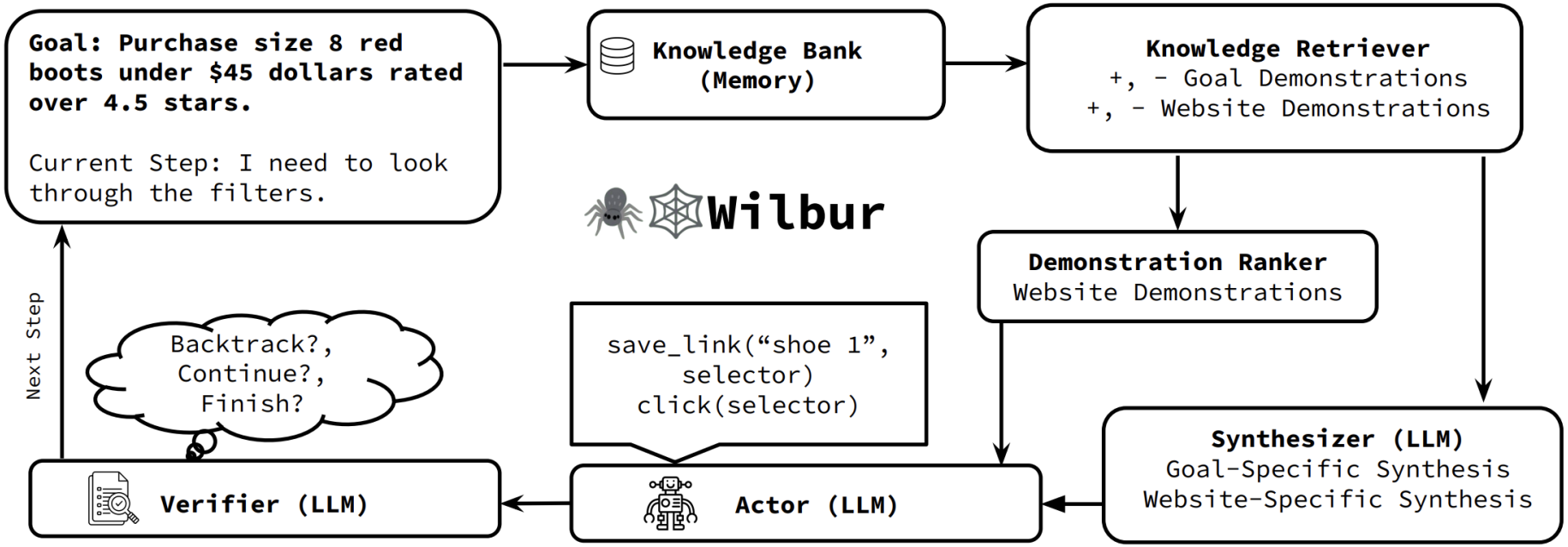
- The workflow provides interpretable structure and interfaces (**what is done when**), while the reactive loop supplies closed-loop grounding and error recovery (**how it is done in context**).

Example: Web Agent

- Web agent: interact with the web through a browser.
  - Encoding the Document Object Model (DOM)
  - Analyzing a screenshot of the page in a multimodal fashion
- Performance still NOT good enough…
  - The reasoning limitations of the underlying LLM
  - The need to learn how specific websites works: faced with a never-seen-before website, one needs to explore, try different approaches, and adjust.
  - More than a billion websites in the world, LLM can NOT memorize all of them just from pretraining. Zero-shot hard to work!

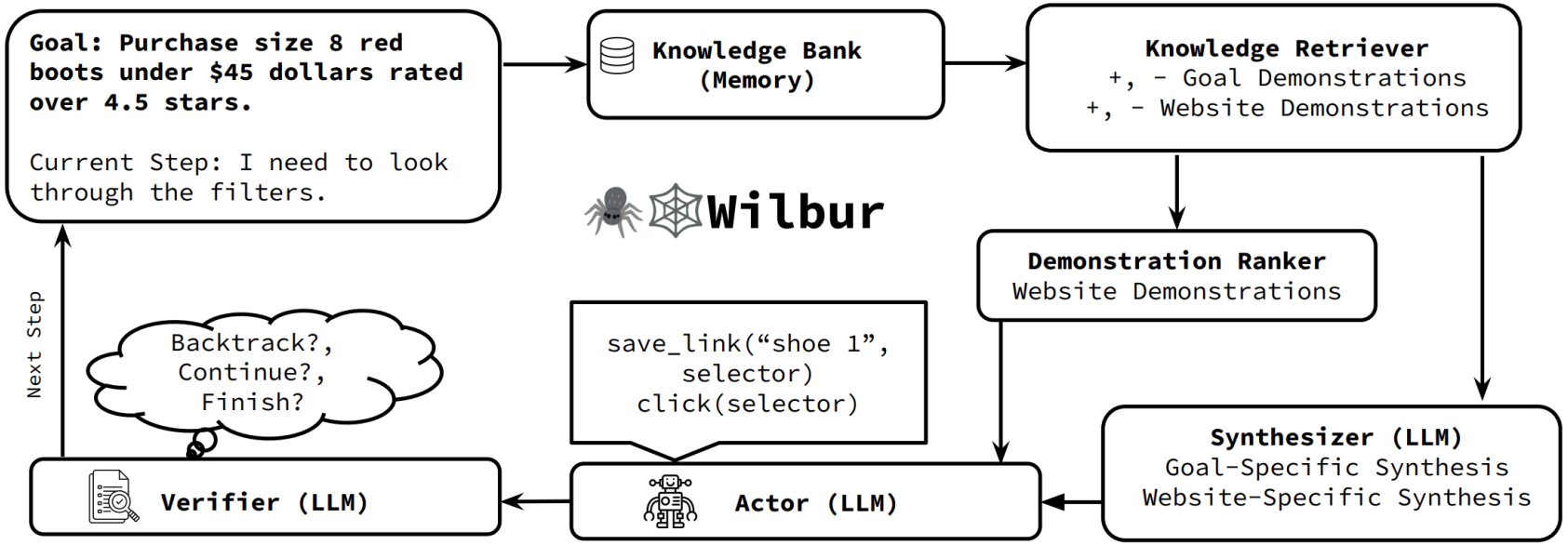
WILBUR: Adaptive In-Context Learning for Robust and Accurate Web Agents https://arxiv.org/pdf/2404.05902

Example: Web Agent (Cont'd)



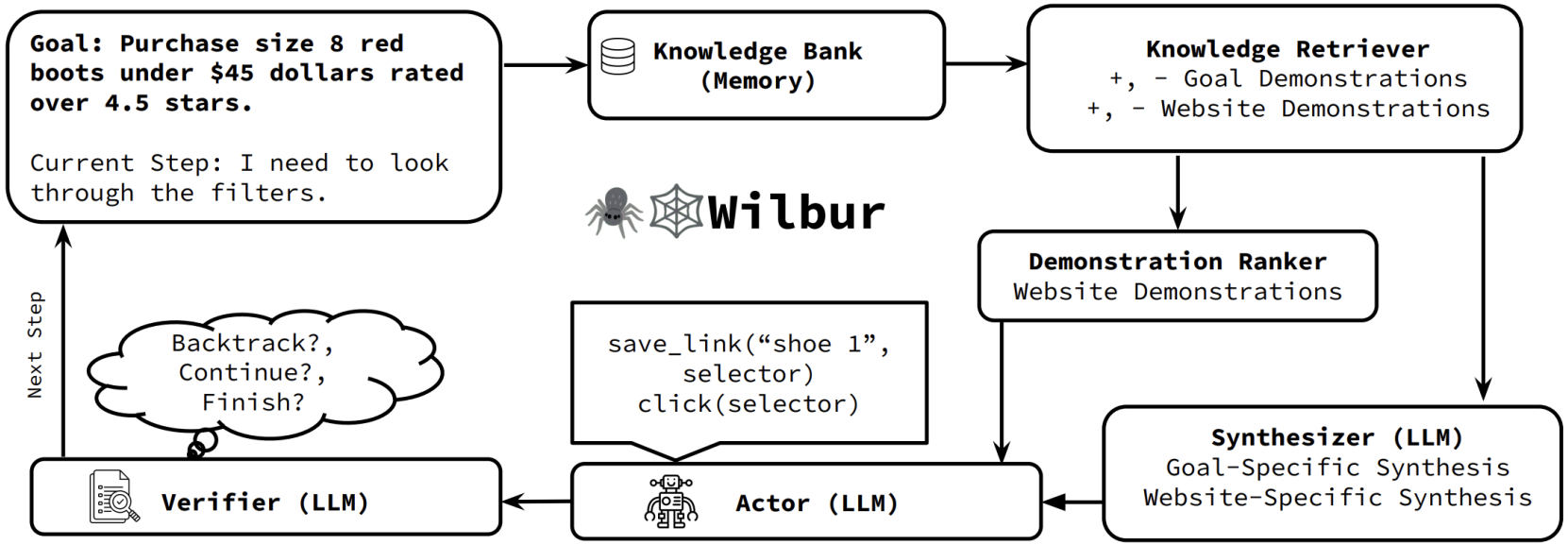
- Goal-conditioned demonstrations teach WILBUR how to perform a similar task on a potentially unseen website
- Website-conditioned demonstrations teach WILBUR how to act on a similar web page, regardless of the overall task.

Example: Web Agent (Cont'd)



- A dedicated demonstration ranking model to select the most helpful ones: This model is trained to predict whether the actions will lead to a successful execution or not and optimally populates a model's context.
- Summarizing a large sample of successful and unsuccessful actions into concise instructions.

Example: Web Agent (Cont'd)



- WILBUR proceeds by executing an action sampled from an LLM.
- After observing the new page state, it queries a reflection LM to verify that the action contributed progress toward the goal.
- If verification fails, WILBUR dynamically backtracks to a previous successful state, while storing the failure in the model's context for all future steps.

Graph-based Agent Planning with Parallel Tool Use

https://www.youtube.com/watch?v=B9CKm8J9sHY

This Lecture

- Search for Planning
  - Monte Carlo Tree Search (MCTS)
- Workflow Design
- Process Formulation
  - Planning Domain Definition Language (PDDL)

Process Formalization

- Formalizing planning through symbolic representations, programming languages, or logic frameworks.
- Incorporating symbolic logic or procedural programming into LLM prompting or output generation.
- These representations enable downstream tool execution and interface more cleanly with classical planners or robot controllers.

Process Formalization: PDDL Task

- **Planning Domain Definition Language (PDDL)** is the standard, Lisp-inspired language for modeling AI planning problems. It enables automated planners to generate action sequences to reach goals.

- PDDL separates planning into a domain file (types, predicates, actions) and a problem file (objects, initial state, goal).
  - Objects: Things in the world that interest us.
  - Initial state: The state of the world that we start in.
  - Goal specification: Things that we want to be true.
  - Predicates: Properties of objects that we are interested in; can be true or false.
  - Actions/Operators: Ways of changing the state of the world.

References

- https://pearls-lab.github.io/ai-agents-course/assets/lectures/search_planning_simulation.pdf
- https://www.ibm.com/think/topics/ai-agent-planning
- https://openreview.net/forum?id=7bJIVHEvLm
- https://arxiv.org/pdf/2404.05902