

# Tool Use & Agents

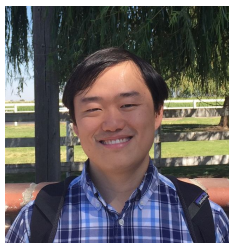
CS 159 (Spring 2024): LLMs for Reasoning

# Administrative

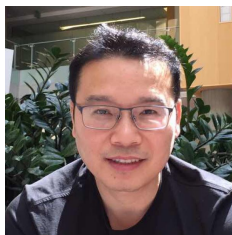
Join Piazza!

Student Presentation Sign-up (see link in Piazza)

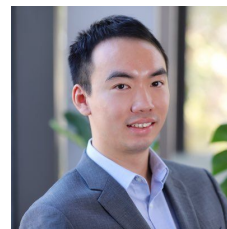
- We assigned presenters for next Tuesday
- More assignments will come early next week



Yuandong Tian  
4/18/2024



Denny Zhou  
4/23/2024



Kaiyu Yang  
5/2/2024

# Definitions (some are approximate)

Instruction Prompting: Giving detailed instructions in the input prompt.

In-Context Learning: Giving examples of desired behavior in the input prompt

Tool Use: LLM outputs calls to external APIs

Agents: capable of autonomous actions to achieve complex tasks

LLM Agent: given language description of task, solve task

- Almost always using tools
- Often designed via Instruction Prompting & In-Context Learning
- Can be compositions of multiple sub-agents

# Why use tools?

LLMs have trouble doing arbitrary reasoning  
(e.g., precise mathematics)

LLMs may not have the requisite knowledge for your domain  
(e.g., current events)

LLM Agents need to interact with its environment



You

At what point does  $2^x$  become larger than  $x^3$ ? You can compute both functions numerically and compare the values.

ChatGPT (GPT4)

LLMs are not precise calculators!

Using a tool (e.g., Python eval)  
can enable LLMs to give precise  
answers.

Given the nature of exponential and polynomial functions, the crossover will happen at a point where the rapid growth of the exponential function  $2^x$  overtakes the cubic growth of  $x^3$ , which isn't at the very start of the graph. Let me correct this by looking at a different method or adjusting the range and method of calculation to find the exact point of transition. [↗]

✓ Finished analyzing ▾

Always expand output? ☐

python

Copy code

```
# Correcting the approach: Using a more precise method to find the exact crossover point
from scipy.optimize import fsolve

# Define a function that represents the difference between the two functions
def difference(x):
    return 2**x - x**3

# Find the root of the difference function, which indicates where the two functions are equal
# Starting with an initial guess that's in the ballpark of where we might expect the crossover
x_crossover = fsolve(difference, 10) # Starting guess is 10 based on the nature of the functions

x_crossover
```

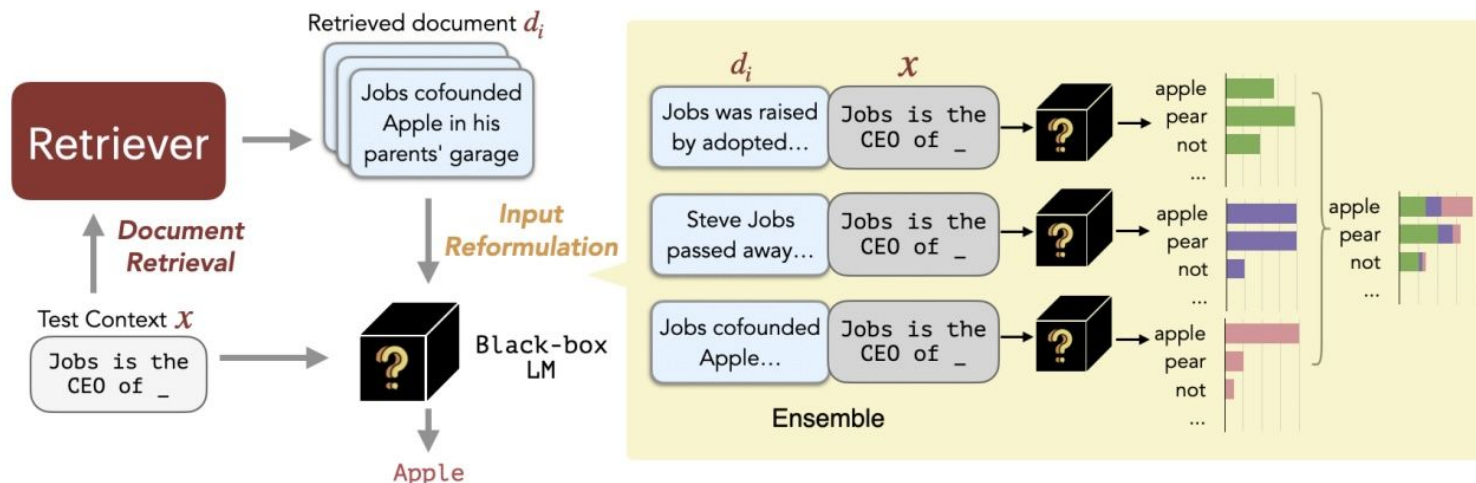
Result

```
array([9.93953514])
```

The precise point at which  $2^x$  becomes larger than  $x^3$  is at  $x \approx 9.94$ . This is where the exponential growth rate of  $2^x$  surpasses the cubic growth rate of  $x^3$ . [↗]

# Aside: Retrieval Augmented Generation (RAG)

LLM generates results using external data store



<https://arxiv.org/abs/2005.11401> (predates modern LLMs)

<https://arxiv.org/abs/2301.12652> & <https://arxiv.org/abs/2212.01349> (RAG for LLMs)

# RAG Example with Simple Prompt

## Retrieved Document

**Knowledge:** Arctic Ocean. Although over half of Europe's original forests disappeared through the centuries of deforestation, Europe still has over one quarter of its land area as forest, such as the broadleaf and mixed forests, taiga of Scandinavia and Russia, mixed rainforests of the Caucasus and the Cork oak forests in the western Mediterranean. During recent times, deforestation has been slowed and many trees have been planted. However, in many cases monoculture plantations of conifers have replaced the original mixed natural forest, because these grow quicker. The plantations now cover vast areas of land, but offer poorer habitats for many European

**Question:** As of 2015, since 1990 forests have \_\_\_\_\_ in Europe and have \_\_\_\_\_ in Africa and the Americas.

A. "increased, increased" B. "increased, decreased" C. "decreased, increased" D. "decreased, decreased"

**Answer:** B

## Complete Prompt

Will cover how this idea applies to general tool-use & reasoning

# External APIs (Tools) can be very diverse!

Python scripts

Bash commands

Custom Libraries

Web browsing

Mathematica

UI Commands (mouse clicks)

Theorem Checkers

Robotic Commands

Simulations & Renderers

Etc...

Compilers



# Visual Reasoning (First Tool Use Example)

Suppose we had LLM with no direct image capabilities.



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

# ViperGPT

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



**ViperGPT**

<https://viper.cs.columbia.edu/>

## Generated Code

```
def execute_command(image):  
    image_patch = ImagePatch(image)  
    muffin_patches = image_patch.find("muffin")  
    kid_patches = image_patch.find("kid")  
    return str(len(muffin_patches) // len(kid_patches))
```

"tools"



## Execution

```
muffin_patches =  
image_patch.find("muffin")
```



```
kid_patches =  
image_patch.find("kid")
```



```
► len(muffin_patches)=8  
► len(kid_patches)=2
```

```
► 8//2 = 4
```

**Result:4**



You



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



ChatGPT

In the image, I see a wire cooling rack with muffins on it, and there are two children who appear to be about to eat them. To determine how many muffins each child can have to ensure it's fair, we simply need to divide the total number of muffins by two.


There are six muffins visible in the image, so dividing them equally, each child can have three muffins. That would make it fair, assuming all the muffins are the same size and none has been claimed already!

## Claude 3 Sonnet



YY How many muffins can each kid have for it to be fair?

Based on the image, there appear to be 8 muffins or baked goods on the tray in front of the two children. To divide them fairly so each child gets an equal amount, each child could have 4 muffins.

 Copy  Retry  



Claude can make mistakes. Please double-check responses.



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



This image has  
been removed



Sorry, I can't help with images of people yet.

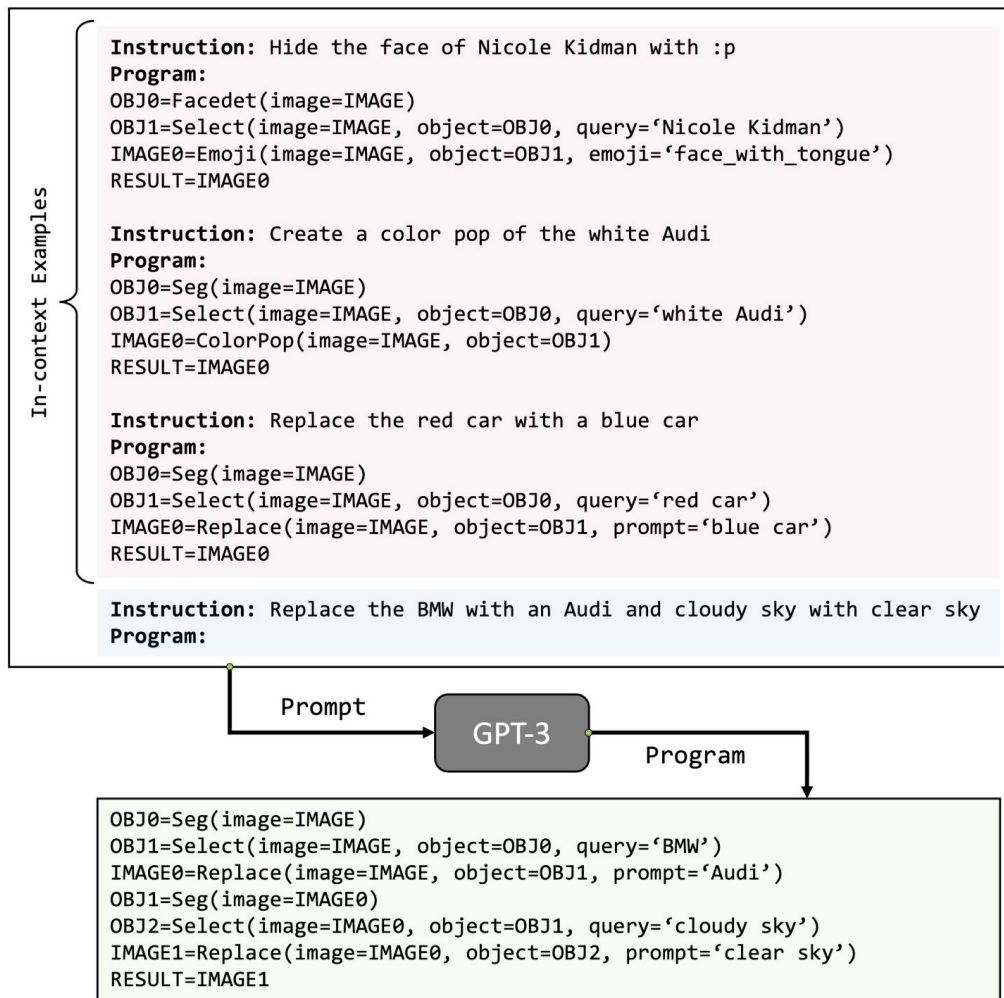
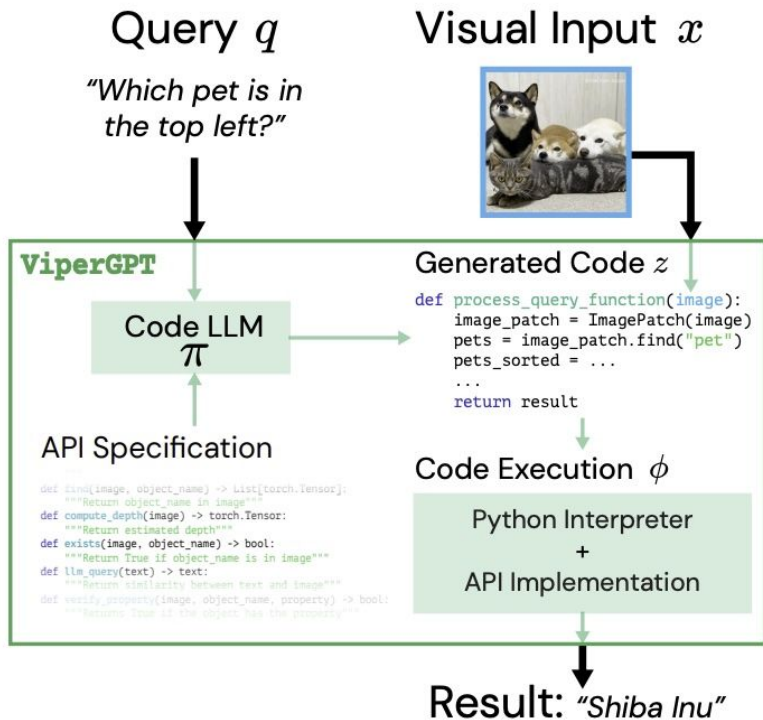
# Visual Programming (CVPR 2023 Best Paper)

<https://prior.allenai.org/projects/visprog>





Both ViperGPT & Visual Programming use a combination of API specification and in-context examples in prompt.



# Skill Library

Combinations of atomic actions (can be templated)

Common examples in embodied agents:

- Robotics
- Video games
- Complex designs

Will be covered in class!





# Example 1: Skills Implemented as Action Sequences

(Voyager, covered in student presentations)

Rather than having to generate entire sequences of actions, store as skill in a skill library.



# Retrieve from Skill Library RAG-style

## Program Generated by GPT-4

```
async function combatZombie(bot) {  
  // Equip a weapon  
  const sword =  
  bot.inventory.findInventoryItem(  
    mcData.itemsByName[  
      "stone_sword"  
    ].id  
  );  
  if (sword) {  
    await bot.equip(sword, "hand");  
  } else {  
    await craftStoneSword(bot);  
    ...  
  }  
  // Craft and equip a shield  
  ...  
  // Recover hunger  
  ...  
  // Look for and combat a zombie  
  ...  
}
```

GPT-3.5

## Program Description

```
async function combatZombie(bot) {  
  // The function is about  
  equipping a stone sword to combat  
  a zombie. If a stone sword is not  
  found, it will craft one.  
  Additionally, it crafts and equips  
  a shield for added protection.  
  Afterwards, it proceeds to cook  
  sticks in order to restore hunger.  
  Once hunger is replenished, it  
  actively searches for a zombie and  
  engages in combat with it.  
}
```

Embedding

Key

Value

Add

## Skill Library

- Mine Wood Log
- Make Crafting Table
- Craft Wooden Pickaxe
- Craft Stone Sword
- Make Furnace
- ...
- Combat Cow
- Cook Steak
- Craft Iron Axe
- Combat Zombie

## Task: Craft Iron Pickaxe

How to craft an iron pickaxe in Minecraft?

GPT-3.5

To craft an iron pickaxe, you need to 3 iron ingots and 2 sticks. Once you have gathered the materials, ...

Environment Feedback

Embedding

Query

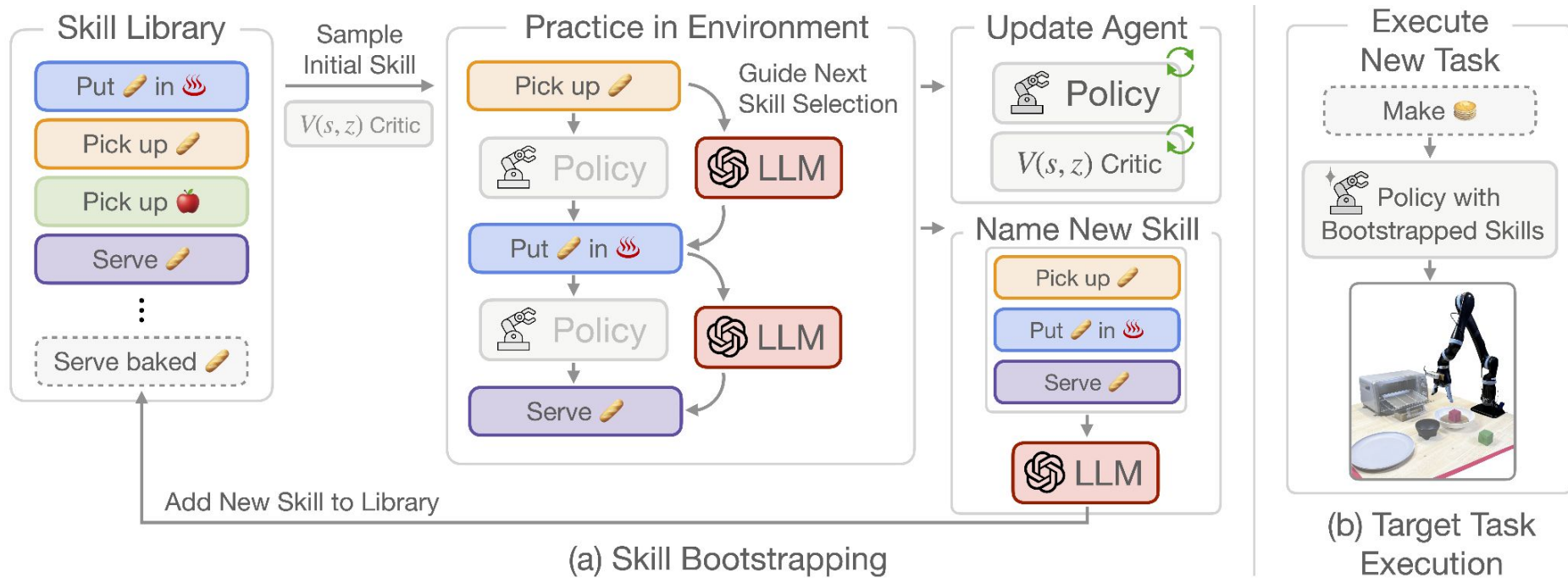
Skill Library

Retrieve

## Top-5 Relevant Skills

- Smelt Iron Ingot
- Craft Stick
- Make Crafting Table
- Make Furnace
- Craft Wooden Pickaxe

## Example 2: Skills implemented by RL Policy (BOSS)

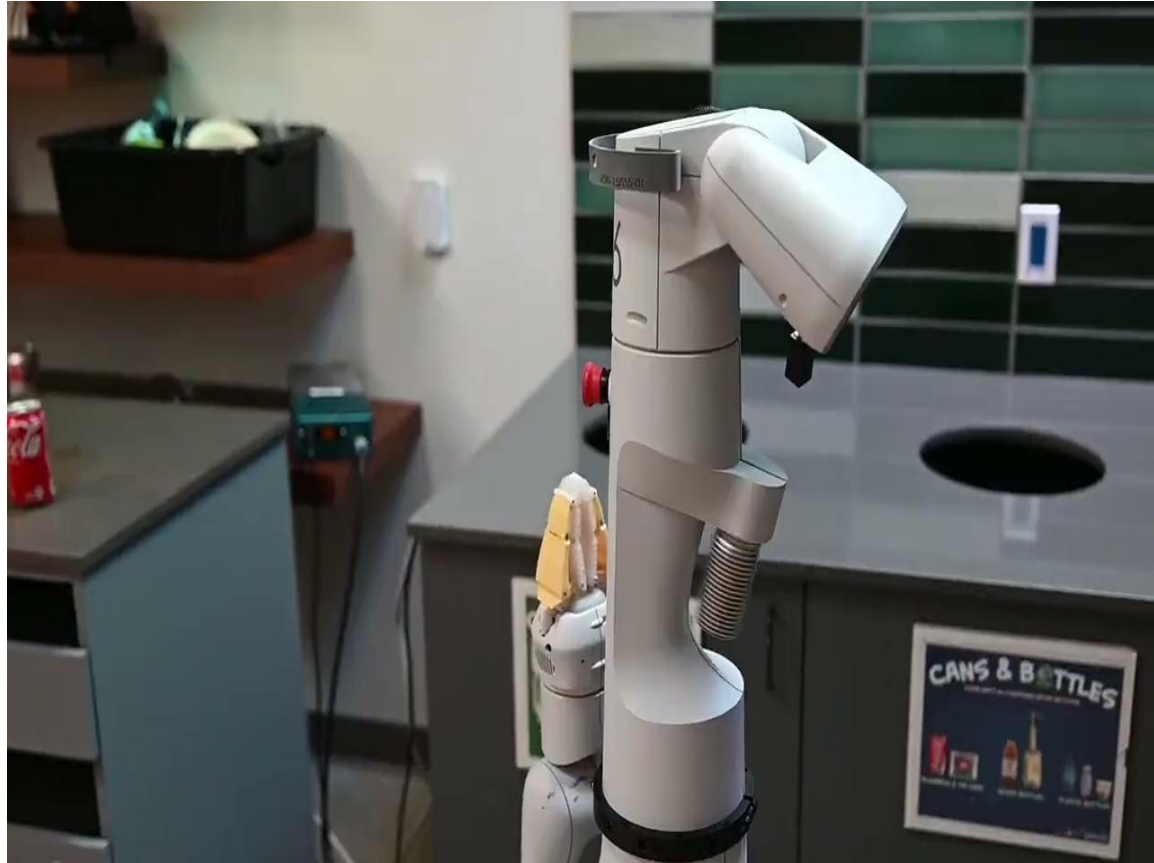


# Earlier Approach: SayCan

(covered in student presentations)

Each LLM output needs to be implementable by a control policy.

<https://say-can.github.io/>



# SayCan Summary

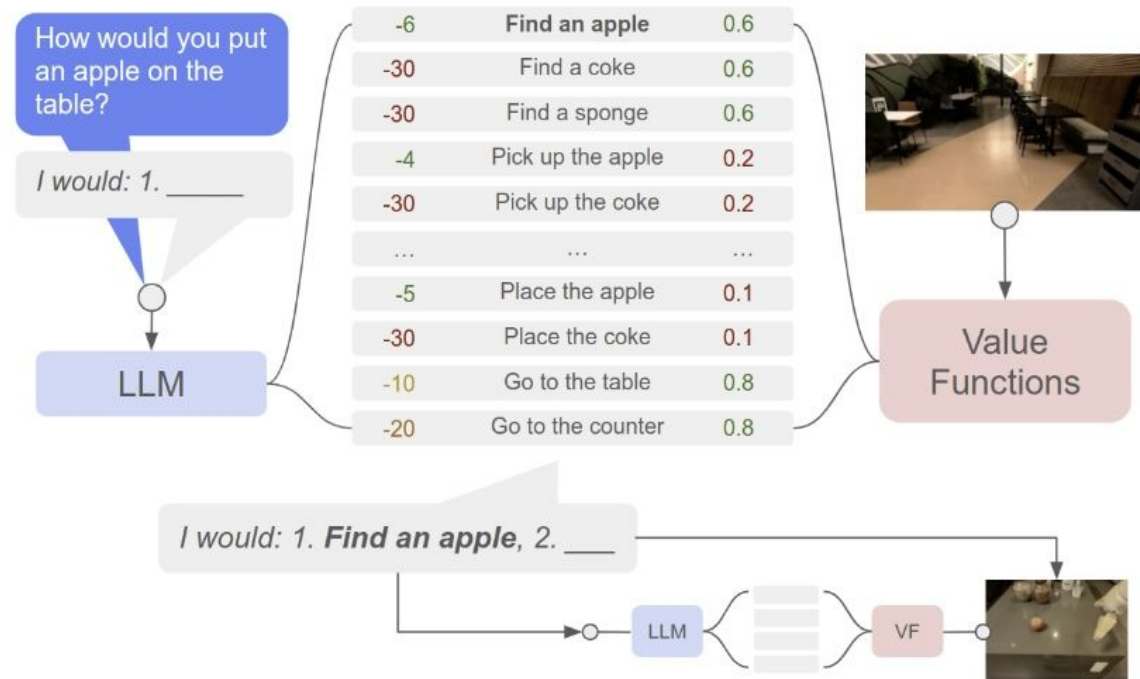
SayCan uses:

- LLM generates (a few) common sense plans
- Pretrained value function picks one
- Depends on skill of underlying control policy

Instruction Relevance with LLMs

Combined

Task Affordances with Value Functions



Will go into more depth in student presentation!

<https://say-can.github.io/>

# Use of Verifiers

Does code compile?

Does code pass unit tests?

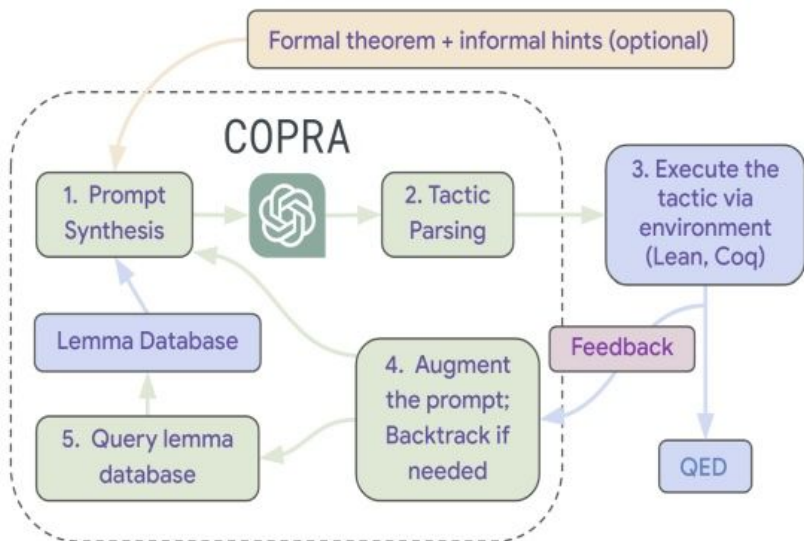
Does proof pass proof checker?

If only goal is to generate verified solution, often think of Verifier as the entire “environment”.

Otherwise, Verifier is part of broader tool set.



# Example for Theorem Proving

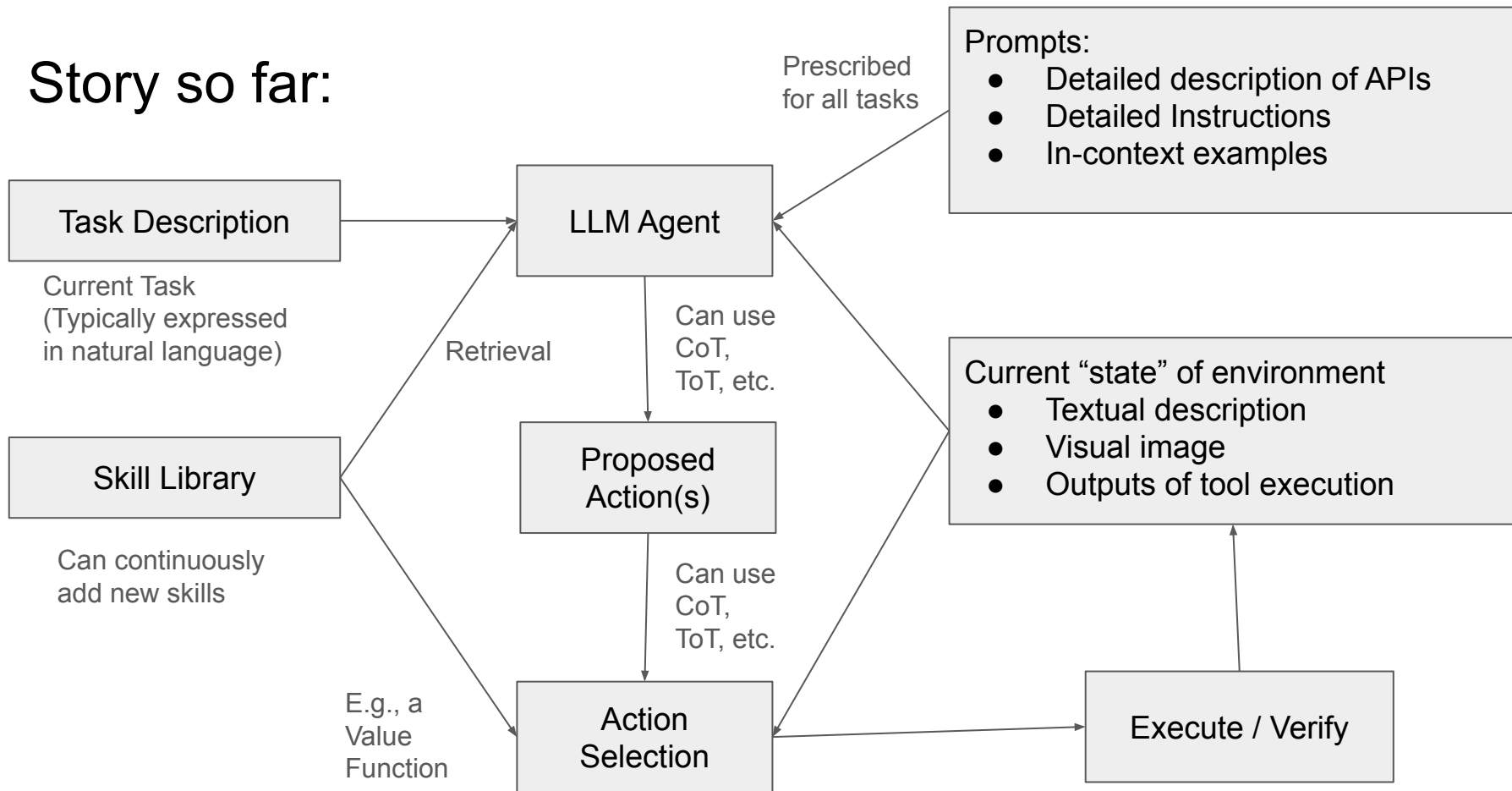


You are a proficient formal theorem-proving agent in Lean 3. You are tasked with predicting the next proof step given the current proof state, which is described in the following format:

- All the goals are described under the ``[GOALS]`` keyword. Each goal following ``[GOALS]`` is described under the keyword ``[GOAL] i``, where `i` is a positive integer. For example, ``[GOAL] 1``, ``[GOAL] 2``, etc.
- Within each ``[GOAL] i`` keyword, the goal is described as a human-readable serialized version of the proof state as shown while running the `lean` command. Each goal may be accompanied by hypotheses, which are described under the keyword ``[HYPOTHESES] i``. Each hypothesis following ``[HYPOTHESES] i`` starts with the prefix ``[HYPOTHESIS]``. Apart from goals and hypotheses, the OPTIONAL keywords ``[DEFINITIONS] i`` and ``[THEOREMS] i`` may appear, which respectively describe the relevant definitions of symbols and terms used in that goal and some potentially useful theorems or lemmas which might help in proving the goal. Each definition within ``[DEFINITIONS]`` starts with the prefix ``[DEFINITION]``. Similarly, each theorem/lemma within ``[THEOREMS]`` starts with the prefix ``[THEOREM]``. If you choose to use a theorem described in ``[THEOREMS] i``, be SURE that it applies and is useful for proving the goal.
- The ``[STEPS]`` keyword is used to describe the proof-steps which were used to obtain the current proof state from the original theorem. Each proof step starts with the prefix ``[STEP]`` and is a valid Lean tactic. For example, ``[STEPS][STEP]rw h1 at h2,[STEP]{linarith},``.
- Sometimes the ``[INCORRECT STEPS]`` keyword will appear, which describes proof-steps which should NOT be generated. For example, ``[INCORRECT STEPS][STEP]apply h1,[STEP]rw +h1. **DO NOT** generate these `[INCORRECT STEPS]` again, as they are failed proof steps which have already been tried. Re-generating such proof steps will cause backtracking and early termination of your proof search.`
- There is also an optional ``[LAST STEP]`` keyword which describes the proof-step generated last time. If the proof-step was incorrect, then it is also followed by an error message from Lean 3 environment described with the ``[ERROR MESSAGE]`` keyword. For example, ``[LAST STEP]linarith,\n[ERROR MESSAGE]linarith failed to find a contradiction\nstate:\nx y : ℝ,\nh1 : x = 3 - 2 * y,\nh2 : 2 * x - y = 1\n- false``. You can use the error message as guidance in predicting a correct proof-step. Do not generate tactics which you believe will result in the same error. If the proof-step was correct then it is followed by the keyword ``[SUCCESS]``. For example, ``[LAST STEP]linarith,[SUCCESS]``. Do NOT generate the last proof-step again if it was NOT successful, this will also cause early termination of your proof search.
- Sometimes there can be errors in the format of the generated response. This is reported using the keyword ``[ERROR]`` followed by the error message. For example, ``[ERROR]\nInvalid response:\n'Great! The proof is complete.', \nStopping Reason: 'stop'.\nPlease respond only in the format specified.[END]``. This means that the response generated by you was not in the specified format.

Start your response with ``[RUN TACTIC]`` followed by the tactic which will help in proving the current proof state, and then ``[END]``. For example, ``[RUN TACTIC] induction c, [END]``. Do NOT finish the proof in one shot ending with ``end``, which signifies the end of a Lean 3 proof. Generate exactly ONE proof-step. Multiple proof steps are more error prone, because you will not get a chance to see intermediate proof state descriptions.....

# Story so far:





# Important Discussion: Tradeoffs w.r.t. Base Model

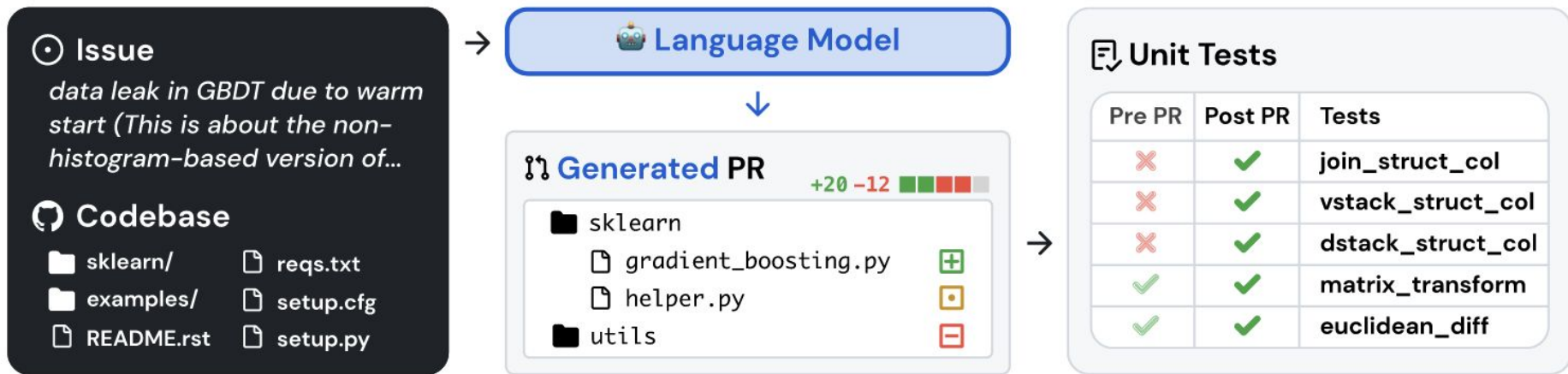
LLMs have many types of capabilities (non-exhaustive list):

- Common-sense knowledge
  - Can include other modalities such as image
- Common-sense Logic
- Mathematical reasoning (math)
  - Precise reasoning is still unreliable
- Generating code (seems to be prioritized by many LLMs)
  - (Executing code to get answer is an example of tool use)

**What tasks needs tools, skills, or careful prompt engineering will evolve as base models get stronger.**

(Note: stronger models are slower & more expensive.)

# Case Study: SWE-Agent & SWE-Bench



SWE-Bench is an evaluation framework with real-world GitHub issues.

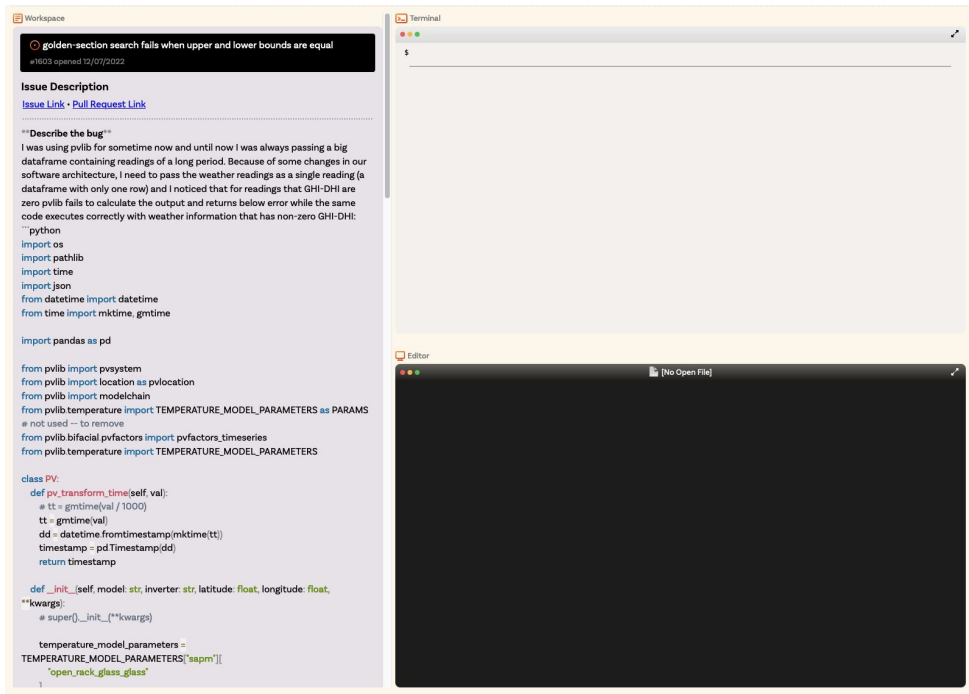
Supports submitting patch, and running unit tests



<https://www.swebench.com/>

# SWE-Agent Execution

(<https://swe-agent.com/demo>)



## Thought & Action

Our reproduction script confirms the reported issue: Max & Min are not being converted to R. Let's search for files related to R code generation.

```
$ search_dir "rcode"
```

## Observation

Found 16 matches for "rcode":  
reproduce\_issue.py (3 matches)  
sympy/printing/lambdarepr.py (1 matches)  
sympy/printing/rcode.py (12 matches)

## Thought & Action

The responsible file is likely rcode.py. We should open and inspect this file.

```
$ open sympy/printing/rcode.py
```

## Observation

```
[sympy/printing/rcode.py (412 lines)]
1: """The RCodePrinter converts sympy ...
2: from sympy.core import S
...
20: # dictionary of sympy to C functions
21: known_functions = {
22:     "Abs": "abs",
```

## Thought & Action

We need to add entries Max/Min to known\_functions, mapping them to corresponding R functions, max/min.

```
$ edit 22:22
    "Abs": "abs",
    "Max": "max",
    "Min": "min",
end_of_edit
```

# SWE-Agent Prompt Templates

## **system\_template:** |-

SETTING: You are an autonomous programmer, and you're working directly in the command line with a special interface....

## **instance\_template:** |-

We're currently solving the following issue within our repository. Here's the issue text: ISSUE: {issue} ...

## **next\_step\_template:** |- ...

## **next\_step\_no\_output\_template:** |- ...

## **demonstration\_template:** |-

Here is a demonstration of how to correctly accomplish this task....

# SWE-Agent Prompt (Describe a Persona)

SETTING: You are an autonomous programmer, and you're working directly in the command line with a special interface.

The special interface consists of a file editor that shows you {WINDOW} lines of a file at a time.

In addition to typical bash commands, you can also use the following commands to help you navigate and edit files.

COMMANDS:  
{command\_docs}

# SWE-Agent Prompt (Yell at the model about indentation)

Please note that THE EDIT COMMAND REQUIRES PROPER INDENTATION.

If you'd like to add the line ' print(x)' you must fully write that out, with all those spaces before the code! Indentation is important and code that is not indented correctly will fail and require fixing before it can be run.

NOTE ABOUT THE EDIT COMMAND: Indentation really matters! When editing a file, make sure to insert appropriate indentation before each line!

# SWE-Agent Prompt (One-Shot Learning)

## RESPONSE FORMAT:

You need to format your output using two fields; discussion and command.

Your output should always include `_one_` discussion and `_one_` command field EXACTLY as in the following example:

## DISCUSSION

First I'll start by using `ls` to see what files are in the current directory. Then maybe we can look at some relevant files to see what they look like.

...

`ls -a`

...

# SWE-Agent Prompt (Closed-Loop Bugs)

You should only include a *\*SINGLE\** command in the command section and then wait for a response from the shell before continuing with more discussion and commands. Everything you include in the DISCUSSION section will be saved for future reference.

If you'd like to issue two commands at once, PLEASE DO NOT DO THAT! Please instead first submit just the first command, and then after receiving a response you'll be able to issue the second command.

Remember, YOU CAN ONLY ENTER ONE COMMAND AT A TIME. You should always wait for feedback after every command.



# SWE-Agent Prompt (Closed-Loop Bugs)

(Inside Instance Template)

Now, you're going to solve this issue on your own. Your terminal session has started and you're in the repository's root directory. You can use any bash commands or the special interface to help you. Edit all the files you need to and run any checks or tests that you want.

Remember, **YOU CAN ONLY ENTER ONE COMMAND AT A TIME**. You should always wait for feedback after every command.

When you're satisfied with all of the changes you've made, you can submit your changes to the code base by simply running the submit command.

Note however that you cannot use any interactive session commands (e.g. python, vim) in this environment, but you can write scripts and run them. E.g. you can write a python script and then run it with `python <script_name>.py``.

# SWE-Agent Prompt (Closed-Loop Bugs)

(Inside Instance Template)

If you run a command and it doesn't work, try running a different command. A command that did not work once will not work the second time unless you modify it!

Always make sure to look at the currently open file and the current working directory (which appears right after the currently open file). The currently open file might be in a different directory than the working directory! Note that some commands, such as 'create', open files, so they might change the current open file.

When editing files, it is easy to accidentally specify a wrong line number or to write code with incorrect indentation. Always check the code after you issue an edit to make sure that it reflects what you wanted to accomplish. If it didn't, issue another command to fix it.

# Simple Colab Example

<https://colab.research.google.com/drive/1vCHvNfnXxmAb3FCrNKryus6kUZHihRCm?usp=sharing>

# L3GO: Chain of 3D Thoughts

<https://arxiv.org/abs/2402.09052>

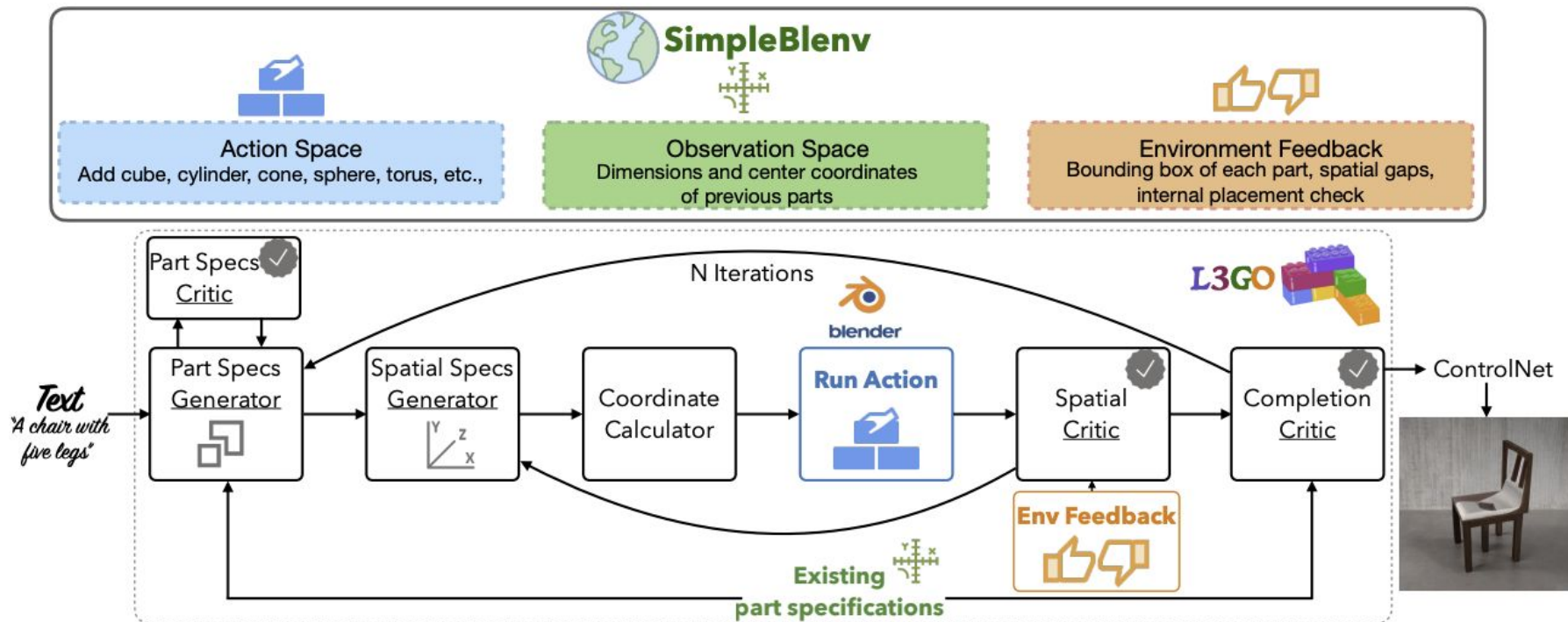
Covered in student presentations!

Prompt: "a chair with five legs"



DALL-E 3

L3GO



# L3GO: Chain of 3D Thoughts

<https://arxiv.org/abs/2402.09052>

## Parts Critic Prompt

You are a helpful assistant to complete any 3D modeling task specified by me.

You are trying to create a 3D model of {object\_name}.

The base part is {base\_part\_name}, and the dimensions of the base part is {part\_size}.

Here are the requirements for the dimensions of the base part:

1. The width, depth, and height dimensions correspond to the x, y, and z-axis respectively in the 3D space.
2. When dealing with an object that moves, such as a vehicle, ensure that the base part is extended in the y-direction(depth).

After explaining your reasoning, please respond with a simple 'yes' or 'no' to the following questions:

Question 1: Does the dimensions of the base part meet the above requirements?

RESPONSE FORMAT:

Reasoning: (Clearly state the width(x-axis), depth(y-axis), and height(z-axis). Then, compare these values and form a judgment.)

Answer 1: (yes or no)

# L3GO: Chain of 3D Thoughts

<https://arxiv.org/abs/2402.09052>

## Spatial Generator Prompt Template

You are a helpful assistant who has excellent spatial reasoning skills.

I'll provide you with a list of object parts that you have already created, along with a new object part that you are going to make.

Your first task is to determine which of the existing parts can serve as a base for attaching the new object part.

Your second task is to write requirement(s) for the spatial relations that the two object parts need to fulfill.

# Example 1:

object: table

new object part: front-left leg

Object parts you have already created:

tabletop, drawer

Base object part:

tabletop.

Spatial requirements:

- the front-left leg is directly attached to the bottom side of tabletop and positioned underneath it.

You should only respond in the format as described below:

RESPONSE FORMAT:

Base object part: ...

Spatial requirements: ...

# Recall: Tradeoffs w.r.t. Base Model

Agents have EXTREMELY complex prompts.

For LLMs with large context windows, can enter all instructions into prompt

- Even so, unclear if LLM can follow instructions reliably
- For code it seems more reasonable (a lot of training data)

Often need to design multiple instructions depending on situation:

- SWE-Agent has (at least) 5
- L3GO

**How to automatically design? How to debug Agent design?**

# Agent Design Criteria

Debuggability of individual sub-tasks

Compute & Inference speed

- Large models are expensive and slow (GPT4, Gemini Ultra, Claude Opus)
- Smaller models might suffice on certain sub-tasks

Complex interactions in prompt design

Complex interactions with skill library design

Etc...



# Research Ideas

Designing Agent Pipelines (see also [DSPy](#))

Designing skill libraries & other abstractions

Understanding how to criticize and inspect imperfect solutions  
(many tasks don't have obvious right answer)


Search algorithms for complex tasks


Incorporating human in the loop feedback

Multi-agent interactions


# Research Ideas Continued

Precise reasoning  
(often spatial or  
mathematical)

 You



Where is Gate C12?

 ChatGPT

Gate C12 is located to the left in the image, as indicated by the signage in the airport terminal. It's in the direction where it says "Terminals Concessions Baggage Claim Gates C1 - C19," so passengers looking for Gate C12 would follow that sign to the left.