LLM Agents and Reasoning

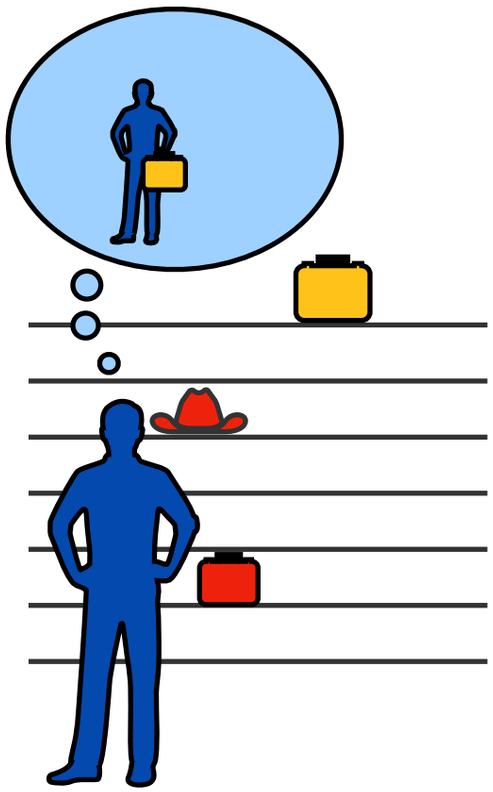
EECS 183/283a: Natural Language Processing

Agents

- What is an agent?

- Definition we will take here: an entity that takes actions that influence the state of the **dynamic** system in which it is embodied
  - Simple vision+language tasks (captioning, entailment): non-agentic, because context is a static image
  - What kinds of contexts are dynamic?

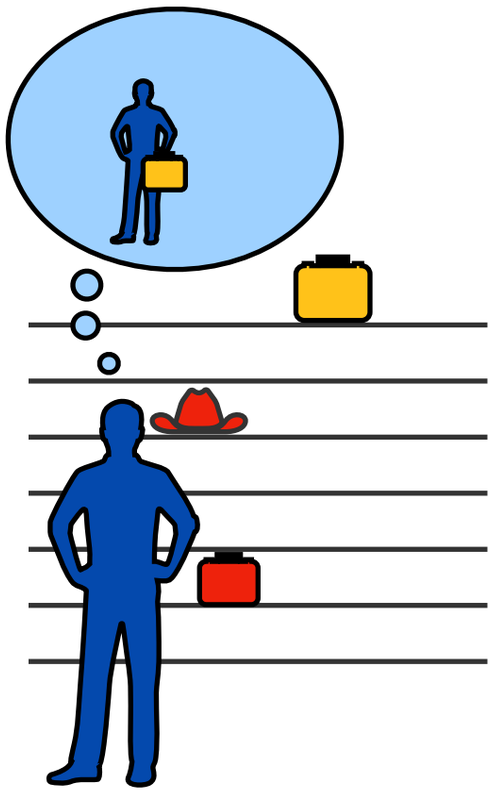
Partially Observable Markov Decision Processes



<u>My goal:</u>
Get the suitcase

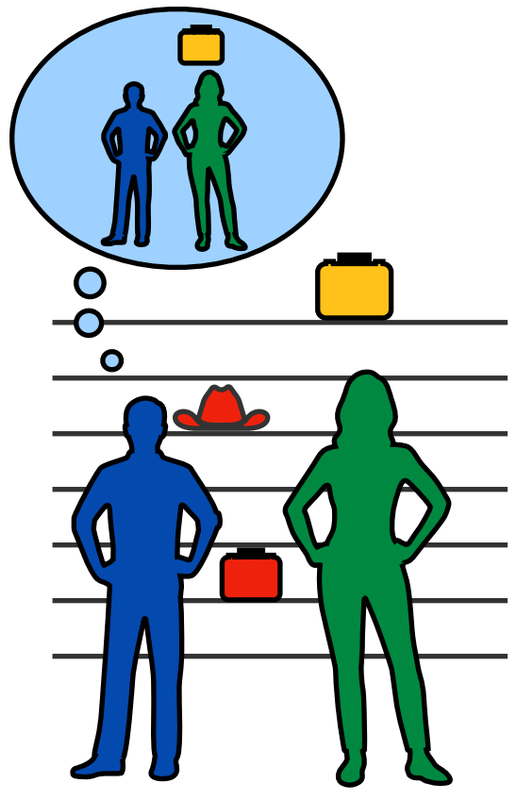
but… I'm too short

Partially Observable Markov Decision Processes



My goal:
Get the suitcase

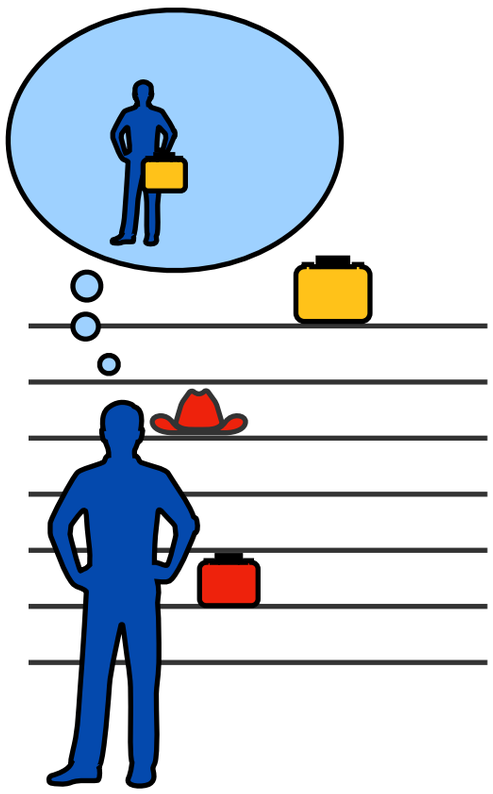
but… I'm too short



Observation:
Green person can reach it

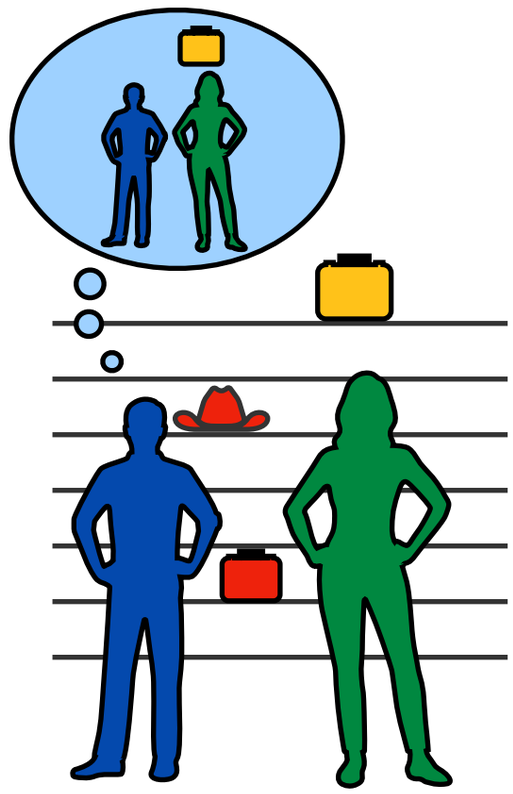
Optimal action:
Green person gives it to me

Partially Observable Markov Decision Processes



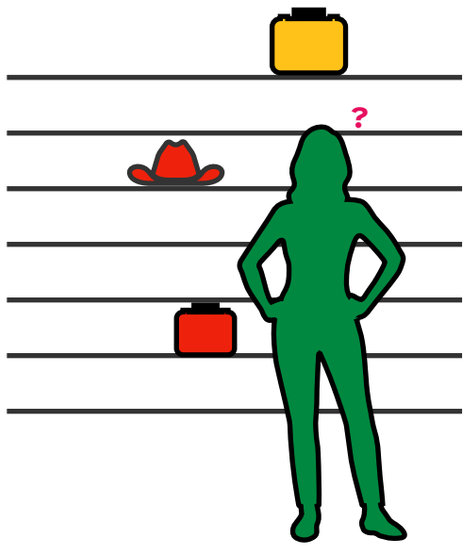
My goal:
Get the suitcase

but… I'm too short



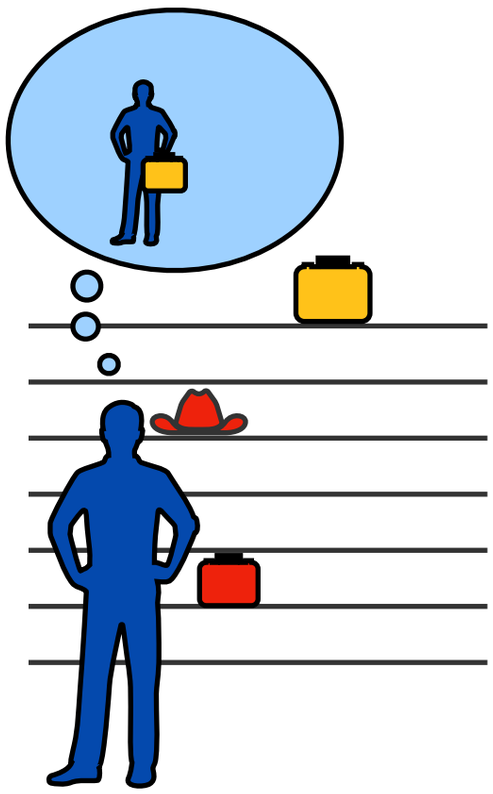
Observation:
Green person can reach it

Optimal action:
Green person gives it to me



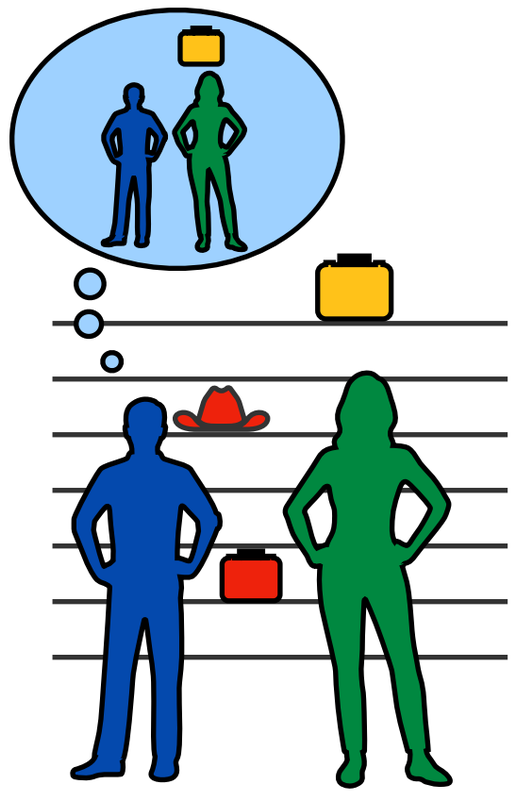
Belief:
Green person doesn't know my goal

Partially Observable Markov Decision Processes



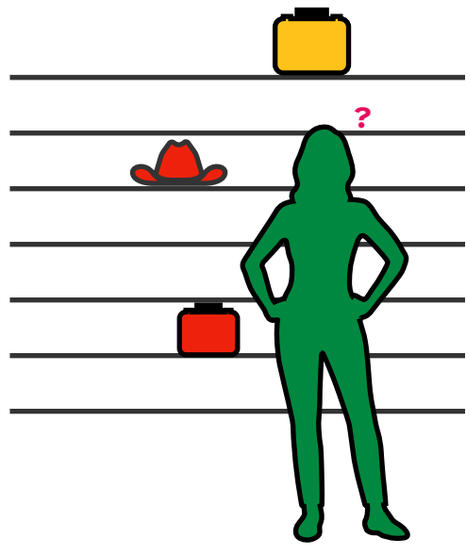
My goal:
Get the suitcase

but… I'm too short

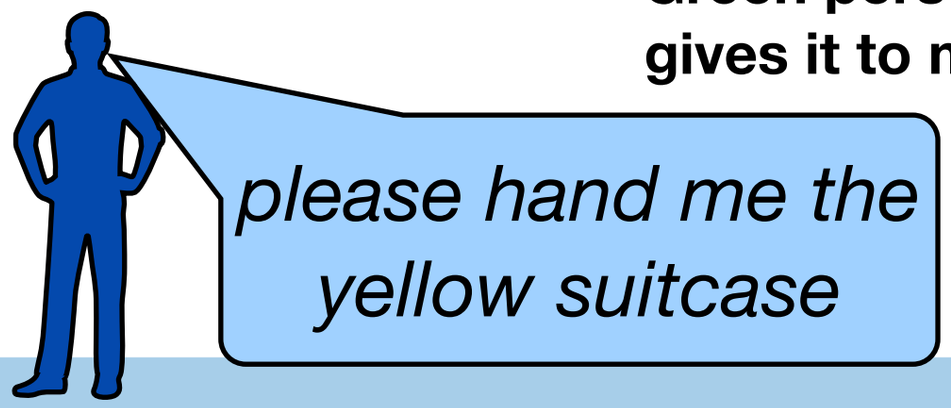


Observation:
Green person can reach it

Optimal action:
Green person gives it to me



Belief:
Green person doesn't know my goal



Task:
Language Generation

States:

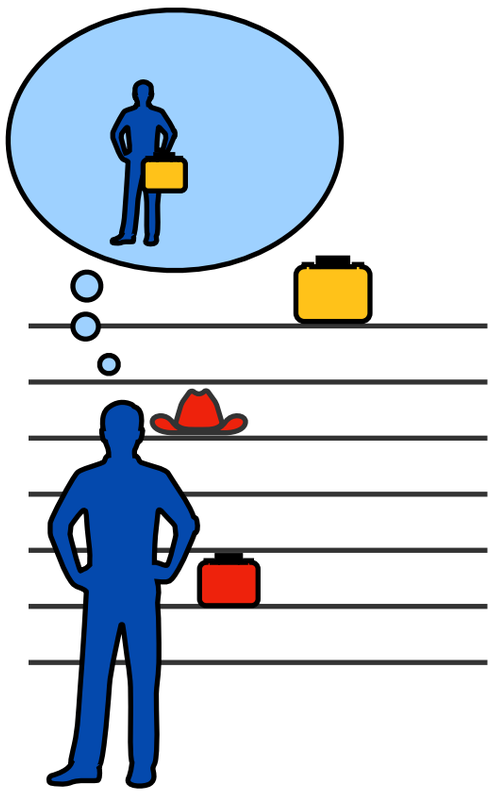
Observations:

Actions:

Transition function:

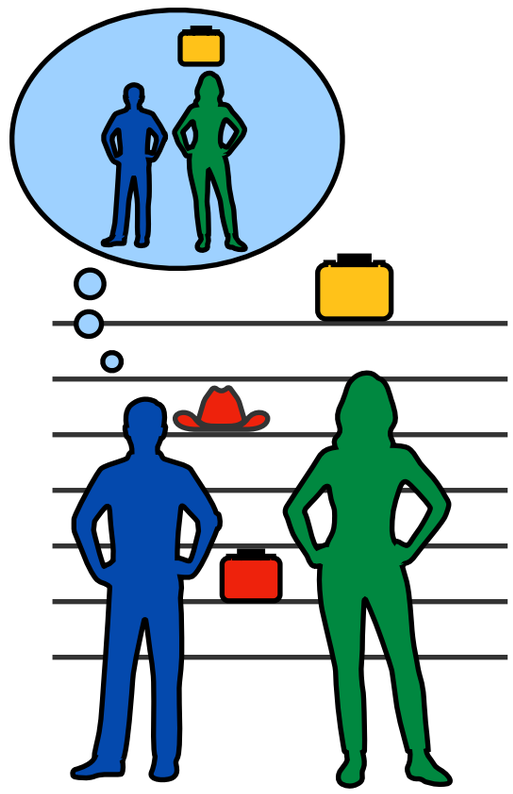
Reward function:

Partially Observable Markov Decision Processes



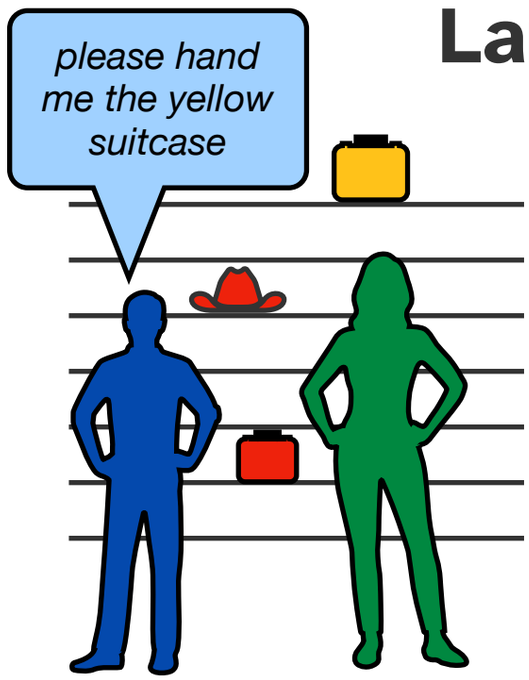
My goal:
Get the suitcase

but… I'm too short



Observation:
Green person can reach it

Optimal action:
Green person gives it to me



Belief:
Green person doesn't know my goal

Task:
Language Understanding

States:

Observations:

Actions:

Transition function:

Reward function:

POMDP:
Grounded Instruction Following

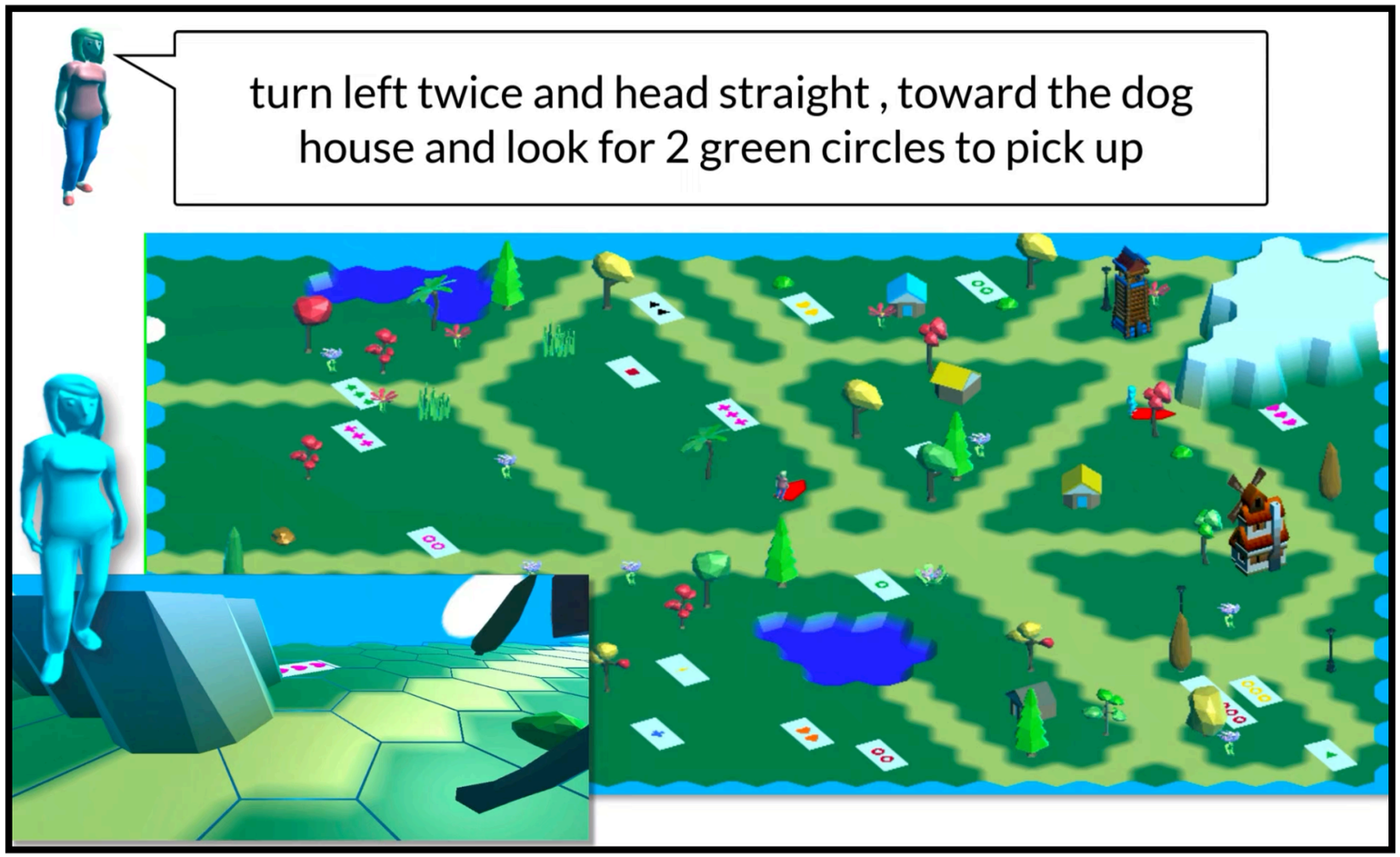
States:

Observations:

Actions:

Transition function:

Reward function:



CerealBar, Suhr et al. 2019

POMDP:
Software Engineering

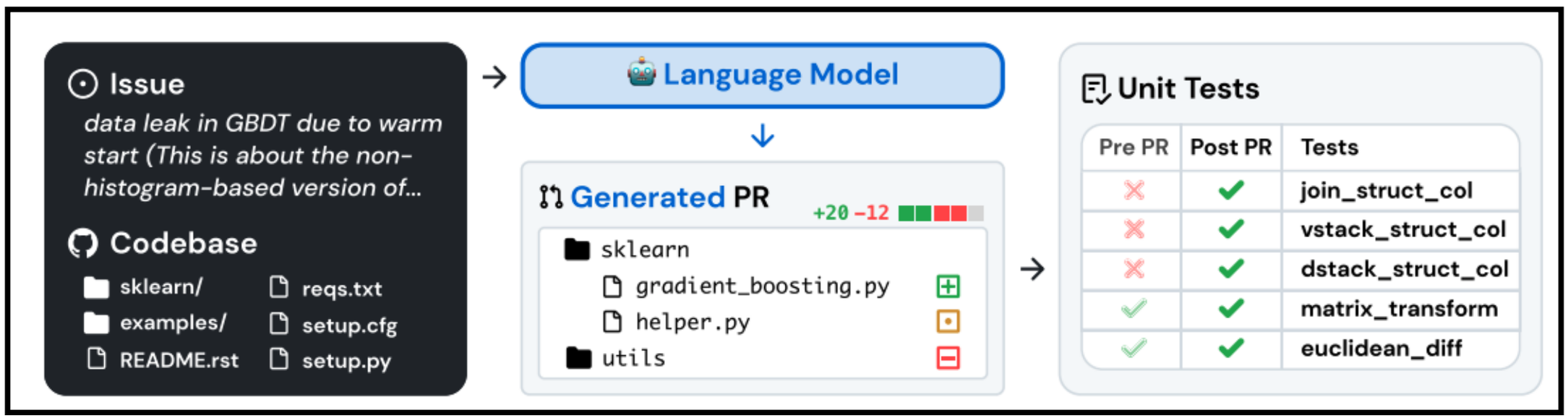
States:

Transition function:

Observations:

Reward function:

Actions:



SWE-Bench, Jimenez et al. 2023

POMDP:
Device Control

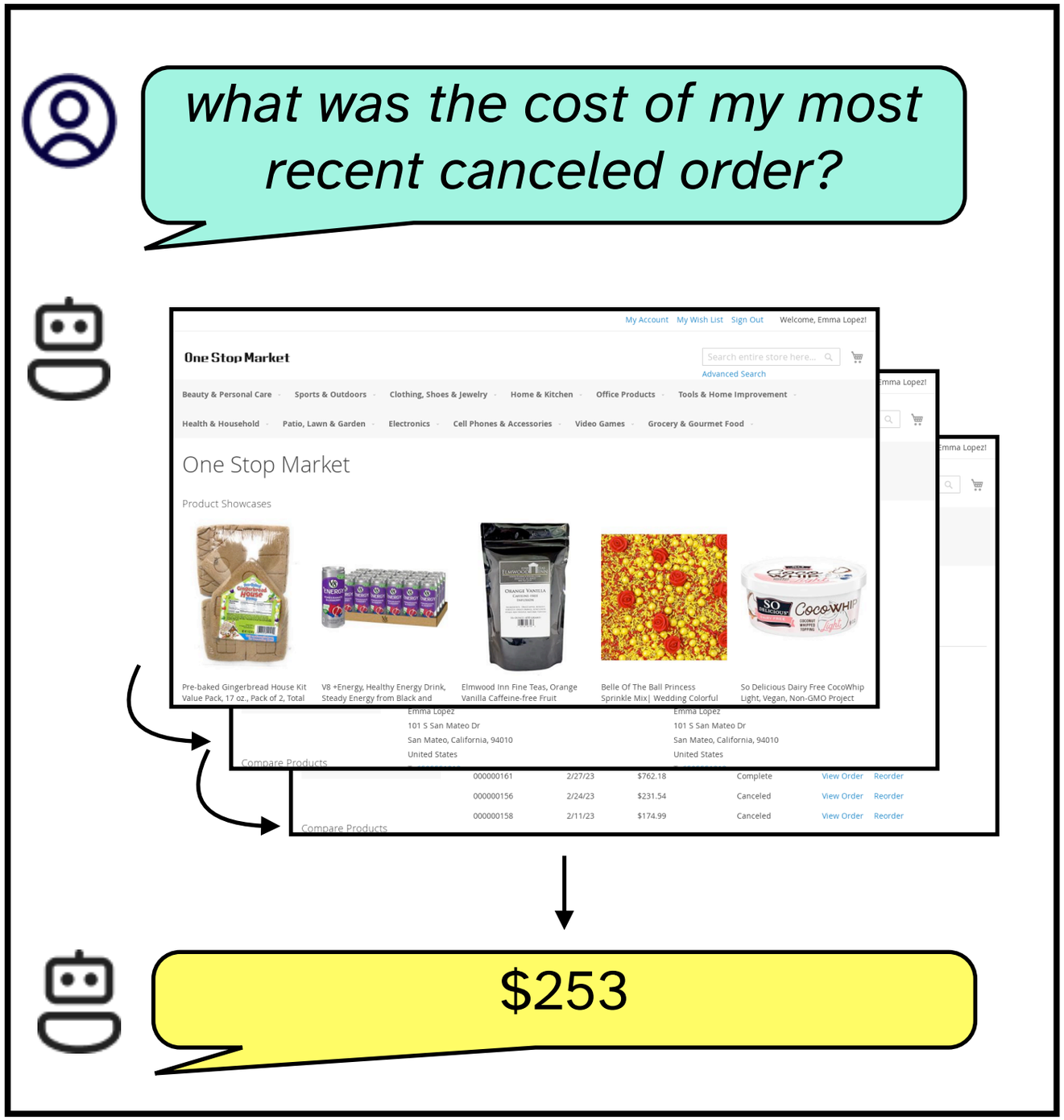
States:

Observations:

Actions:

Transition function:

Reward function:



WebArena, Zhou et al. 2023

Agent Challenges

- Input and output space is domain-specific, and pretrained LMs may struggle to generalize to new domains
- Policy must reason about environment dynamics
  - Sequential decision-making: chance for error propagation!
  - How have we addressed error propagation in the past? (hint: remember RLHF?)

Domain Generalization via Tools

- Instead of expecting model to do everything itself, **build tools that we can prompt it to use**

- Benefits:
  - Tools can more easily provide guarantees of success (e.g., calculator)
  - When prompted right, instruction-tuned LLMs can reliably call tools and use their results
  - LLMs don’t need to be fine-tuned to successfully take actions in target domains

Slides adapted from Zora Wang

Kinds of Tools

- **Perception:** collect new data from the environment
- **Action:** exert actions by changing the environment state
- **Computation:** offload computation onto an actual computer

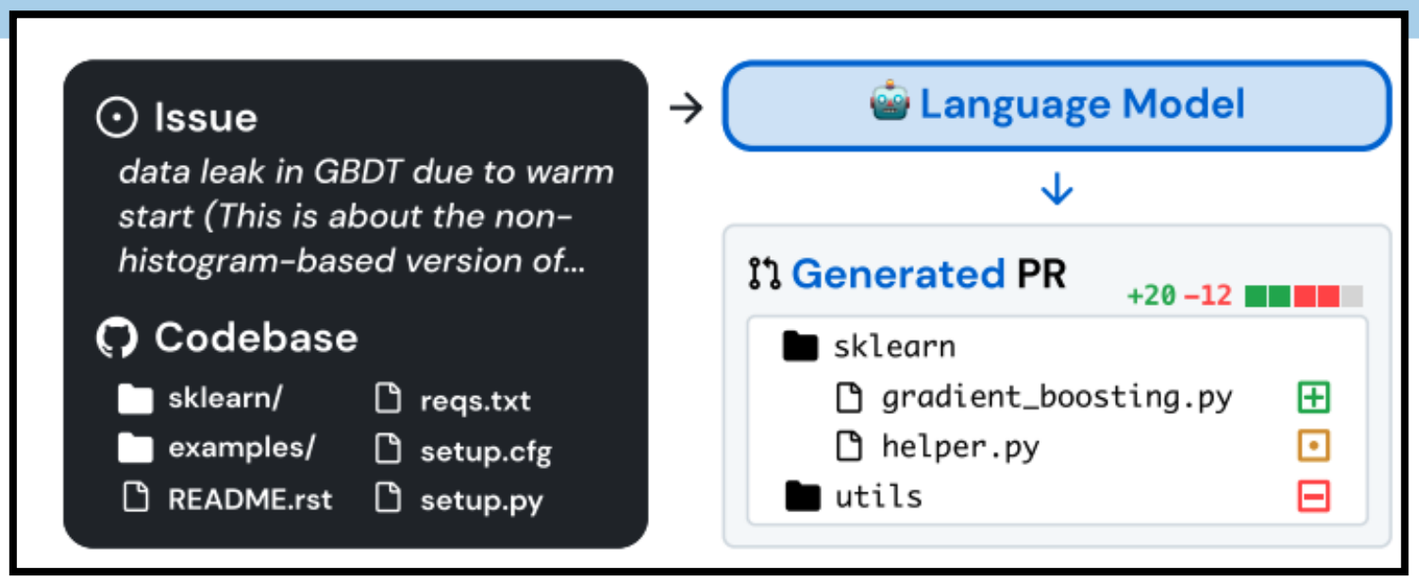
Kinds of Tools

Edit this image so there are twice as many cats on the porch.

Scenario: Embodied Instruction Following

- **Perception:** collect new data from the environment
- **Action:** exert actions by changing the environment state
- **Computation:** offload computation onto an actual computer

Kinds of Tools

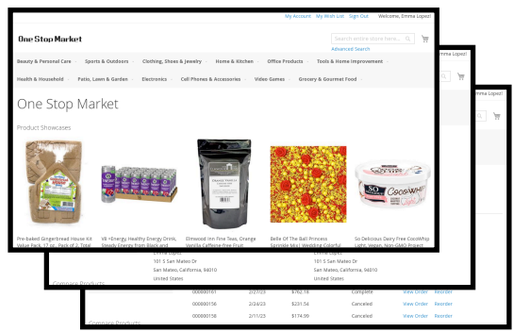


Scenario: Automated Software Engineering

- **Perception:** collect new data from the environment
- **Action:** exert actions by changing the environment state
- **Computation:** offload computation onto an actual computer

Kinds of Tools

what was the cost of my most recent canceled order?



Scenario: Device Control

- **Perception:** collect new data from the environment
- **Action:** exert actions by changing the environment state
- **Computation:** offload computation onto an actual computer

Prompt-Based Agents

- Defining the POMDP ourselves allows us to deploy LLMs as agents in a variety of domains
  - Observation space
  - Action space and transition function (tools)
- The right prompt design and inference method can further improve agent performance, e.g.:
  - Prompt the model to plan its actions, and revise its plan, before executing the plan
  - Sample multiple possible trajectories, then choose the one with the highest potential for success with some reward model
- Supervised fine-tuning with expert demonstrations can also help!

Error Propagation in Sequential Decision Making

- Core challenge in dynamic systems: executing a policy may result in states for which it has poor estimates of what's best to do, e.g.:
  - Stuck in a corner
  - Repeated the same action over and over
  - Opened an app it was never trained on that looks similar to the correct app
- Problem: even SFT'd models haven't been trained on what to do, because experts rarely get into these situations

Reinforcement Learning for Language Agents

- During learning:
  - Sample a trajectory from the current policy
  - Assign the trajectory a reward
  - Optimize the policy to maximize the reward

- What we need:
  - States — including a diverse set of instructions!
  - Actions and transition function — an executable environment!
  - Reward function — some kind of automatic reward function!

Language Agents

Where do you see this going?

What would you want from a language agent?

Reasoning

- Chain of thought: generate a sequence of intermediate reasoning steps before producing a final answer

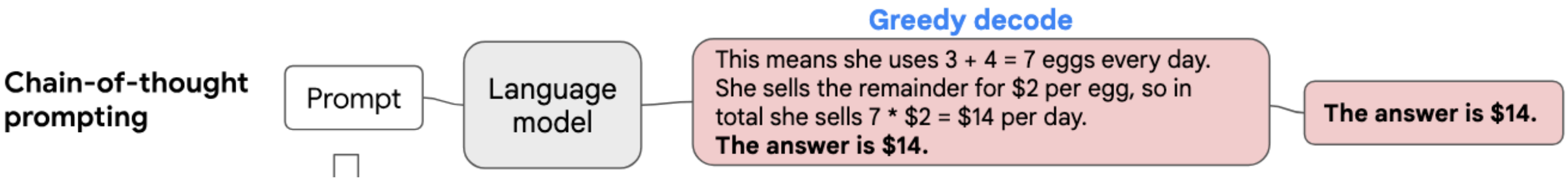
- Advantages:
  - Extra tokens provide adaptive computation time
  - If the chain of thought is faithful to the final answer, then it supports better verification of model output

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

A: Let's think step by step. The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9.

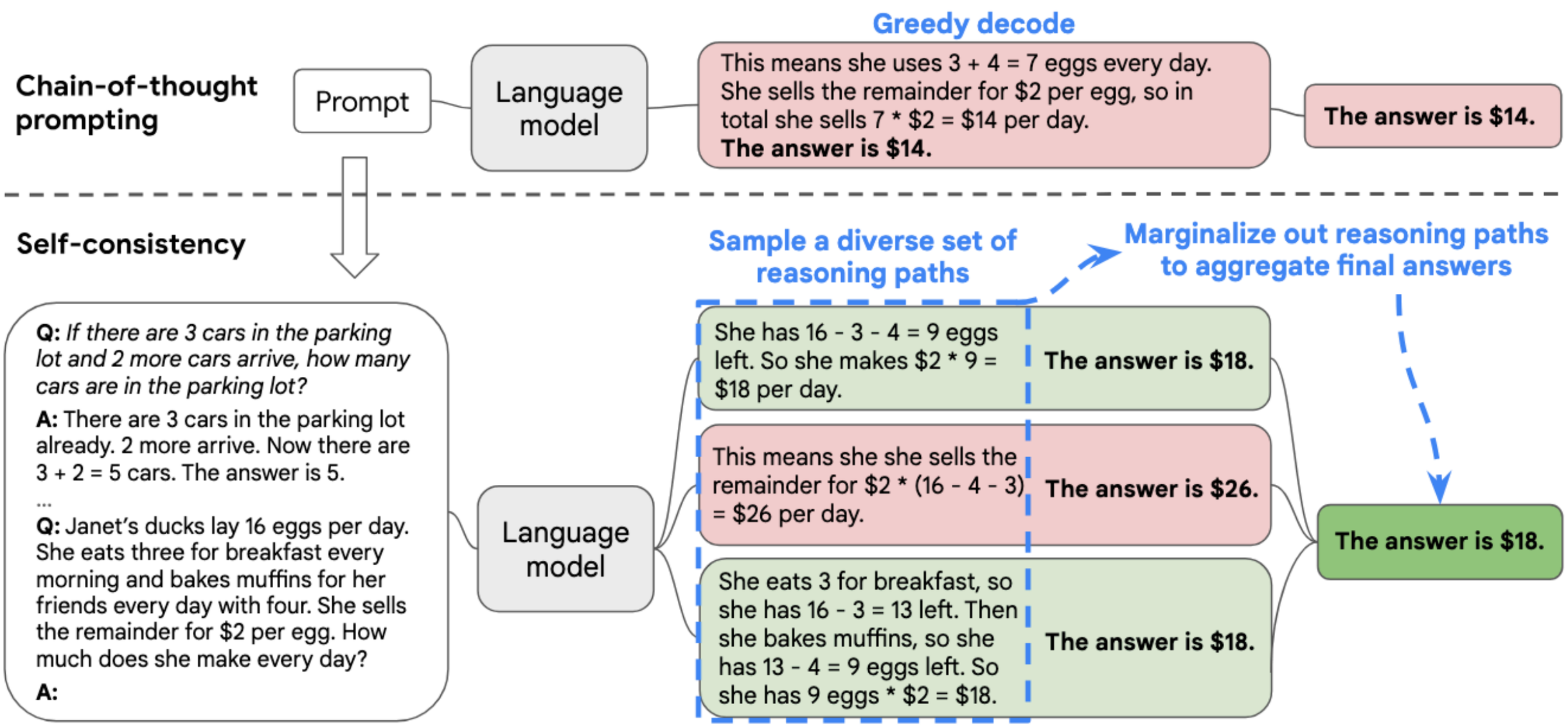
Structured Prompting

Self-Consistency



Structured Prompting

Self-Consistency



Main insight: by marginalizing over possible latent reasoning chains, we get better performance

Limitations of CoT

- Found to perform best on tasks involving math and logic, and less well on open-ended reasoning tasks (e.g., general natural language reasoning)

- Reasoning traces aren't necessarily faithful to:
  - The answers produced
  - The model's "underlying reasoning"

- When instructed to explain an *incorrect* answer to a question, models produce very plausible and confident explanations of why!

- Longer chains of thought generally lead to better model performance

Generation vs. Verification

- **Generation:** map from some task description to an answer/ solution
  - Zero-shot
  - Chain-of-thought
  - Structured prompting

- **Verification:** map from some task description and an answer/ solution, to a judgment of the answer/solution
  - Pass/fail
  - Numerical score
  - Natural language critique
  - Edit

Self-Refinement

- **Main idea:**
  - We have been using LLMs to do both *generation* and *verification* independently
  - Why not combine these into a single inference method to get better generation overall?

- In self-refinement, we iteratively:
  - Generate: map from task description to answer/solution, possibly given existing critique of past answers/solutions
  - Verify: map from task description and a proposed answer/solution to some type of feedback

- Terminate when the verifier is “satisfied”, or until some maximum number of iterations

Limitations of Self-Refinement

- Intrinsic self-correction often fails without external feedback (e.g., from an environment) or oracle feedback
- Performance may actually degrade from iteration to iteration!
- Reasons for failures:
  - Models struggle to identify errors they're likely to make
  - Models tend to judge initial answers/reasoning as correct (sycophancy?)
  - Models can't correct what they don't know
- Works well for surface-level errors (e.g., formatting) and with strong verifier models
- Works poorly when the task requires deep reasoning

Reasoning Models

- CoT is a powerful approach for getting better performance on any instruction-tuned model

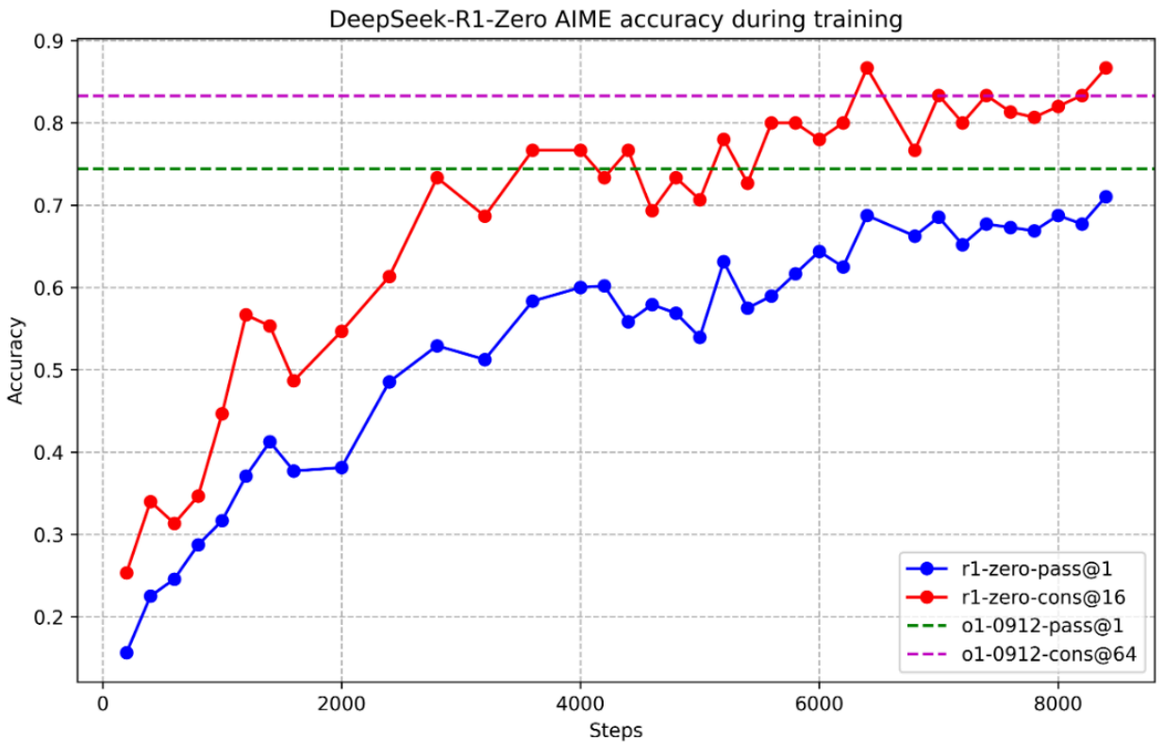
- A reasoning model is explicitly trained to use generate long chains-of-thought (CoTs)

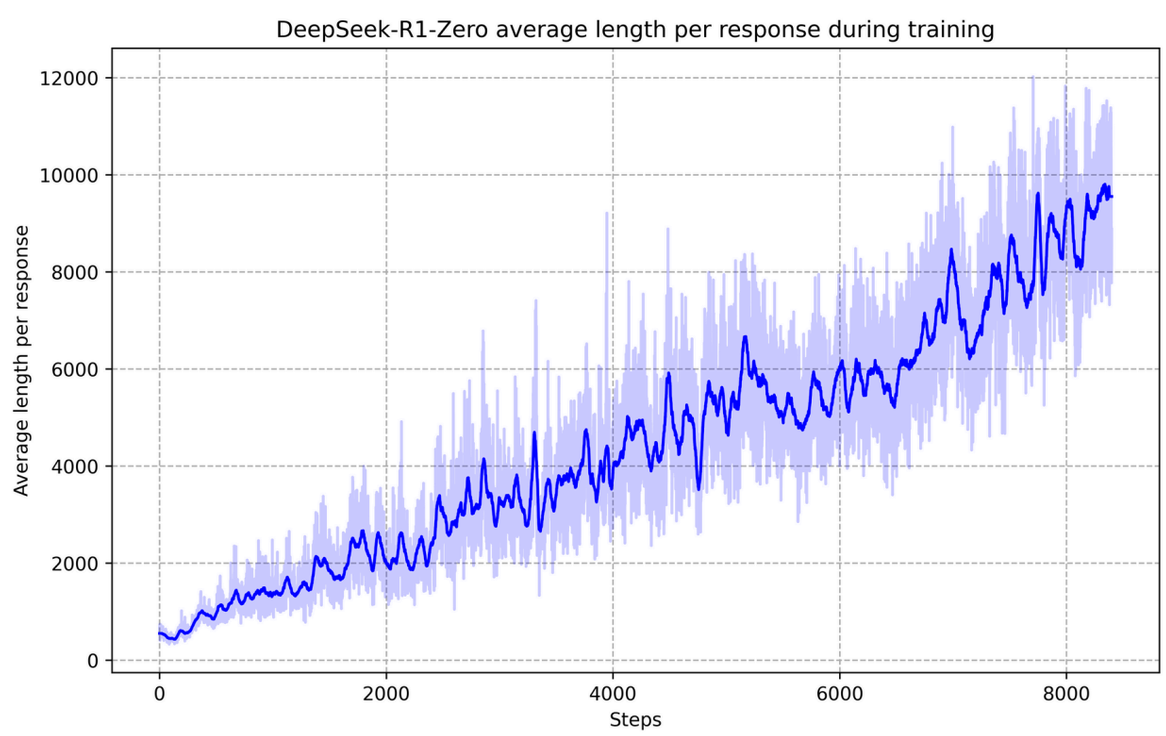
Self-Taught Reasoner (STaR)

- During training, iteratively:
  - Map from questions to answers with CoT using existing policy, resulting in <question, CoT, answer> tuples
  - Modify tuples based on correctness:
    - If the answer is correct, keep it as is
    - If the answer is incorrect, prompt policy to map from question and correct answer to a new CoT (“rationalization”)
  - Fine-tune policy on modified tuples
- Why not train only using rationalizations?

DeepSeek-R1

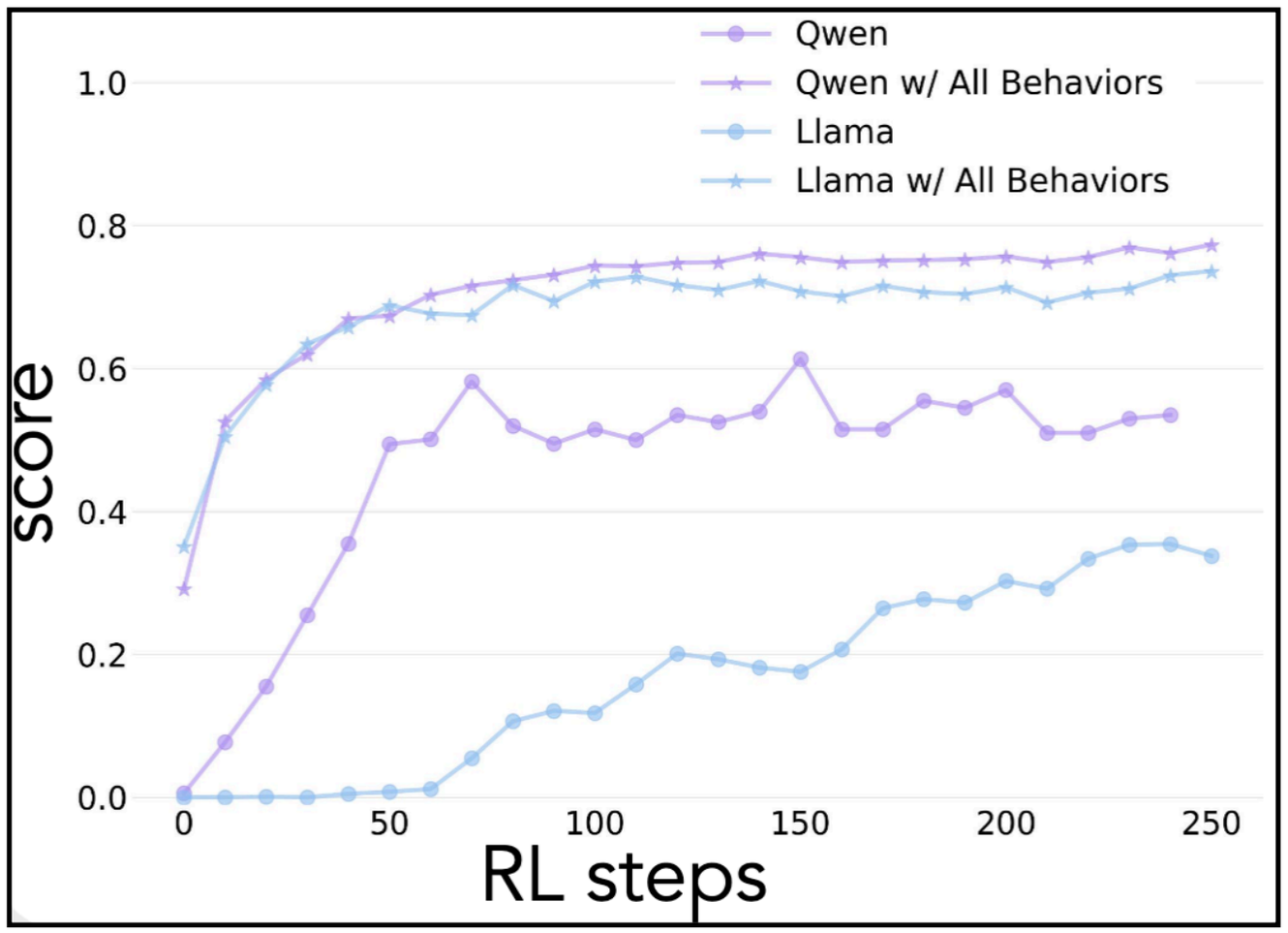
- During training, iteratively:
  - Map from questions to answers with CoT using existing policy, resulting in <question, CoT, answer> tuples
  - Reward tuples based on correctness
  - Optimize policy to maximize reward





Four Critical Reasoning Behaviors

- Self-verification
- Subgoal setting
- Backtracking
- Backward chaining



- **But:** faithfulness and interpretability problems remain

Some Remaining Questions in Reasoning Models

- How to train most effectively — online RL? pretraining data like s1?
- Under what conditions do different reasoning strategies “emerge”?
- How faithful are learned long CoT to actual reasoning processes? How interpretable are the reasoning traces?
- How well does learning to reason on one task generalize to other tasks?
- **How can we more effectively take advantage of inference-time compute?**

Slides adapted from Graham Neubig