Lecture 11:

# Creating AI Agents (LLM-based problem solving)

Visual Computing Systems
Stanford CS348K, Spring 2024

# Today

- Trends/techniques using LLMs as engines for generating behaviors for characters in virtual worlds
  - Or for problem solving agents in general

- Focus on the principles of evaluation

# Why is there interest in creating agents that operate in virtual worlds?

# Train embodied agents in simulation prior to deploying them in the real world

Using the virtual world as a proxy for the real world. Note this is NOT the use case we will discuss today.


NV Drive Sim

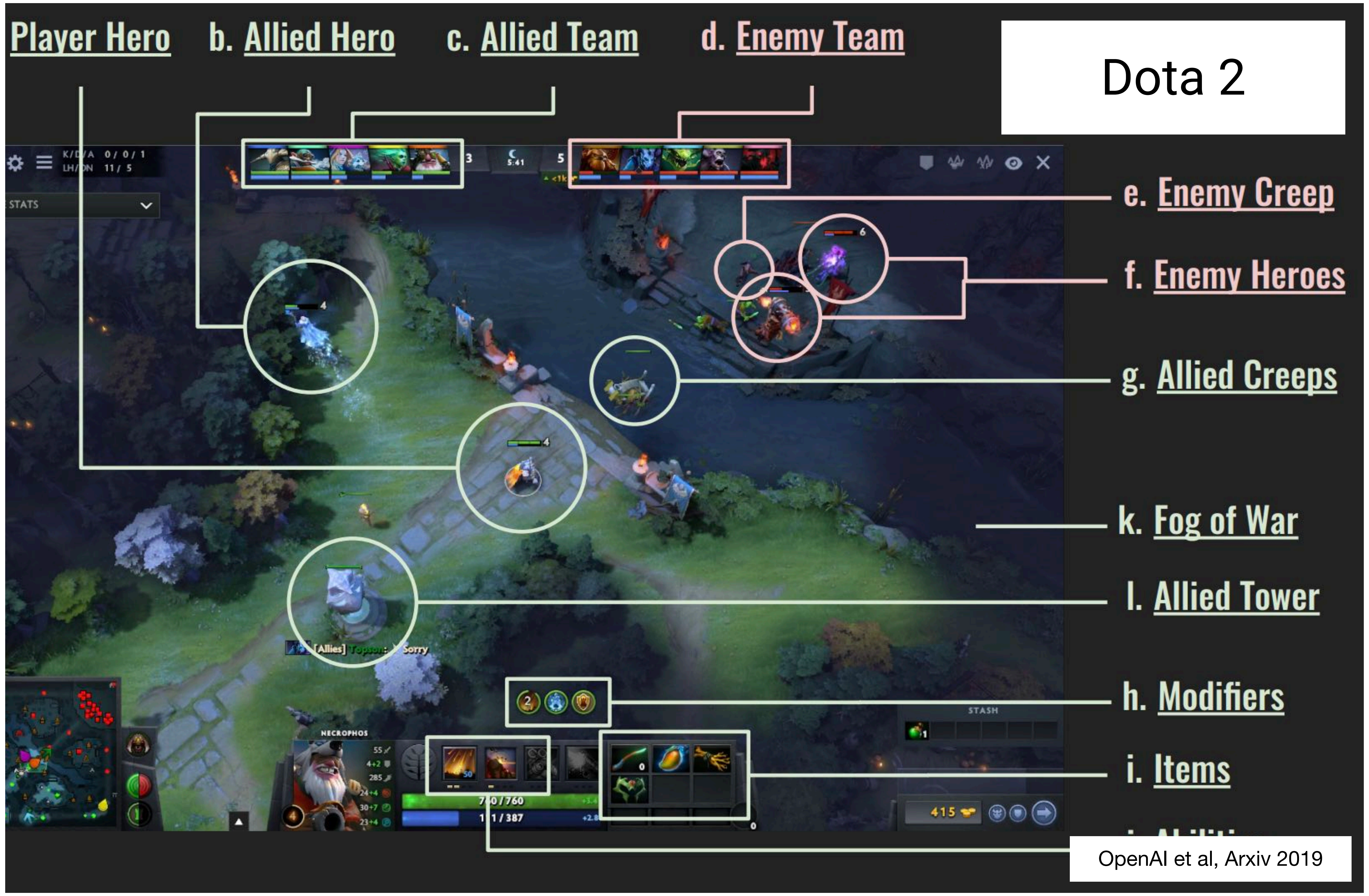
# Train embodied agents in simulation prior to deploying them in the real world

**Using the virtual world as a proxy for the real world. Note this is NOT the use case we will discuss today.**

Szot et al, NeurIPS 2021

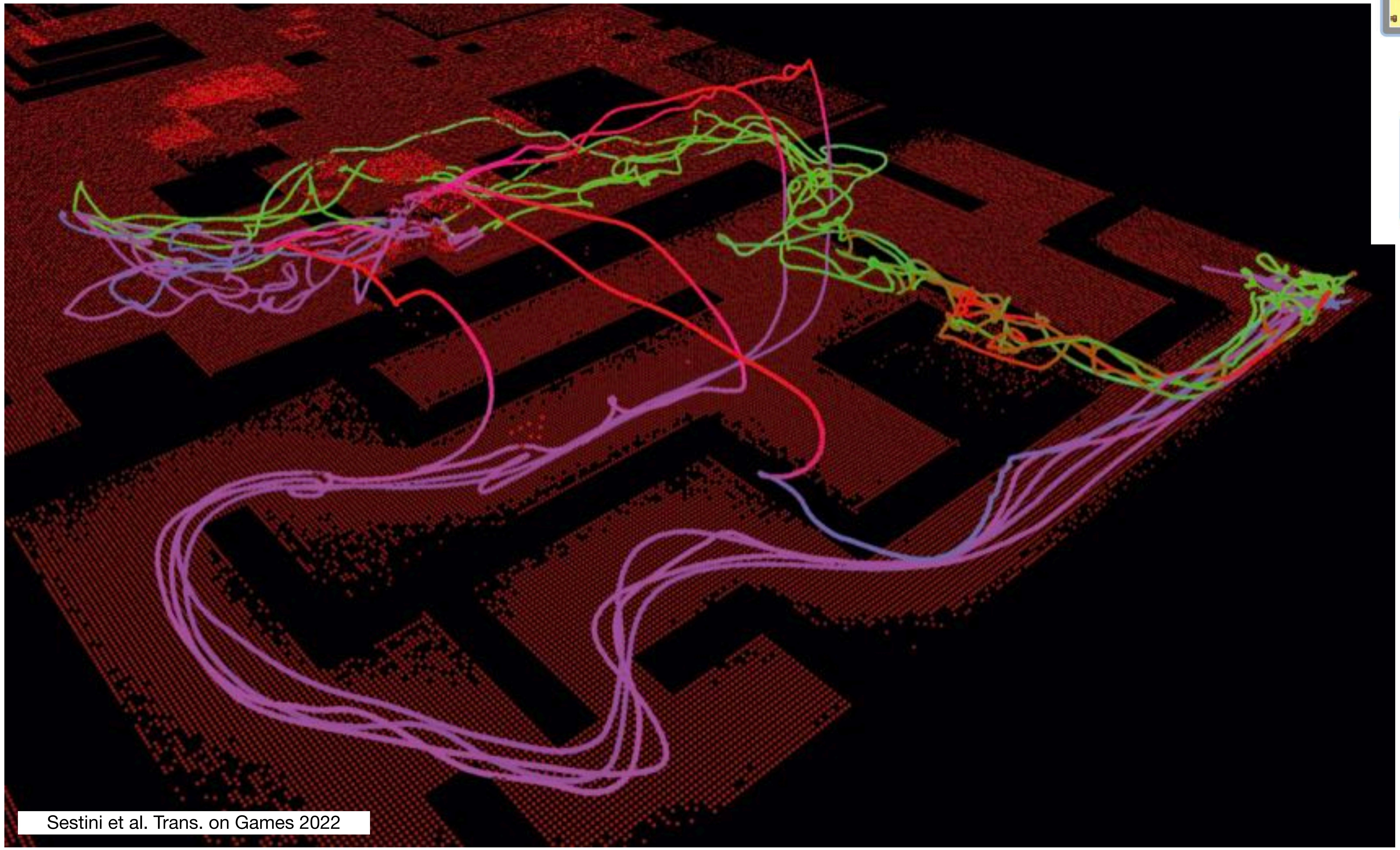
# As virtual characters in interactive experiences

- Virtual teammates in team-based games
- Narrative elements
- Coaches, etc.

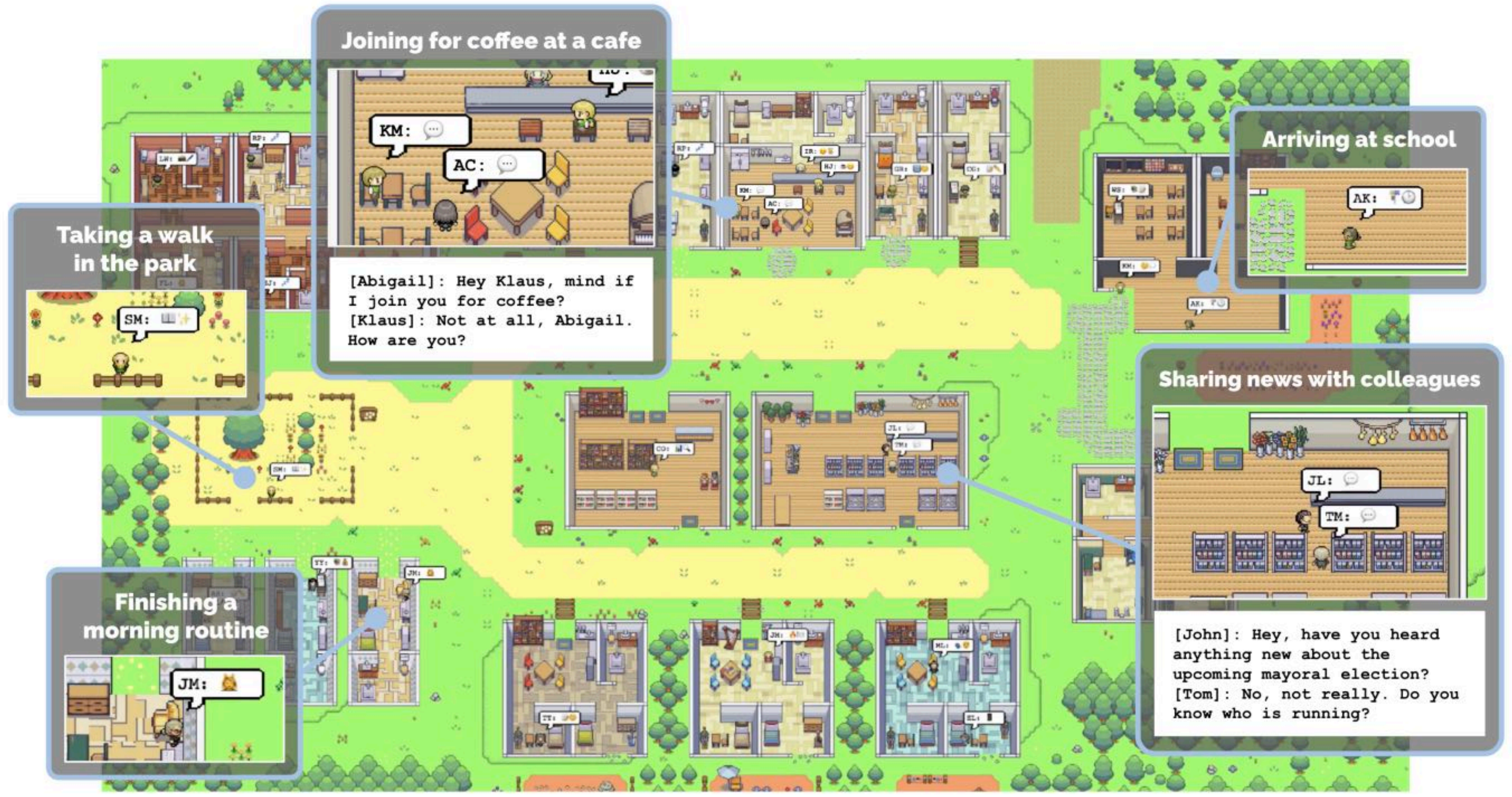



# As proxies for predicting future behavior of real humans

- Feedback on game design
- Game testing



Sestini et al. Trans. on Games 2022




Park et al. CHI 2023

**Note: in this case, agents in the virtual world are being used to simulate the plausible behavior of humans in the virtual world.**

**The goal is not using virtual world simulation to train agents to be capable to operate in the real world with humans.**

# General design strategies for LLM-based agents

# A simple problem solving activity

KA

Consider a multi-core processor that runs at 2 GHz and has 4 cores. Each core can perform up to one 8-wide SIMD vector instruction per clock What is the maximum throughput of the processor in units of floating point operations per second?

The answer is 64 GFLOPS (64 giga-floating point operations per second).

# Another problem solving activity

KA

I want to eat avocado toast, what are steps for making it?

Avocado toast is a delicious and nutritious meal that's quick and easy to prepare. Here are the steps to make a basic avocado toast:

**Ingredients:**

- 1 ripe avocado
- 2 slices of your favorite bread (such as whole wheat, sourdough, or rye)
- Salt and pepper to taste
- Optional toppings (e.g., red pepper flakes, sliced tomatoes, poached or fried egg, feta cheese, herbs, etc.)

**Instructions:**

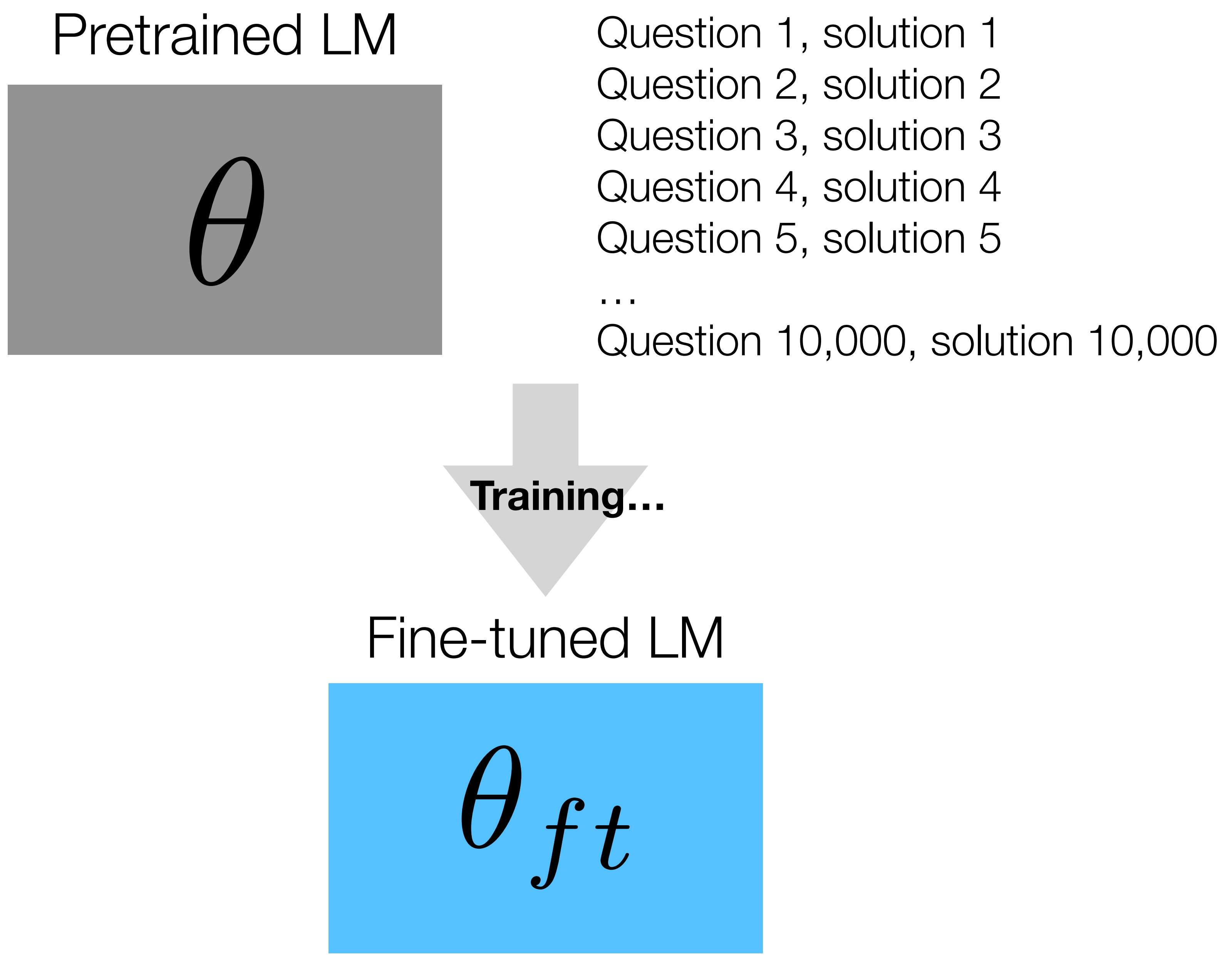
1. **Select a Ripe Avocado:** Choose a ripe avocado that yields slightly to gentle pressure when you squeeze it. If it's too firm, it won't spread easily on the toast.
2. **Slice the Avocado:** Cut the avocado in half by running a knife around the pit. Twist the two halves apart. Remove the pit by carefully striking it with a knife and twisting it out. Be very careful when doing this to avoid injury.
3. **Scoop the Flesh:** Use a spoon to scoop out the avocado flesh from each half. Place it in a bowl.
4. **Mash the Avocado:** Mash the avocado with a fork. You can leave it slightly chunky or make it smoother, depending on your preference.
5. **Season the Avocado:** Add a pinch of salt and some black pepper to the mashed avocado. You can also add a dash of lemon juice for some extra flavor and to prevent browning.
6. **Toast the Bread:** While you're preparing the avocado, toast the bread slices to your desired level of crispiness.
7. **Spread the Avocado:** Once the bread is toasted, spread the mashed avocado evenly over each slice.
8. **Add Toppings (Optional):** You can get creative with your toppings. Some popular options include sliced tomatoes, red pepper flakes, a poached or fried egg, feta cheese, herbs like cilantro or basil, or a drizzle of olive oil.
9. **Serve and Enjoy:** Your avocado toast is ready to be served! It makes a great breakfast, brunch, or even a quick and healthy snack.

Remember that avocado toast is highly customizable, so feel free to experiment with different toppings and seasonings to suit your taste. Enjoy your delicious and nutritious meal!

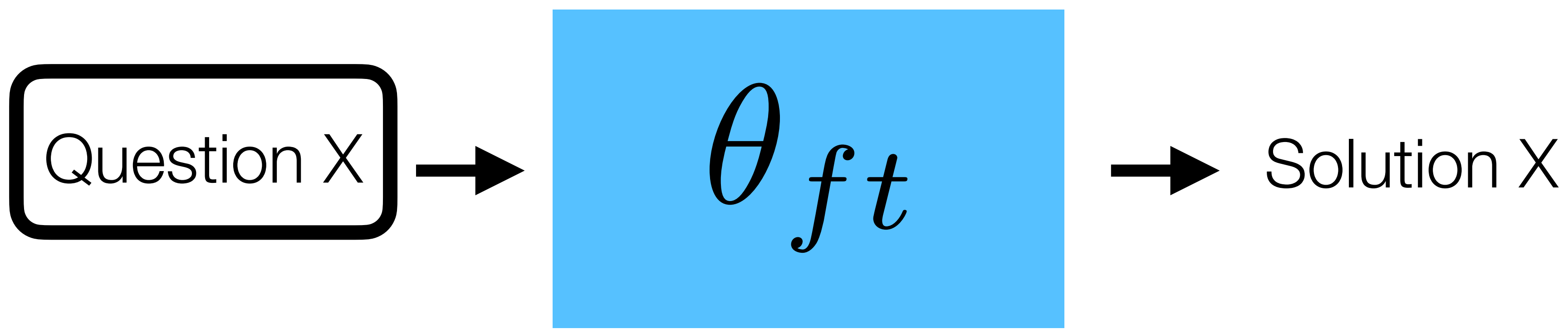
# Fine-tuning vs. in-context learning

## Fine-tuning:

**Step 1: Fine tune existing model on in-domain training set:**

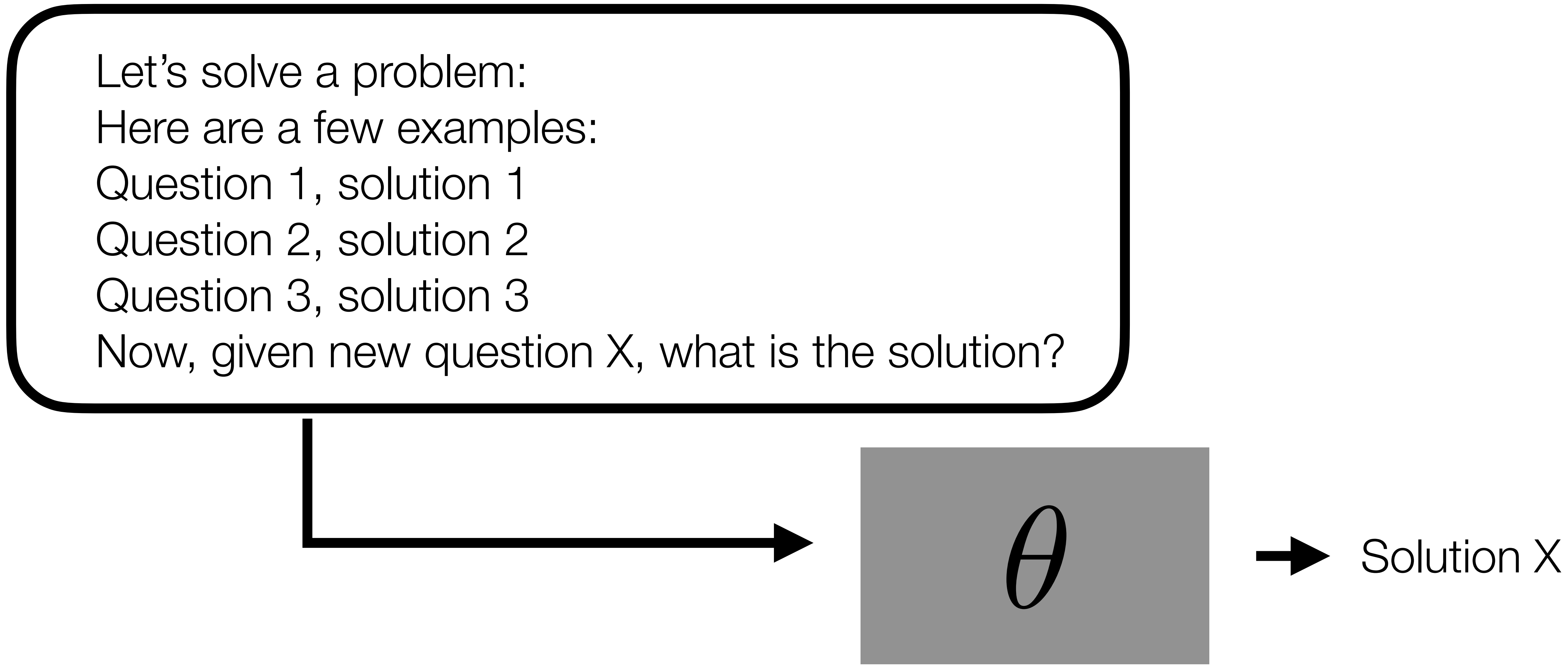



**Step 2: Test time execution**




## In context learning:

**Test time:**



# In-context learning example (failure)

**Model Input**

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

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

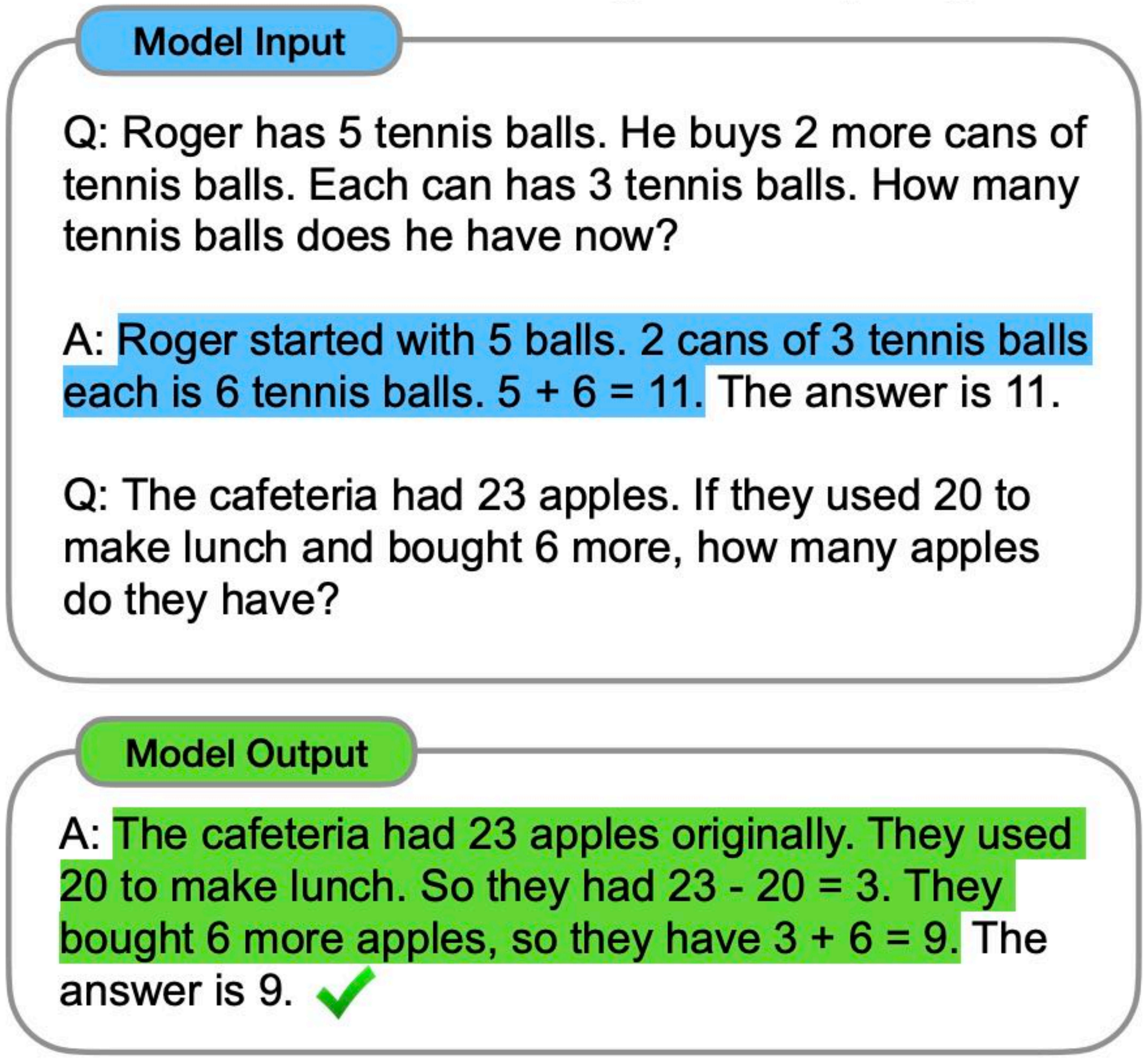
**Model Output**

A: The answer is 27. ❌

# Chain of thought prompting

[Wei et al. 2022]

**Main idea: modify the (in-context) prompted examples so they explicitly break down the solution into steps. By "coaching" the model to think step by step, the model is able to be more successful in its reasoning.**




No fine-tuning step.

Step-by-step "training data" harder to come by, in-context reasoning needs less of it.

By running in steps, model can expend more computation to solve problem

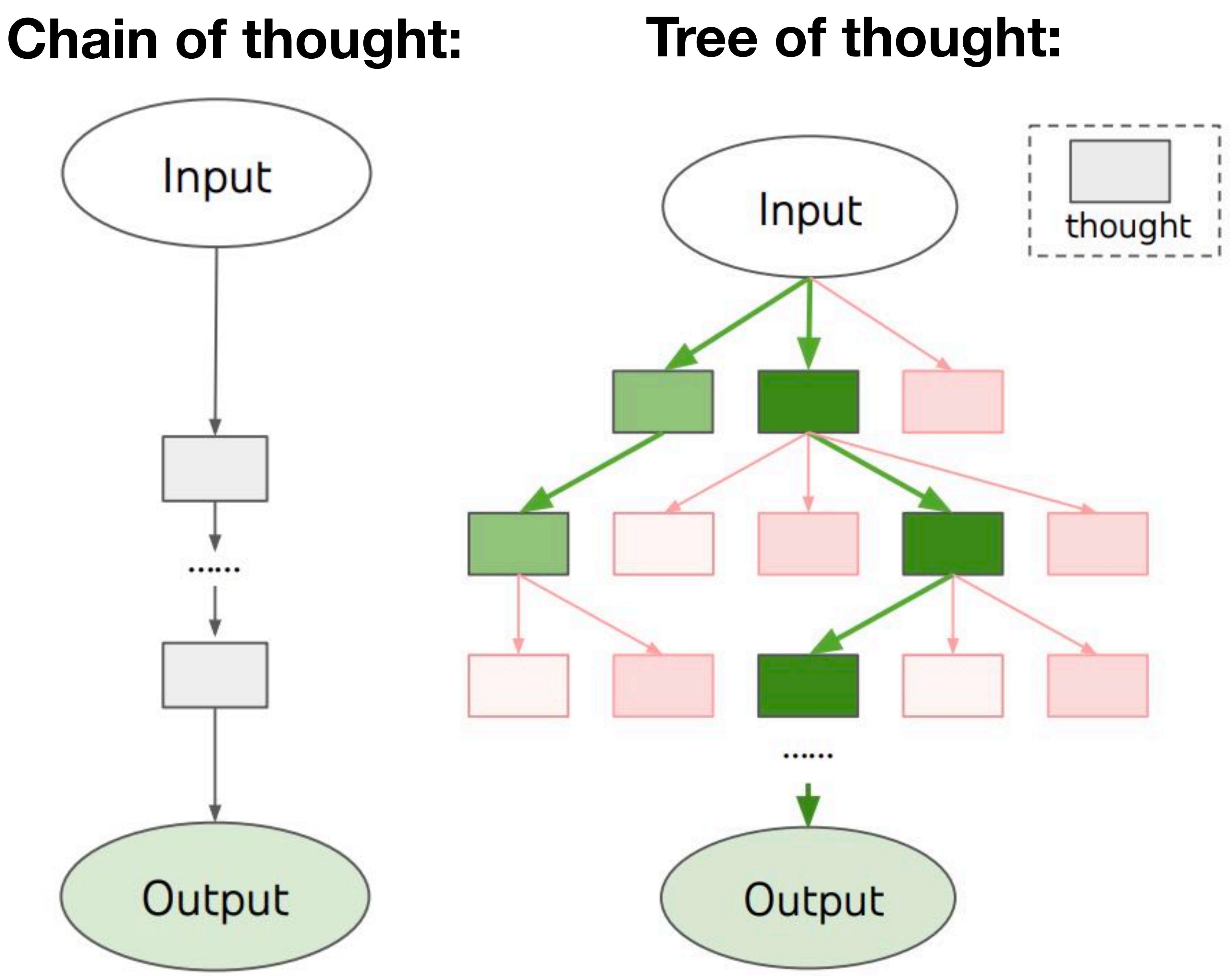
More interpretable answer, since chain of reasoning is revealed.

**Empirically: chain of thought needs larger language models to work.**

# Tree of thoughts

[Yao et al. 2023]

- Brings together traditional idea of backtracking search and chain-of-thought
- Generate answer step-by-step (like chain of thought)
- But generate multiple possibilities each step
- Choose most promising next step: use LM a judge ** to "score" the possibilities




**Judge possibilities:**

**Given this question, which step do you think is best?**

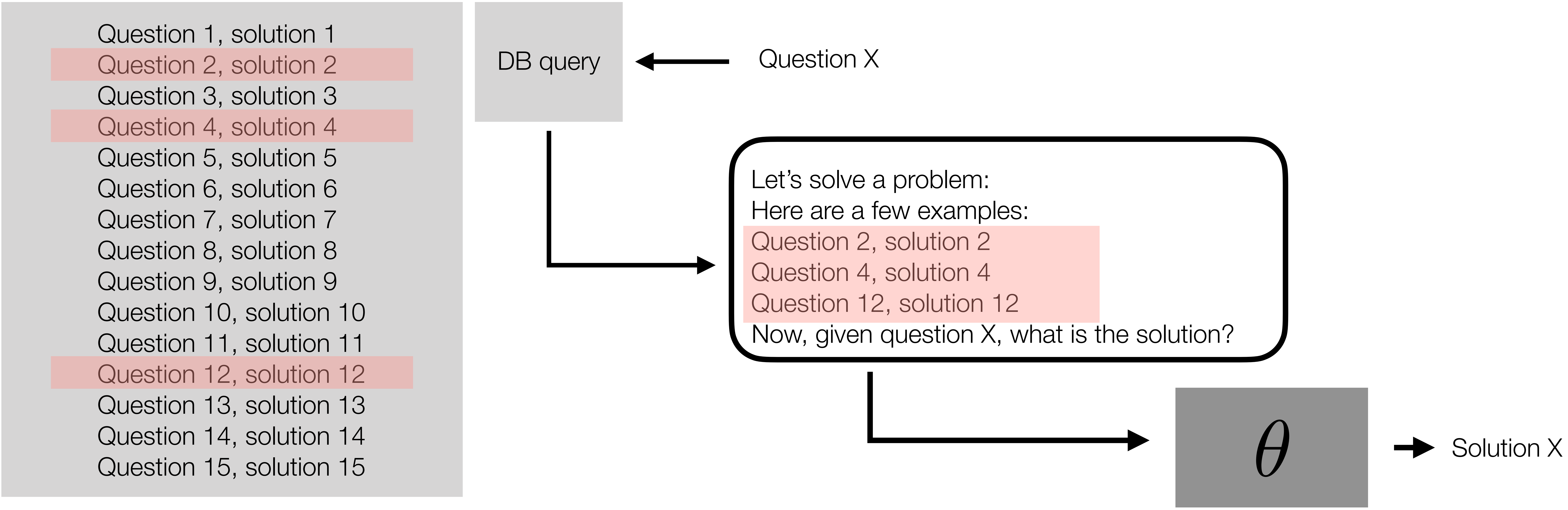
**OR**

**Compute some independent score for each option, take best.**

** The judge is a form of self reflection, see later in talk

# Enhancement: dynamic lookup into "Memory"

- **Challenge: LMs have finite input context length (e.g., 4096 tokens, etc.)**
- **Training dataset may have many examples**
- **Idea: choose examples are the "most relevant" examples to provide the LM as context**




- **Typical query solution: embed question string, K-NN lookup in "vector database"**

# Use of memory

- Lookup most relevant Q-A examples

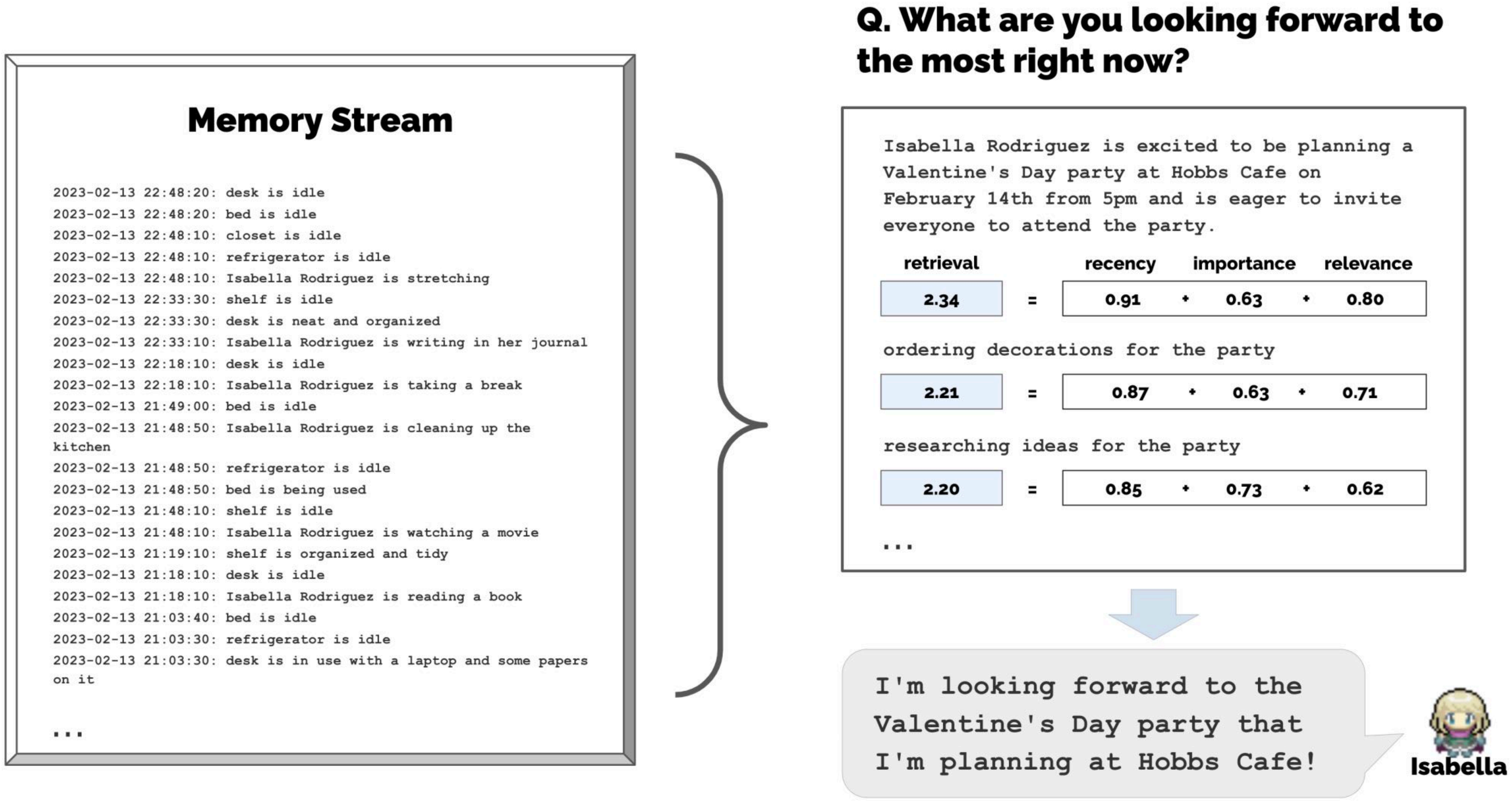
- Examples:
  - Robot planning: If the task is making a plan to "cook an egg", then in-context examples should be as relevant as possible. i.e, if you have an example plan for "cook a potato" in the DB then use it.

  - Social decision making: If the task is deciding what to do next when you see Maneesh, DB lookup might be retrieve all items in history that record interactions with Maneesh

# Use of memory

[Park et al. 2023]

## Example: retrieving from agent's history of interactions



# Self reflection

[Yao et al. 2022]

- If chain-of-thought was about forcing the model to "show your work", then self reflection is about forcing the model to "tell me why" you took a particular step

- Model outputs two pieces of information:

1. A list of actions (steps in a plan)
   - Actions modify "the world" and solicit new observations
2. A list of thoughts
   - Thoughts do not modify the world, but update the state of the planner (by forcing it to emit more tokens)

# Self reflection examples

[Yao et al. 2022]

### Example 1: Robot acting in virtual environment
### (Observations are symbolic representation of world state)

**(b) ReAct (Human Thought edit)**

......
**Obs 16:** You open the drawer 4. The drawer 4 is open. In it, you see a keychain 3, and a watch 2.
**Act 17:** Think [Now I find the first keychain (3). Next, I need to take it. ~~I can find the second keychain (2) later in drawer 4.~~ ]
**Act 18:** take keychain 3 from drawer 4
**Obs 18:** You pick up the keychain 3 from the drawer 4.
......
**Act 22:** put keychain 3 in/on safe 1
**Obs 22:** You put the keychain 3 in/on the safe 1.
**Act 23:** Think [Now I put the first keychain in safe. Next, I need to find the second keychain, which is more likely to appear in dresser (1), garbagecan (1), safe (1), sidetable (1), sofa (1), shelf (1-12).]
**Act 24:** go to dresser 1
**Obs 24:** On the dresser 1, you see a box 1, a creditcard 1, a keychain 2, a keychain 1, a laptop 1...
**Act 25:** Think [Now I find the second keychain (2). Next, I need to take it.]
**Act 26:** take keychain 2 from dresser 1
**Obs 26:** You pick up the keychain 2 from the dresser 1.
**Act 27:** Think [Now I take the second keychain (2). Next, I need to put it in/on safe 1.]
**Act 28:** go to safe 1
**Obs 28:** The safe 1 is open. In it, you see a keychain 3, a statue 2, and a watch 3.
**Act 29:** put keychain 2 in/on safe 1
**Obs 29:** You put the keychain 2 in/on the safe 1.

### Example 2: digital assistant browsing the web
### (Observations are web page contents)

**(1d) ReAct (Reason + Act)**

**Thought 1:** I need to search Cirque du Soleil show Mystere, find the hotel it is in, then find the number of rooms in the hotel.
**Act 1:** Search[Cirque du Soleil show Mystere ]
**Obs 1:** Could not find Cirque du Soleil show Mystere. Similar: ['Cirque du Soleil' …

**Thought 2:** To find the hotel, I can search Mystere (Cirque du Soleil).
**Act 2:** Search[Mystere (Cirque du Soleil) ]
**Obs 2:** Mystère (my-steer) is … Held in a custom theatre at the Treasure Island Hotel and Casino …

**Thought 3:** Mystere is held in a custom theatre at the Treasure Island Hotel and Casino. So I need to search Treasure Island Hotel and Casino next and find the number of rooms in the hotel.
**Act 3:** Search[Treasure Island Hotel and Casino ]
**Obs 3:** Treasure Island Hotel and Casino … with 2,884 rooms and 220 suites …

**Thought 4:** Treasure Island Hotel and Casino has 2,884 rooms and 220 suites. So the answer is 3,104.
**Act 4:** Finish[3,104]

Up-to-date

# Another form of reflection

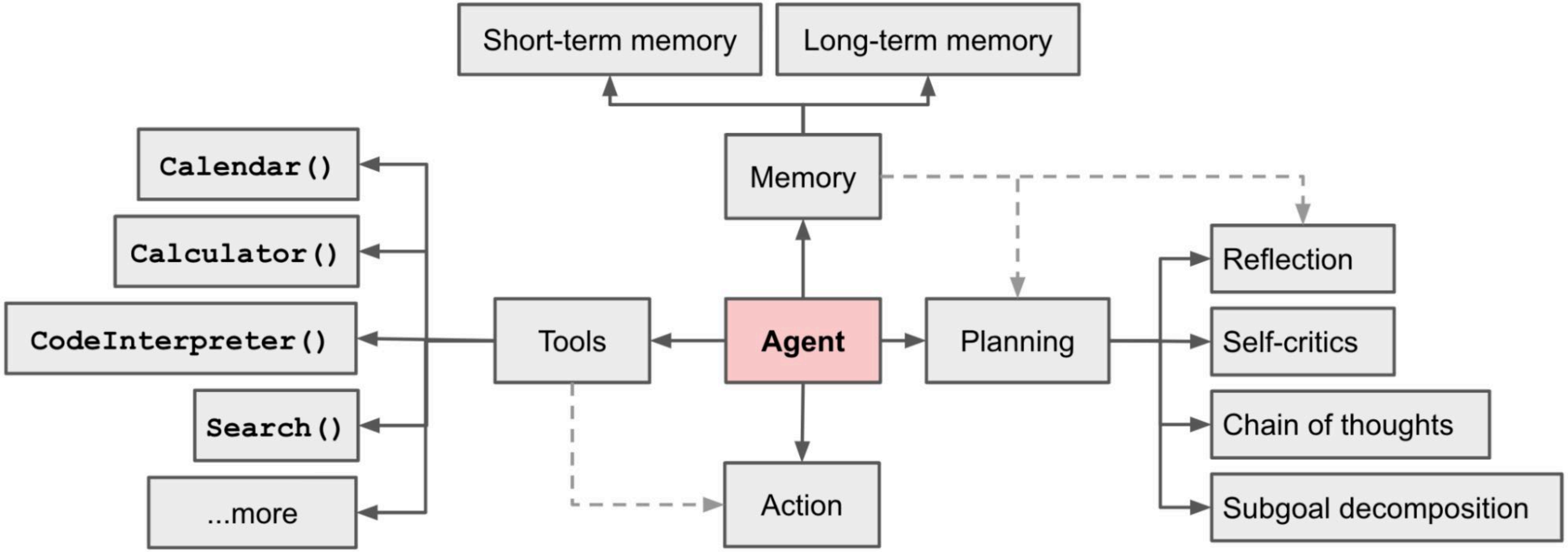
[Shin et al. 2023]

- Reinforcement learning formulation (assumes a world simulator exists)
- Given world state W and task description T (e.g., “get a glass of milk”), the agent executes a plan in the simulated world (“rolls out plan”)
- This generates a log of: (action_i, observation_i) pairs, and a “reward”
- Model model is asked to produce an English-language critique of what went wrong (e.g, what and where in the log)

- Now retry plan generation with: (W, T, log, reward, critique)

- We’ll see the same idea in the Voyager paper you’ll read tonight.

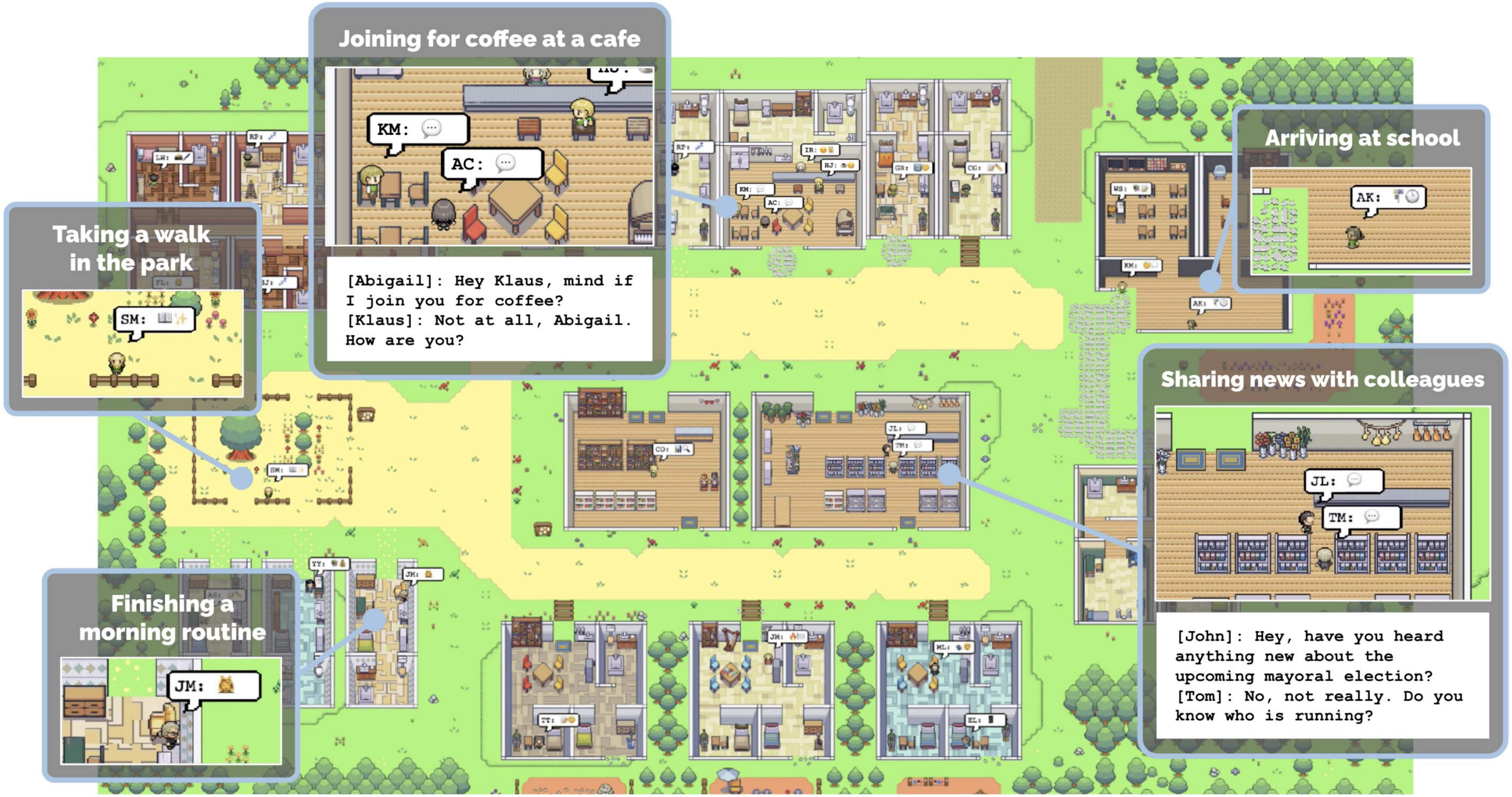
# General LLM-agent architecture




**[Image credit: Lilian Weng]**

# Discussion: Generative Agents

Park et al. 2023



# Before we begin the discussion: thinking about evaluation

# What is the purpose of evaluation?

- Any scientific paper proposes a hypothesis. You can think of a hypothesis as a claim that could be falsified with the right data

- The goal of the evaluation is to:
  - Show that the hypothesis is not falsified by experiments
  - Show that the hypothesis is falsified in certain situations, providing bounds on the hypothesis
  - Provide the reader a better understanding of the trends, limits, value of the hypothesis

# From our lecture 1 reading...

The scientific hypothesis is that the proposed organization will lead to improved X, Y, Z.

Where X, Y, Z might be one or more of performance, developer productivity, maintainability, reliability, safety, new functionality etc.

## The key insight: many good systems papers put forth a novel organizing observation about the structure of a problem.

Given stated goals and constraints, a systems paper will often propose a formulation of a problem that facilitates meeting these requirements. In other words, many systems papers make the argument:

*It is useful to think about the problem in terms of these structures (e.g., using these abstractions, these representations, or by factoring the problem into these modules), because when one does, there are compelling benefits.*

Benefits might take the form of: improved system performance (or scaling), enhanced programmer productivity, greater system extensibility/generality, or the ability to provide new application-level capabilities that have not been possible before.

Identifying useful problem structure often forms the central intellectual idea of a systems paper. As in other areas of computer graphics, elegant conceptual insights can be summarized in a few sentence

# From our lecture 1 reading…

## Good systems papers highlight key design decisions (and discuss alternatives to those decisions).

Given a set of requirements, a systems architect is usually faced with a variety of solution strategies. For example, a performance requirement could be solved through algorithmic innovation, through the design of new specialized hardware, or both (modifying an existing algorithm to better map to existing parallel hardware). Alternatively, the path to better performance might best go through narrowing system scope to a smaller domain of tasks. A productivity goal might be achieved through the design of new programming abstractions, which might be realized as a new domain specific language, or via a library implemented in an existing system.

As a result, a systems paper author must identify the key choices made in architecting their system, and elaborate on the rationale for these design decisions. Doing so typically involves discussing the palette of potential alternatives, and providing an argument for why the choices made are a preferred way to meet the stated requirements and design goals. It is not sufficient to merely describe the path that was taken without saying why was deemed a good one.

# From our lecture 1 reading…

## The evaluation: were the key design decisions responsible for meeting the stated requirements and goals?

If a paper clearly describes a system's goals and constraints, as well as articulates key system design decisions, then the strategy for evaluating the system is to provide evidence that the described decisions were responsible for meeting the stated goals.

Particularly when a system's evaluation focuses on performance, it is tempting to compare the proposed system's end-to-end performance against that of competing alternative systems. While such an evaluation demonstrates that performance goals were met, it is equally (and sometimes more) important to conduct experiments that specifically assess the benefit of key optimizations and design decisions. Evaluation of why success was achieved is necessary to verify that the central claims of the paper are sound. Failing to perform this evaluation leaves open the possibility that the success of the system is due to other factors. (e.g., high-quality software engineering), than the proposed key ideas.

# One more trend: problem solving as an act of program generation (grounding plans in worlds using programs)

# Agent planning in virtual worlds

- Agents in a virtual world have an action space
  - Example at right is action space from VirtualHome

- Given world state not all actions are permissible
  - Grab <arg1> is not possible if agent is not near <arg1>
  - Open <arg1> is not possible if arg1 can't be opened

**"Relax on sofa"**

```
[WALK] ⟨living_room⟩(1)
[WALK] ⟨television⟩(1)
[FIND] ⟨television⟩(1)
[SWITCHON] ⟨television⟩(1)
[FIND] ⟨sofa⟩(1)
[SIT] ⟨sofa⟩(1)
[TURNTO] ⟨television⟩(1)
[WATCH] ⟨television⟩(1)
```

Task: Browse internet
Step 1: Walk to home office
Step 2: Walk to chair
Step 3: Find chair
Step 4: Sit on chair
Step 5: Find computer
Step 6: Switch on computer
Step 7: Turn to computer
Step 8: Look at computer
Step 9: Find keyboard
Step 10: Type on keyboard

🤣

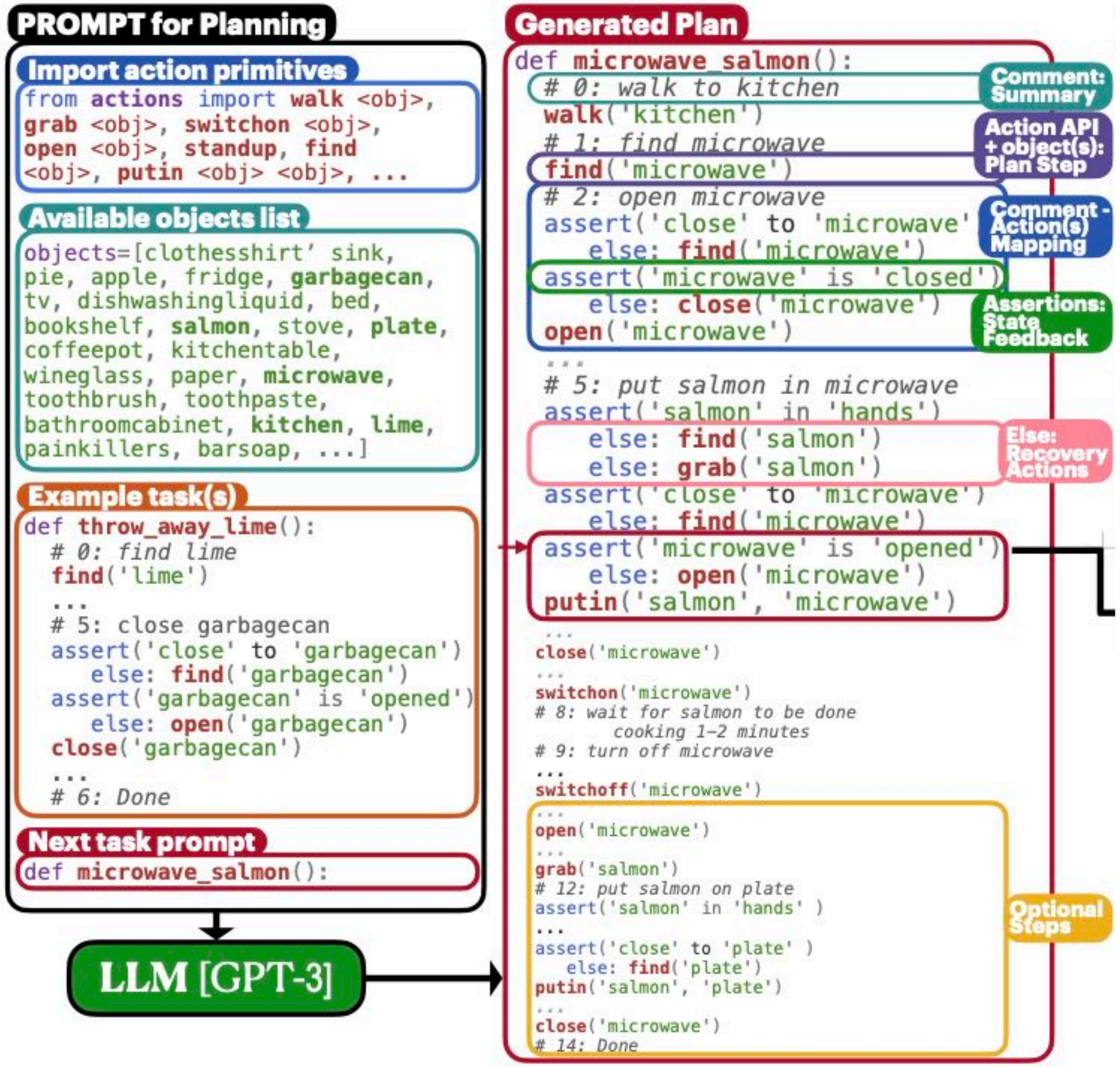
Task: Vacuum
Step 1: Walk to home office
Step 2: Walk to computer
Step 3: Find vacuum cleaner
Step 4: Switch on vacuum cleaner
Step 5: Switch off vacuum cleaner

**Atomic Action in VirtualHome Syntax**

```
[CLOSE] ⟨arg1⟩(1)
[CUT] ⟨arg1⟩(1)
[DRINK] ⟨arg1⟩(1)
[DROP] ⟨arg1⟩(1)
[EAT] ⟨arg1⟩(1)
[FIND] ⟨arg1⟩(1)
[GRAB] ⟨arg1⟩(1)
[GREET] ⟨arg1⟩(1)
[LIE] ⟨arg1⟩(1)
[LOOKAT] ⟨arg1⟩(1)
[MOVE] ⟨arg1⟩(1)
[OPEN] ⟨arg1⟩(1)
[PLUGIN] ⟨arg1⟩(1)
[PLUGOUT] ⟨arg1⟩(1)
[POINTAT] ⟨arg1⟩(1)
[POUR] ⟨arg1⟩(1) ⟨arg2⟩(1)
[PULL] ⟨arg1⟩(1)
[PUSH] ⟨arg1⟩(1)
[PUTBACK] ⟨arg1⟩(1) ⟨arg2⟩(1)
[PUTIN] ⟨arg1⟩(1) ⟨arg2⟩(1)
[PUTOBJBACK] ⟨arg1⟩(1)
[PUTOFF] ⟨arg1⟩(1)
[PUTON] ⟨arg1⟩(1)
[READ] ⟨arg1⟩(1)
[RELEASE]
[RINSE] ⟨arg1⟩(1)
[RUN] ⟨arg1⟩(1)
[SCRUB] ⟨arg1⟩(1)
[SIT] ⟨arg1⟩(1)
[SLEEP]
[SQUEEZE] ⟨arg1⟩(1)
[STANDUP]
[SWITCHOFF] ⟨arg1⟩(1)
[SWITCHON] ⟨arg1⟩(1)
[TOUCH] ⟨arg1⟩(1)
[TURNTO] ⟨arg1⟩(1)
[TYPE] ⟨arg1⟩(1)
[WAKEUP]
[WALK] ⟨arg1⟩(1)
[WASH] ⟨arg1⟩(1)
[WATCH] ⟨arg1⟩(1)
[WIPE] ⟨arg1⟩(1)
```

# Plans as valid programs

"Microwave salmon":

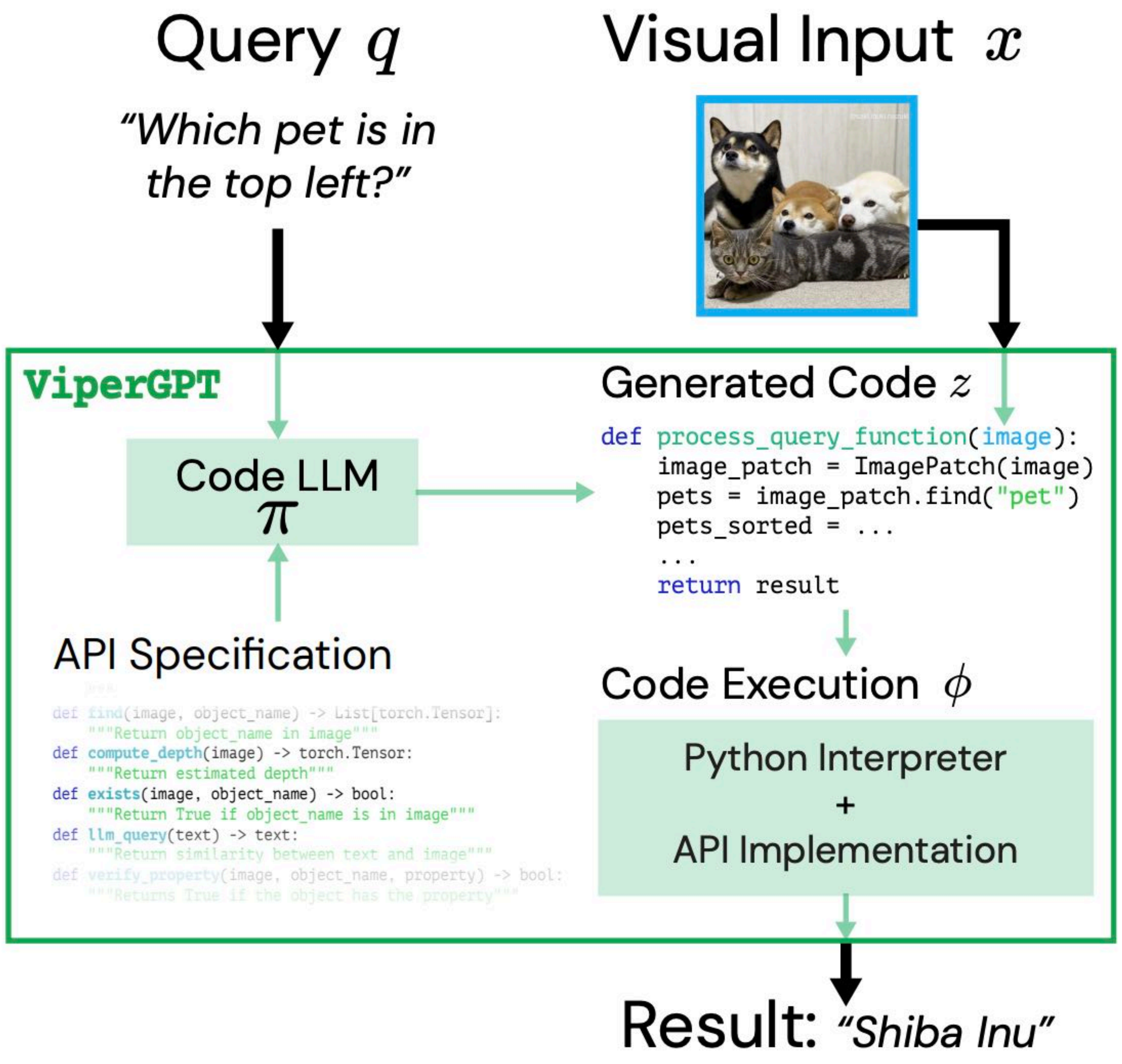



## Example: ProgPrompt [Sing et al. ICRA 23]

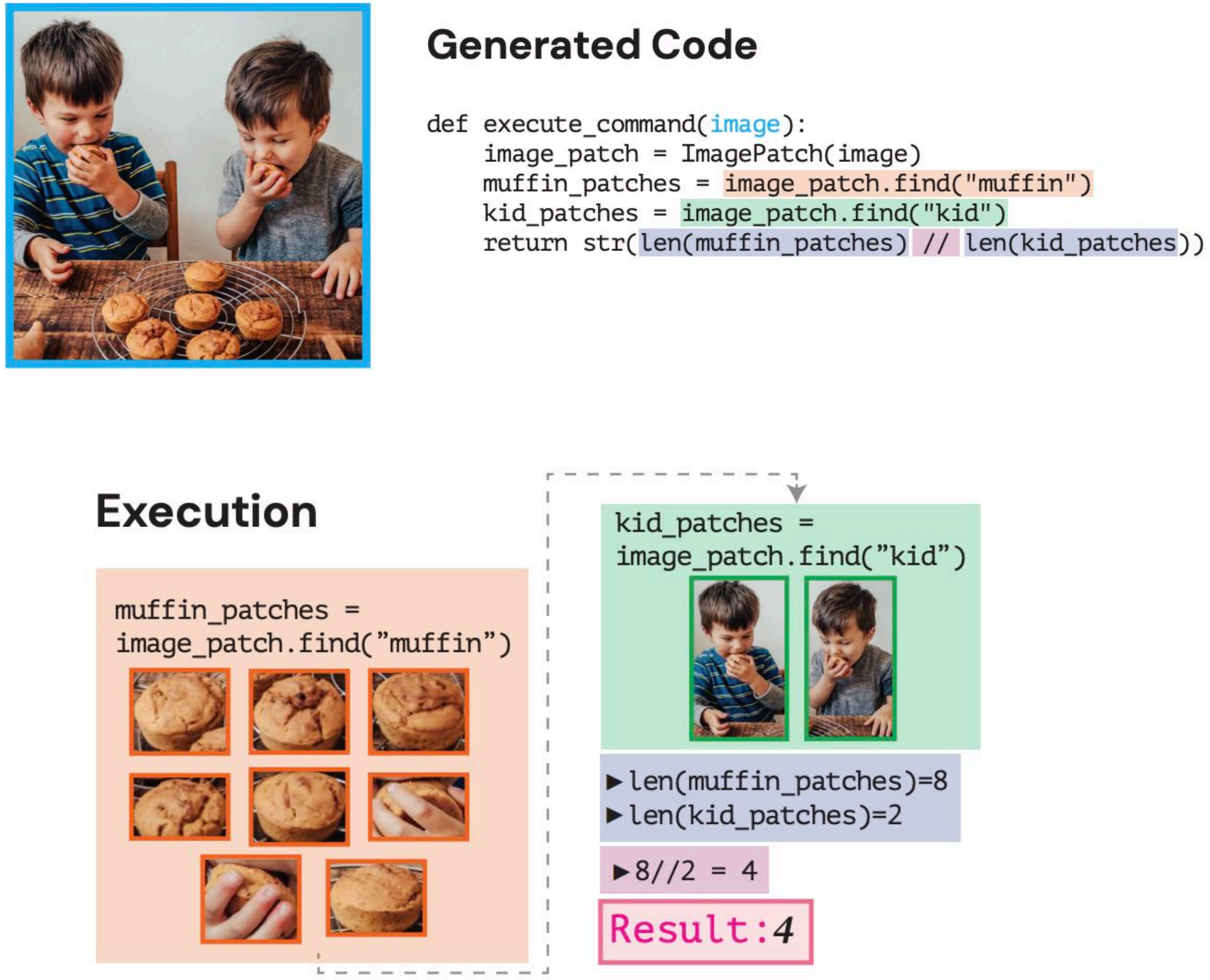
## Key ideas:

- LM as a code generator: a plan is a valid python program with access to subroutines (action space defined by subroutines)
- World state is given by a list of available objects
- Through conditional logic, plans can have grounded recovery policies:
  - If condition is not true, do X

# Another example: ViperGPT

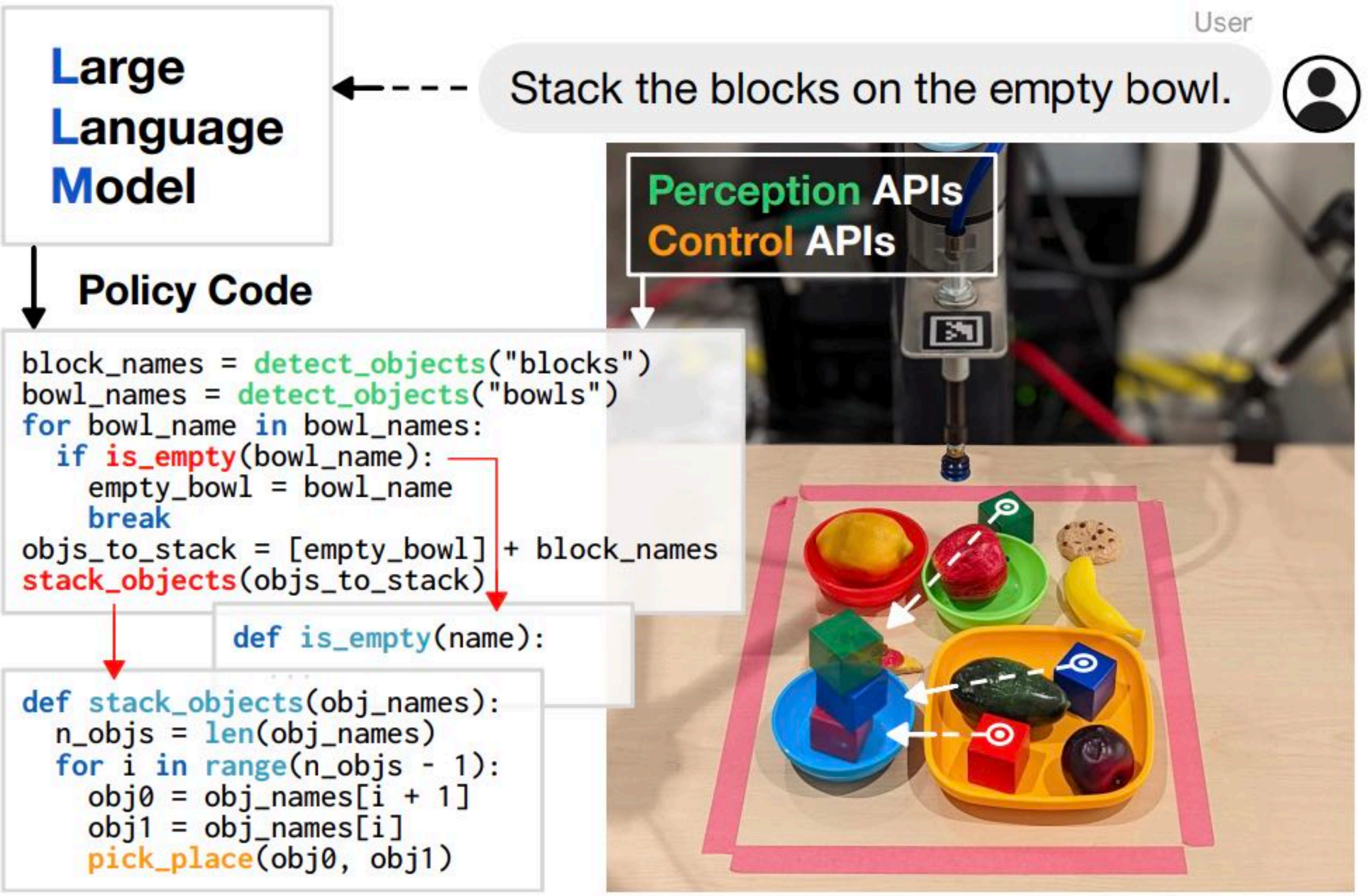



*Question: How many muffins can each kid have for it to be fair?*



# Another example

"Stack the blocks on the empty bowl":




**Example: Code as Policies [Liang et al. 2022]**

**Key ideas:**

- **Program can dynamically query for world state**
- **Simple example: call detect objects function**
- **More interesting: point camera at this location and take picture, etc.**

# Voyager

- Task: do things in Minecraft: make new things, fight Zombies, etc.
- Big ideas:
  - Use LM to generate plans as programs
  - Use LM to repair bad programs: (reflection)
    - Run programs to see if they work:
    - Get compiler feedback and game state feedback
    - Use LM to repair program given feedback


  - Use LM to propose new tasks (AI makes the curriculum)


  - Hierarchical skill library: new tasks get asked to skill library for use as future subroutines
    - As agent develops: increasing granularity of actions in action space