Project

LLMs for Robotics Planning

Angelo Moroncelli
Loris Roveda

LEON Group

30 aprile 2025

0. LLMs for Robotics Planning

How to use LLMs in practice for a robotics application?

0. LLMs for Robotics Planning

How to use LLMs in practice for a robotics application?

Fine-Tuning (action generators)

- Highly accurate on specific tasks
- Domain specific
- VLA models with LLM backbone

How to use LLMs in practice for a robotics application?

Fine-Tuning (action generators)

- Highly accurate on specific tasks
- Domain specific
- VLA models with LLM backbone

Zero-Shot (planners)

- No retraining
- Easier deployment
- Ability to generalize

1.1. Fine tuning

Challenges with LLMs solvable with fine-tuning:

- **High computational cost:** Billions of parameters require significant resources for training and deployment.

1.1. Fine tuning

Challenges with LLMs solvable with fine-tuning:

- **High computational cost:** Billions of parameters require significant resources for training and deployment.
- **Memory constraints:** Large model size makes it hard to deploy on devices with limited resources.

1.1. Fine tuning

Challenges with LLMs solvable with fine-tuning:

- **High computational cost:** Billions of parameters require significant resources for training and deployment.
- **Memory constraints:** Large model size makes it hard to deploy on devices with limited resources.
- **Adaptation to specific tasks:** Pretrained LLMs may need fine-tuning to perform well on specialized tasks.

1.1. Fine tuning

Traditional fine-tuning:

- Fine-tune all model parameters on a task-specific dataset.
  - Requires extensive computational resources and time.
  - Potential overfitting if the fine-tuning dataset is too small.
  - Full model fine-tuning can be a slow process, especially for new tasks or datasets.

1.1. Fine tuning

Traditional fine-tuning:

- Fine-tune all model parameters on a task-specific dataset.
  - Requires extensive computational resources and time.
  - Potential overfitting if the fine-tuning dataset is too small.
  - Full model fine-tuning can be a slow process, especially for new tasks or datasets.

- Keep frozen most of the layers of the model and fine-tune a small number of weights on a task-specific dataset.
  - How to select the layers to fine-tune?

1.1. Parameter-Efficient Fine-Tuning (PEFT)

Solution: Parameter-Efficient Fine-Tuning (PEFT)

1.1. Parameter-Efficient Fine-Tuning (PEFT)

Solution: Parameter-Efficient Fine-Tuning (PEFT)

- A strategy for fine-tuning that focuses on optimizing only a small subset of model parameters.

1.1. Parameter-Efficient Fine-Tuning (PEFT)

Solution: Parameter-Efficient Fine-Tuning (PEFT)

- A strategy for fine-tuning that focuses on optimizing only a small subset of model parameters.
- Minimizes resource consumption while maintaining model performance.

1.1. Parameter-Efficient Fine-Tuning (PEFT)

Solution: Parameter-Efficient Fine-Tuning (PEFT)

- A strategy for fine-tuning that focuses on optimizing only a small subset of model parameters.
- Minimizes resource consumption while maintaining model performance.
- **Lower memory usage:** Fine-tunes only a small number of parameters.

1.1. Parameter-Efficient Fine-Tuning (PEFT)

Solution: Parameter-Efficient Fine-Tuning (PEFT)

- A strategy for fine-tuning that focuses on optimizing only a small subset of model parameters.
- Minimizes resource consumption while maintaining model performance.
- **Lower memory usage:** Fine-tunes only a small number of parameters.
- **Faster adaptation:** Reduced fine-tuning time.

1.1. Parameter-Efficient Fine-Tuning (PEFT)

Solution: Parameter-Efficient Fine-Tuning (PEFT)

- A strategy for fine-tuning that focuses on optimizing only a small subset of model parameters.
- Minimizes resource consumption while maintaining model performance.
- **Lower memory usage:** Fine-tunes only a small number of parameters.
- **Faster adaptation:** Reduced fine-tuning time.
- **Preserves pretrained knowledge:** Only minor adjustments are made to the model.

1.1. Parameter-Efficient Fine-Tuning (PEFT): LoRA

The method we will use in practice is **Low-Rank Adaptation (LoRA)**:

- Introduces low-rank matrices into the model’s weight matrices.
- Fine-tunes just the low-rank components rather than the full weight matrices.
- **Advantage: Efficient** use of parameters and **less resource-intensive**.
- Fine-tunes the model by making **minimal changes to the pretrained weights**, allowing the model to retain its general-purpose knowledge from pretraining while adapting to task-specific requirements.

Credit: https://github.com/huggingface/peft

1.2 Language-Action Model via SyntheticData Generation for Robot Control

- LoRA fine-tuning of **specialized LLMs** to excel in-context learning from user input for action generation (i.e., reference pose) for specific real-world tasks;

1.2 Language-Action Model via SyntheticData Generation for Robot Control

- LoRA fine-tuning of **specialized LLMs** to excel in-context learning from user input for action generation (i.e., reference pose) for specific real-world tasks;
- A **Language-Action** model that relies on **synthetic data** generation and LoRA for LLM specialization;

1.2 Language-Action Model via SyntheticData Generation for Robot Control

- LoRA fine-tuning of **specialized LLMs** to excel in-context learning from user input for action generation (i.e., reference pose) for specific real-world tasks;
- A **Language-Action** model that relies on **synthetic data** generation and LoRA for LLM specialization;
- A **modular framework** for robotic task learning, integrating an LLM with a decoupled machine vision module (e.g., YOLO). The vision system processes data and feeds it to the LLM as contextual input;

1.2 Language-Action Model via SyntheticData Generation for Robot Control

- LoRA fine-tuning of **specialized LLMs** to excel in-context learning from user input for action generation (i.e., reference pose) for specific real-world tasks;
- A **Language-Action** model that relies on **synthetic data** generation and LoRA for LLM specialization;
- A **modular framework** for robotic task learning, integrating an LLM with a decoupled machine vision module (e.g., YOLO). The vision system processes data and feeds it to the LLM as contextual input;
- A LLM-based control system capable of **real-time inference**, tracking objects, and responding quickly to dynamic changes in the environment.

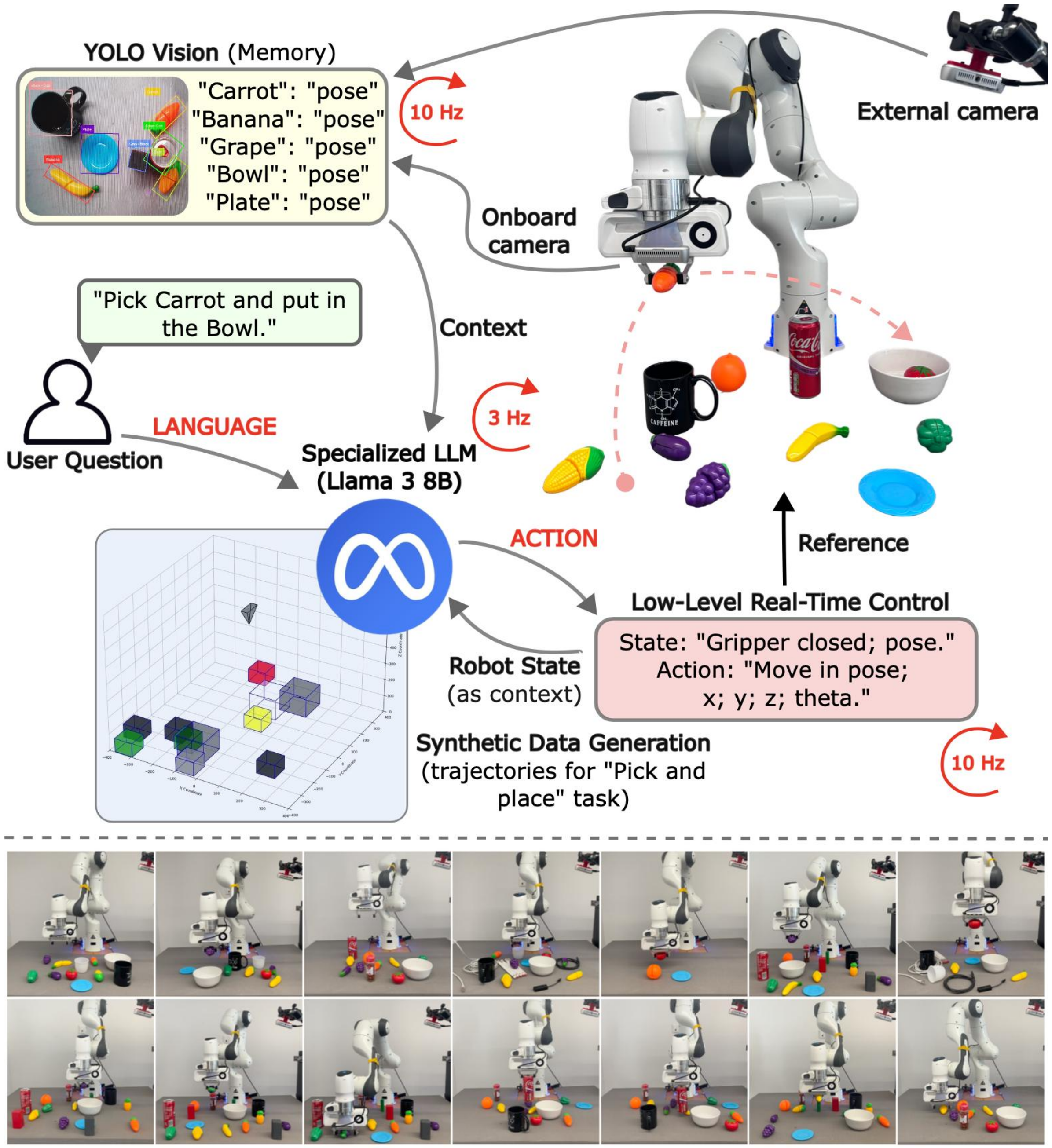
1.2. Language-Action Model via SyntheticData Generation for Robot Control

- Credit: *Maccarini et al., paper coming soon ...*

- **Problem:** Limited perception and in-context learning due to the ad hoc YOLO-based vision module.

↓

Solution: Multimodality.



1.3 Fine-Tuning VLA Models with PEFT

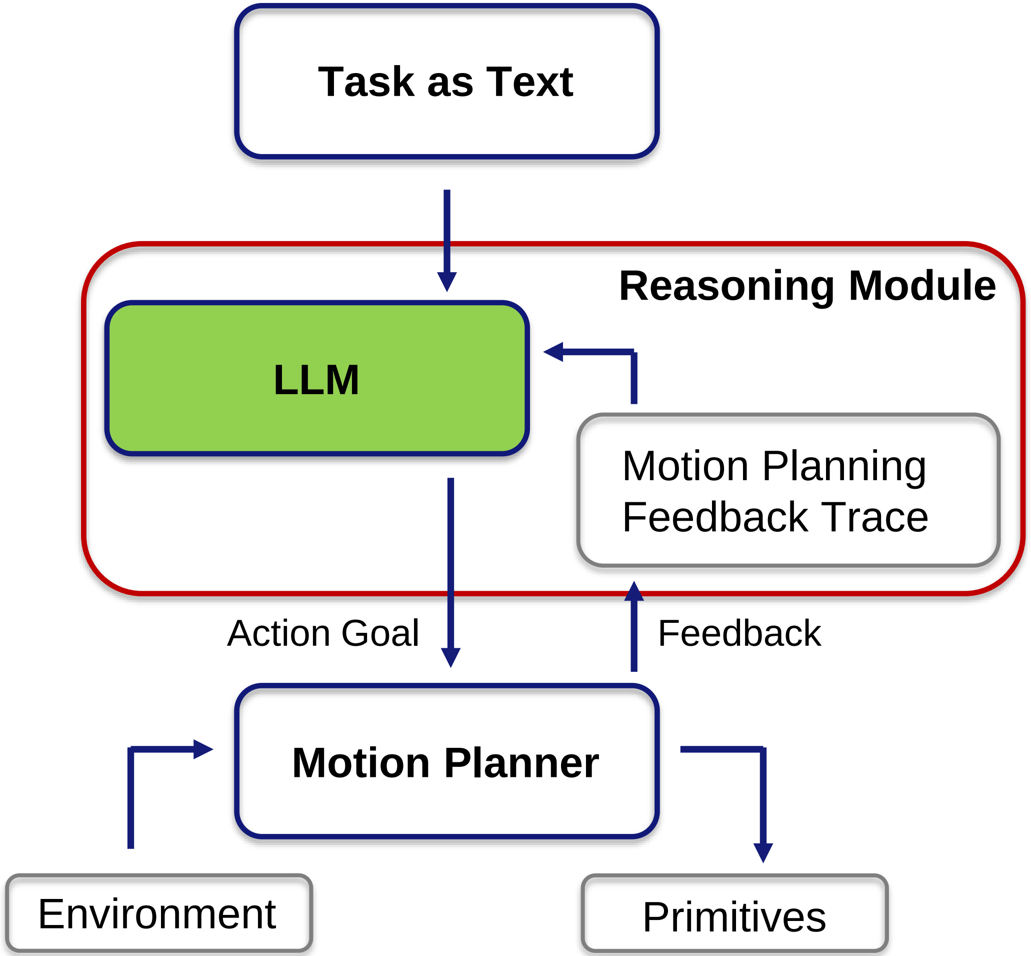
- (+) Fast new skills acquisition without forgetting pre-trained skills.

1.3 Fine-Tuning VLA Models with PEFT

- (+) Fast new skills acquisition without forgetting pre-trained skills.

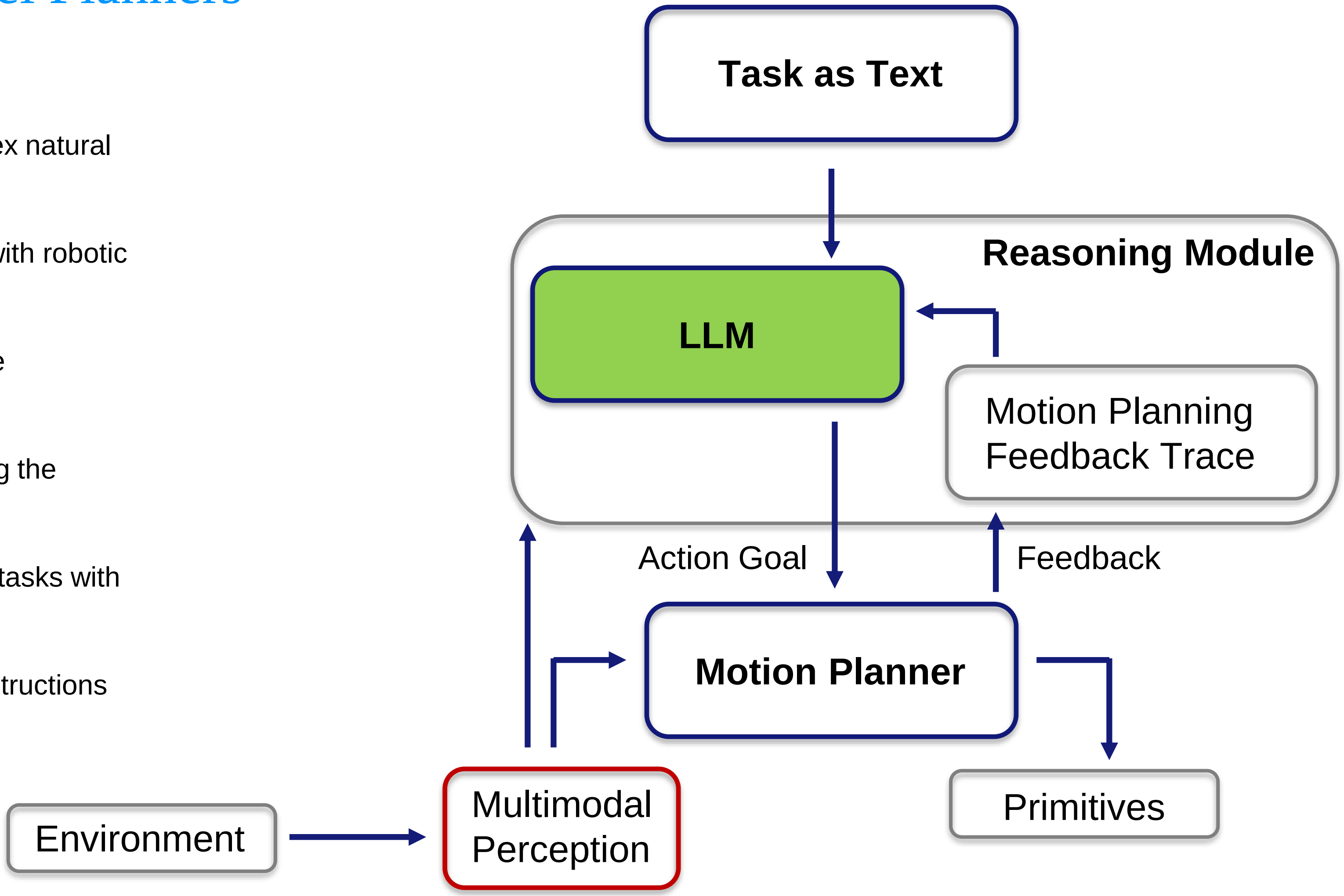
- (-) A backbone model that supports LoRA fine-tuning is required, such as Llama 2.0 LLM for OpenVLA.
- (-) Downstream task dataset collection effort for the specific task and environment (teleoperation).
- (-) GPU hardware requirement + some hours of training.
- (-) New fine-tuning for each new task or environment.

2.1. Zero-Shot LLMs as High-Level Planners



2.1. Zero-Shot LLMs as High-Level Planners

- **Purpose**: SayCan enables robots to follow long and complex natural language instructions.
- **Integration**: Combines LLMs for high-level understanding with robotic skills for low-level control.
- **Mechanism**: LLM proposes useful skills, while skills provide affordances and success probabilities.
- **Decision-making**: Grounds the LLM's outputs by combining the likelihood of skill usefulness with success probabilities.
- **Performance**: Successfully tested on 101 complex kitchen tasks with high success rates.
- **Long-term planning**: Adapts to long and more complex instructions and varied natural language inputs.



❖ **Primitives**: LLMs and VLA models for action generation can act as primitives for LLM-based high-level planners.

2.2. Say Can Model

Challenge: Large language models (LLMs) encode extensive semantic knowledge but lack contextual grounding, limiting their applicability in real-world robotic tasks.

Example: LLMs can describe cleaning a spill but struggle to adapt to a robot's environment and capabilities.

Credit: Do As I Can, Not As I Say   Video

2.2. Say Can Model

Challenge: Large language models (LLMs) encode extensive semantic knowledge but lack contextual grounding, limiting their applicability in real-world robotic tasks.

Example: LLMs can describe cleaning a spill but struggle to adapt to a robot's environment and capabilities.

Idea: Use pretrained robotic behaviors to ground LLMs in context:

- Robots act as the LLM's “hands and eyes” to provide environmental context.
- LLMs supply high-level semantic guidance for task execution.

Credit: Do As I Can, Not As I Say Video

2.2. Say Can Model: Method

1. Combine **low-level value functions** (robotic behaviors) with **LLM-generated high-level knowledge**.
2. LLM proposes **feasible**, contextually appropriate **actions**.
3. **Value functions** connect semantic instructions to the physical environment.

Credit: Do As I Can, Not As I Say

See the project notebook for an implementation of Say Can ...

2.2. Say Can Model: Method

1. Combine **low-level value functions** (robotic behaviors) with **LLM-generated high-level knowledge**.
2. LLM proposes **feasible**, contextually appropriate **actions**.
3. **Value functions** connect semantic instructions to the physical environment.

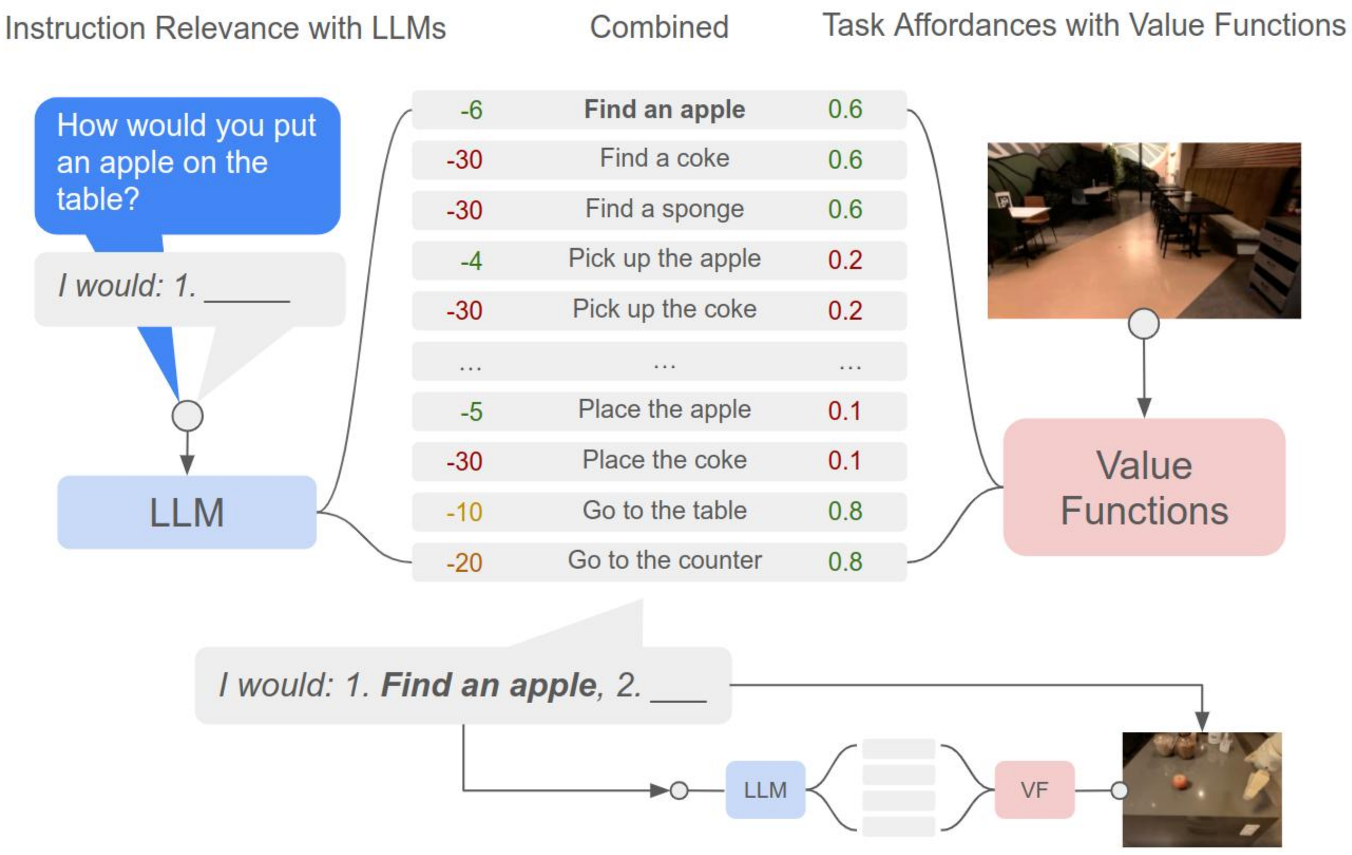
- Demonstrated effectiveness on real-world robotic tasks with a mobile manipulator.
- Successfully completed long-horizon, abstract, natural language instructions.

Credit: [Do As I Can, Not As I Say]

See the project notebook for an implementation of Say Can ...

2.2. Say Can Model: Method

1. Combine **low-level value functions** (robotic behaviors) with **LLM-generated high-level knowledge**.
2. LLM proposes **feasible**, contextually appropriate **actions**.
3. **Value functions** connect semantic instructions to the physical environment.



- Demonstrated effectiveness on real-world robotic tasks with a mobile manipulator.
- Successfully completed long-horizon, abstract, natural language instructions.

Credit: [Do As I Can, Not As I Say]

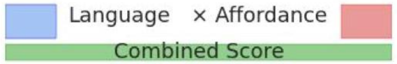
See the project notebook for an implementation of Say Can ...

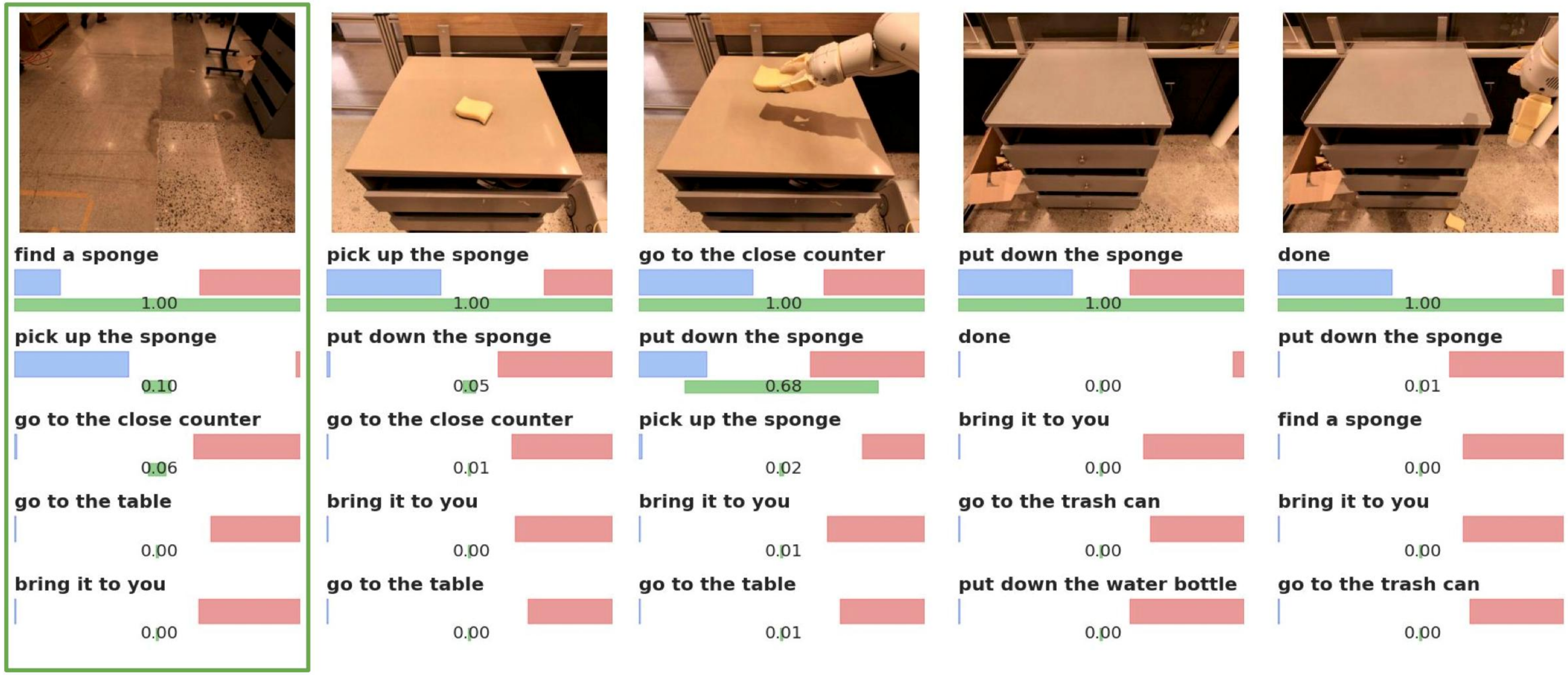
2. Say Can Model: Example

Complete example:

Human: How would you put the sponge on the close counter?

Robot: I would
1. Find a sponge
2. Pick up the sponge
3. Go to close counter
4. Put down the sponge
5. Done



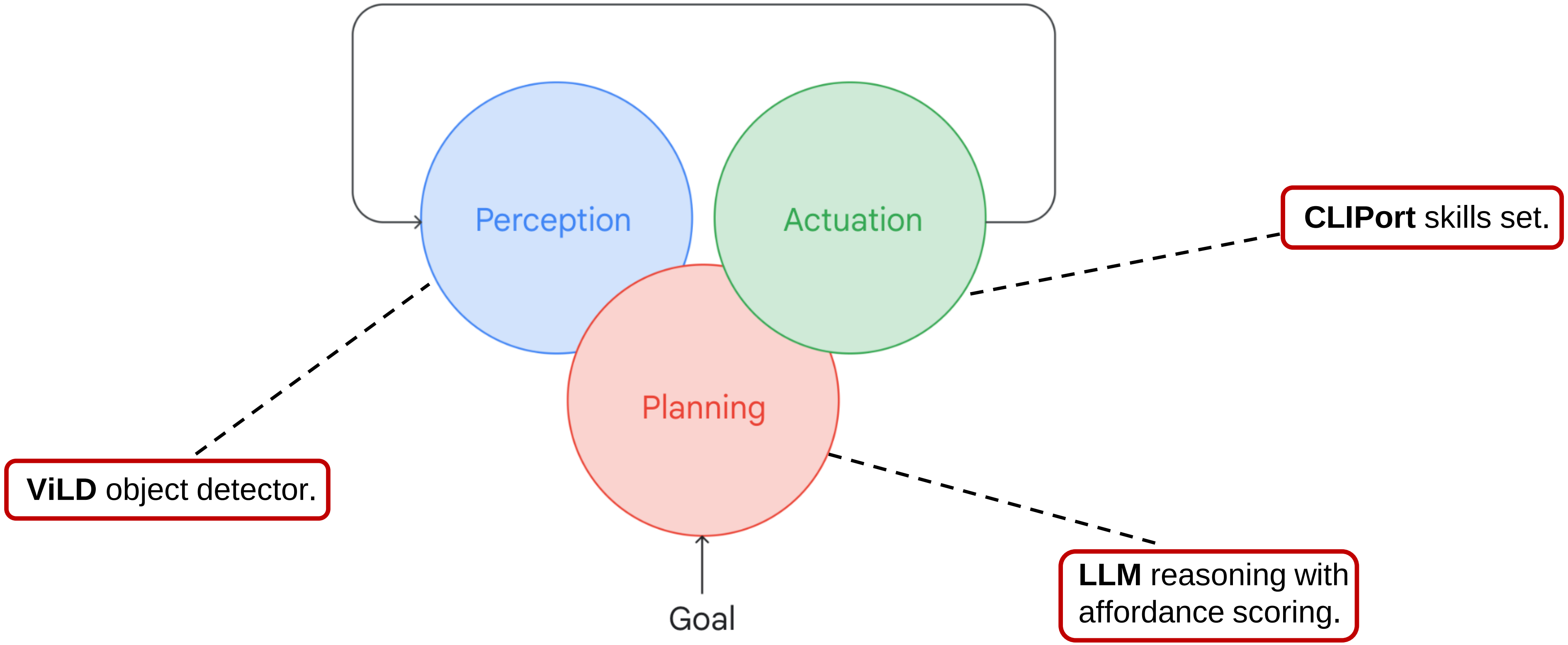


Credit: [Do As I Can, Not As I Say]

2. Say Can Model: Example

Credit: [Do As I Can, Not As I Say]

2. Say Can Model: Details



2. Say Can Model: Details

The robot in SayCan needs a set of "primitive actions" or "skills" it can perform.

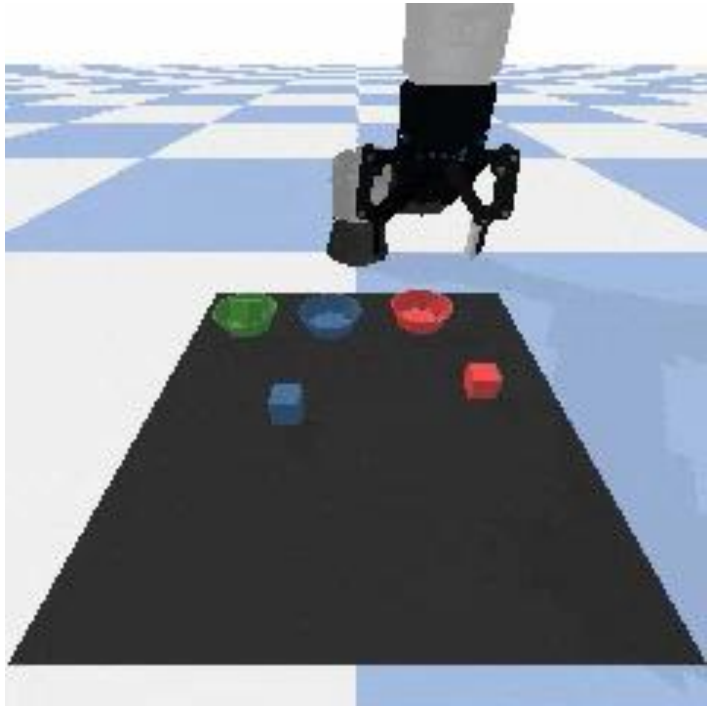
Each **skill** is associated with:

- A **description** (natural language, like "pick the red block"),
- A **policy** (how to execute it),
- A **value function** (how successful that action is likely to be).

Where:

- **Policy**: Code or learned model that knows *how* to do it.
- **Value function**: Predicts the chance of success in current conditions.

The **CLIPort** skill:



2. Say Can Model: Language Model (Say) — Proposing Actions

The notebook **uses a large language model (LLM)** — in practice OpenAI's API — to map a **high-level instruction** ("put the red block on the blue one") into **relevant skills**.

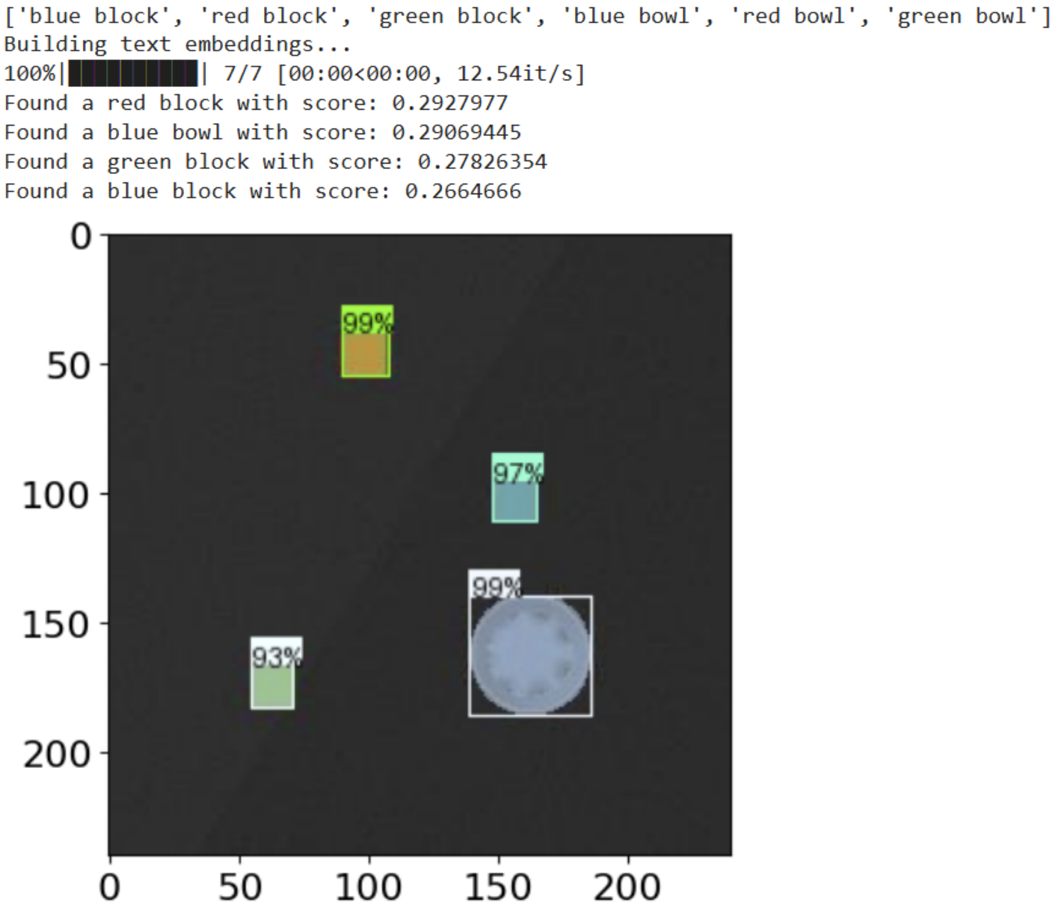
Example prompt given to the LLM:

Instruction: “put the red block on the blue one.”

Available skills:
- pick up red block
- pick up blue block
- place on blue block
- place on table

Which skills are relevant? The LLM responds suggesting the best skills based on the **perception** (ViLD object detection).

The **ViLD** object detection:



2. Say Can Model: Affordance Model (Can) — Feasibility Check

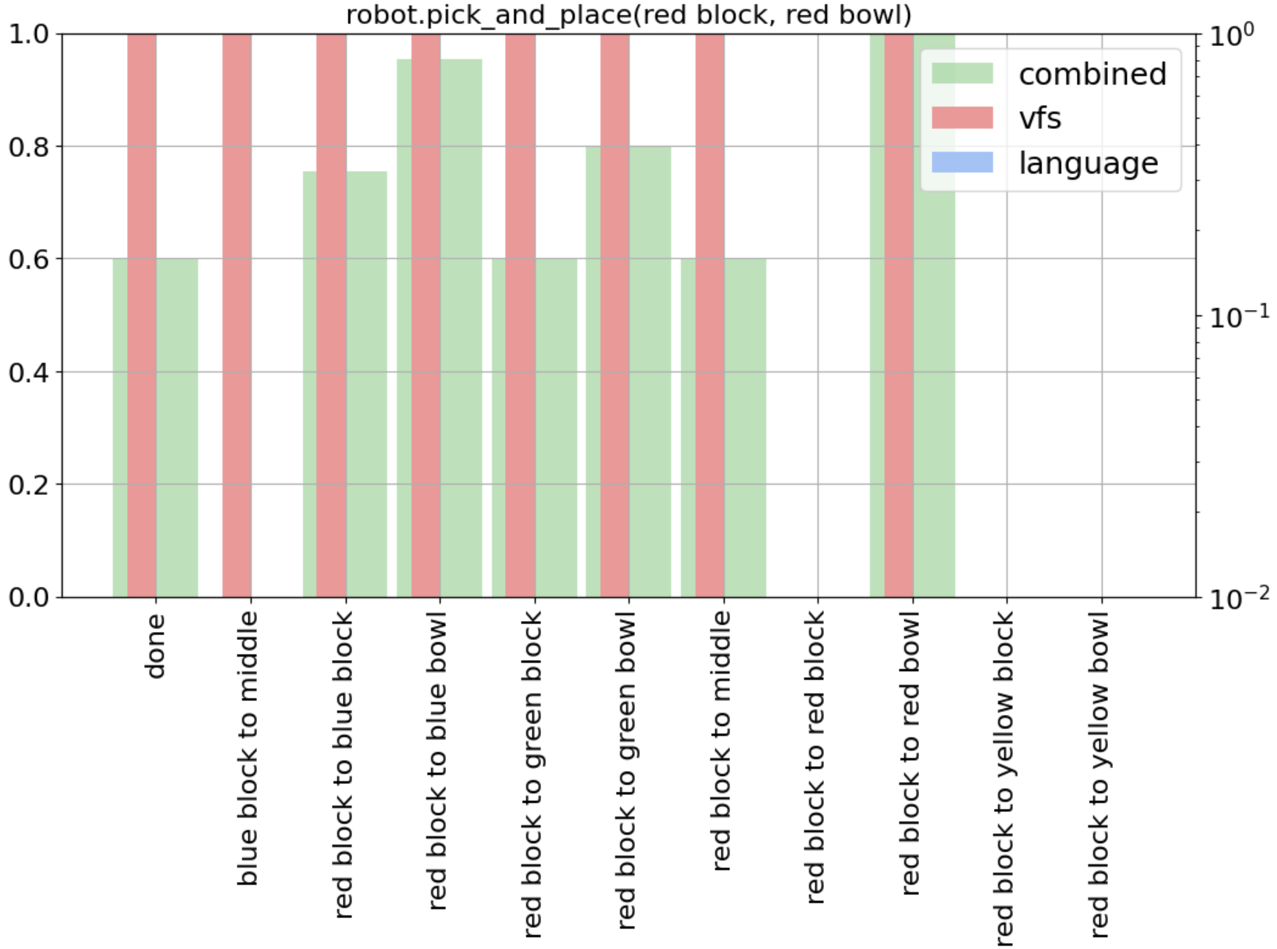
Once the LLM proposes possible skills, each skill is evaluated:

•**Can** it be performed?
•**How likely** is it to succeed now?

Each skill has a **value function**, which estimates a **success probability**.

Here:

•**state** = current observations (camera image, object locations, etc.)
•**value** = real number (0.0 = impossible, 1.0 = guaranteed success).



2. Say Can Model: Combining Say + Can

Now, to choose which skill to execute, **both the LLM's proposal and the robot's feasibility are combined**.

The **score** for each skill is:

score = p(skill | instruction) * p(success | state)

where:

- **p(skill | instruction)**: probability the LLM gives for the skill (semantic relevance),
- **p(success | state)**: affordance model value (physical feasibility).

The robot selects the **highest scoring** skill.

The robot **executes the chosen skill**:

It then **observes** the new environment state, **re-evaluates**, and **plans the next action**.

This process continues **until the final goal is achieved** or a **termination condition** (like failure) is met.

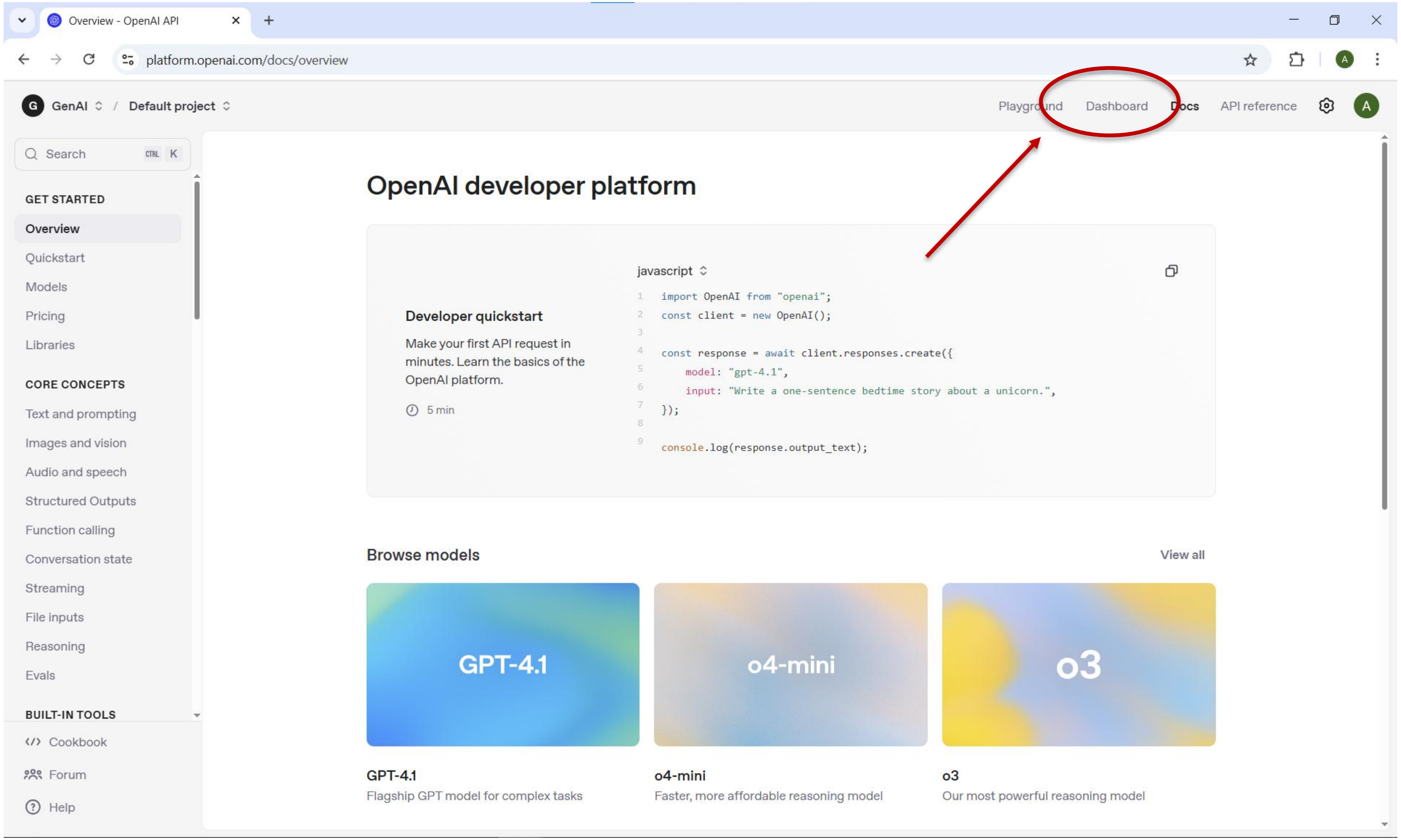
Step	SayCan Process
1	User says: "Put red block on blue block"
2	LLM suggests skills: ["pick red", "place on blue"]
3	Value function says: "picking red = 90% success", "placing on blue = 20%"
4	SayCan picks "pick red" first (highest score)
5	Robot picks up red block
6	New environment → re-check skills
7	Now placing on blue is easier (80% success)
8	Robot places red block on blue block
9	Task complete

3. Practice

1. Preliminary exercises: https://colab.research.google.com/drive/1WgrOtIuBNSXoAJ4AkxZCYh0mnB4a8hMB?usp=sharing .
2. Create an account for each group on the Open AI API portal at the following link and write here your account email to be added in the project (to obtain free tokens): https://platform.openai.com/docs/overview.
3. Create your personal API key and insert it in the correct lines of the notebook.
4. Run the code cells to implement the tutorial explained during the practical lesson.

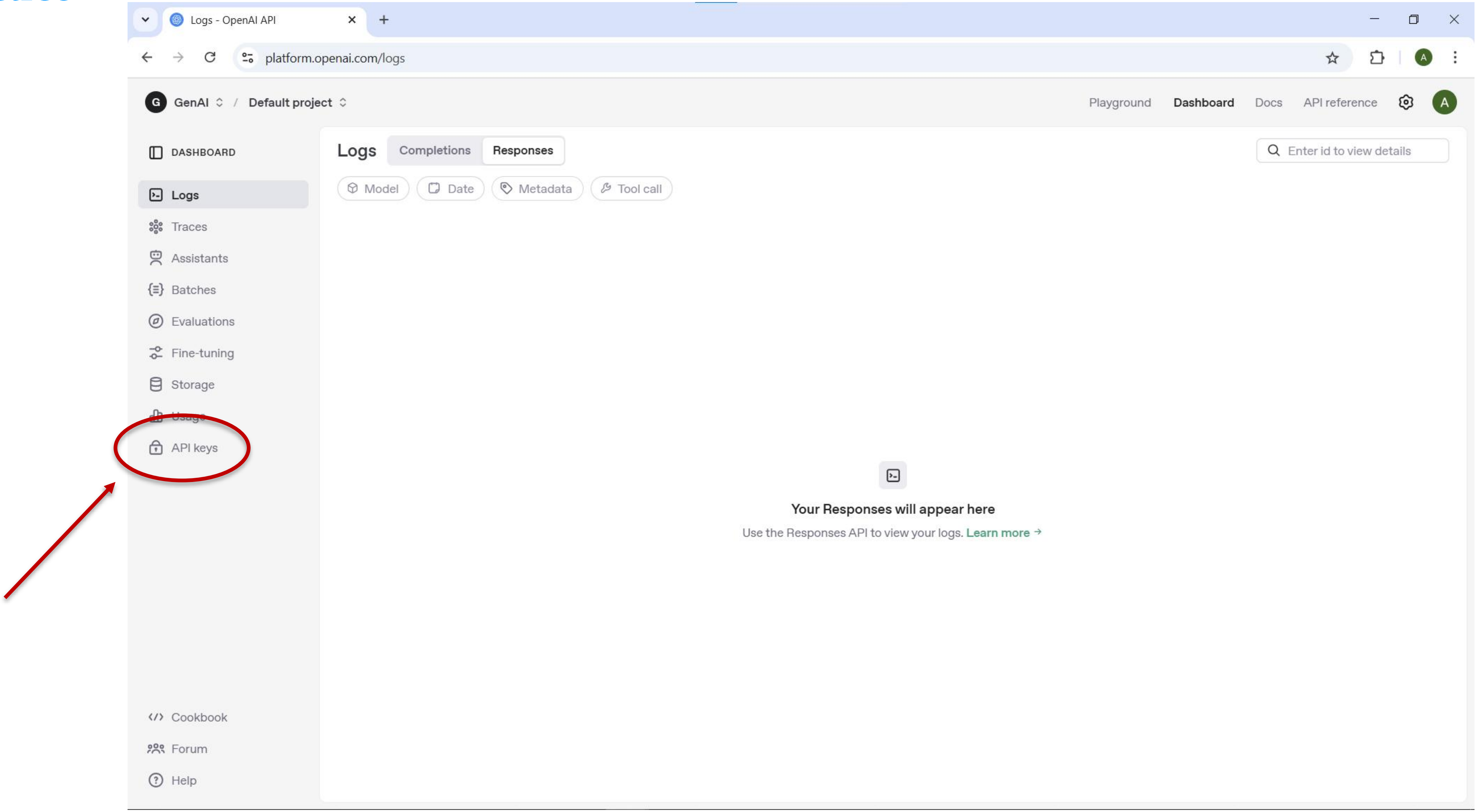
5. Open the Colab notebook of SayCan at this link: https://colab.research.google.com/drive/153PA_ihR68-mDDPNW-6Y74axU4r2GQyZ?usp=sharing .
6. Try yourself: modify the code changing the calls to the LLM in order to create more complex plans.
7. Here a live demo of SayCan by Google: https://sites.research.google/palm-saycan .

3. Practice



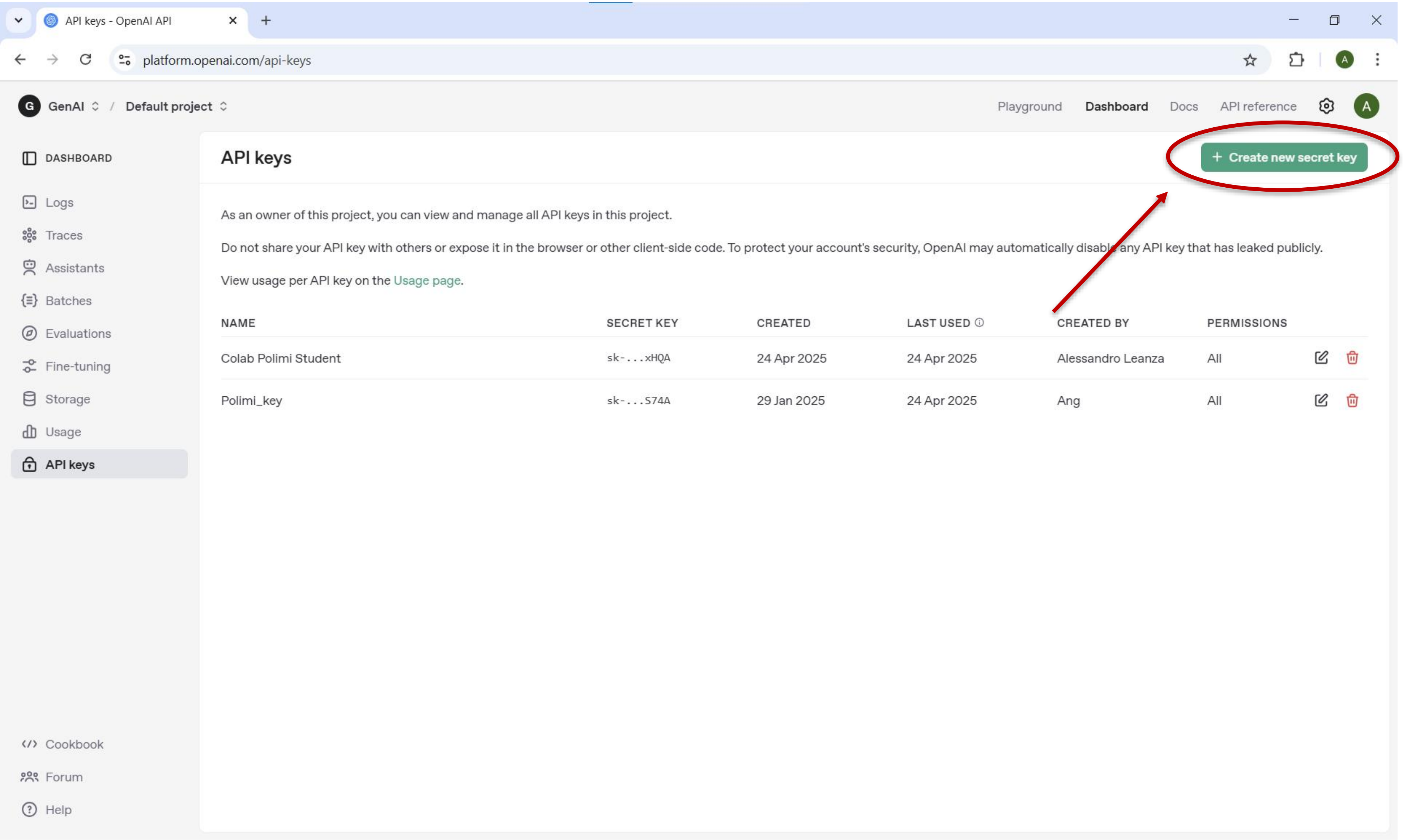
Overview - OpenAI API
platform.openai.com/docs/overview
GenAI / Default project
Playground
Dashboard
Docs
API reference
Search CTRL K
GET STARTED
Overview
Quickstart
Models
Pricing
Libraries
CORE CONCEPTS
Text and prompting
Images and vision
Audio and speech
Structured Outputs
Function calling
Conversation state
Streaming
File inputs
Reasoning
Evals
BUILT-IN TOOLS
Cookbook
Forum
Help
OpenAI developer platform
Developer quickstart
Make your first API request in minutes. Learn the basics of the OpenAI platform.
5 min
javascript
import OpenAI from "openai";
const client = new OpenAI();
const response = await client.responses.create({
model: "gpt-4.1",
input: "Write a one-sentence bedtime story about a unicorn.",
});
console.log(response.output_text);
Browse models
View all
GPT-4.1
o4-mini
o3
GPT-4.1
Flagship GPT model for complex tasks
o4-mini
Faster, more affordable reasoning model
o3
Our most powerful reasoning model

3. Practice

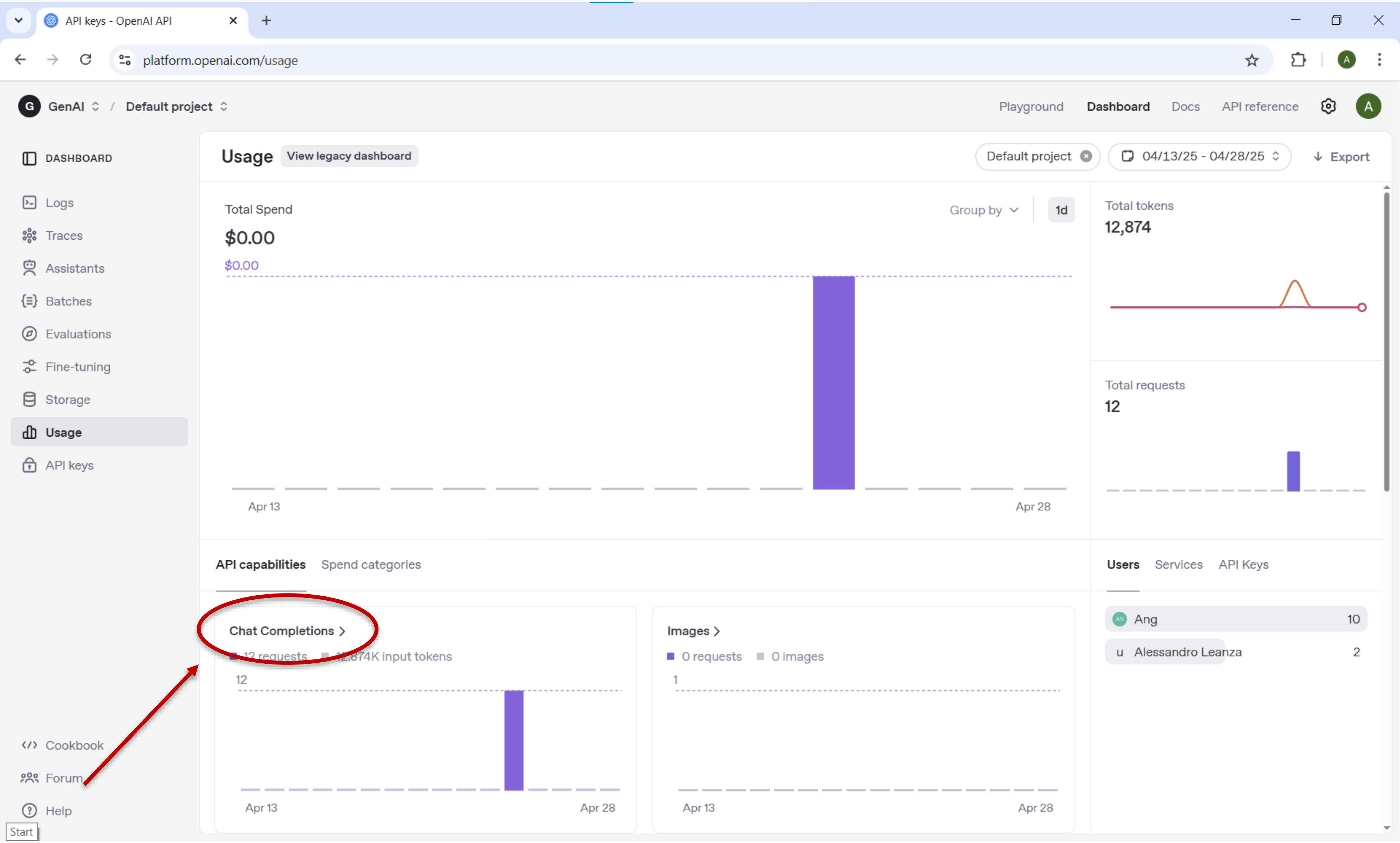


Logs - OpenAI API
platform.openai.com/logs
GenAI / Default project
Playground
Dashboard
Docs
API reference
DASHBOARD
Logs
Traces
Assistants
Batches
Evaluations
Fine-tuning
Storage
Usage
API keys
Cookbook
Forum
Help
Logs
Completions
Responses
Enter id to view details
Model
Date
Metadata
Tool call
Your Responses will appear here
Use the Responses API to view your logs. Learn more →

3. Practice

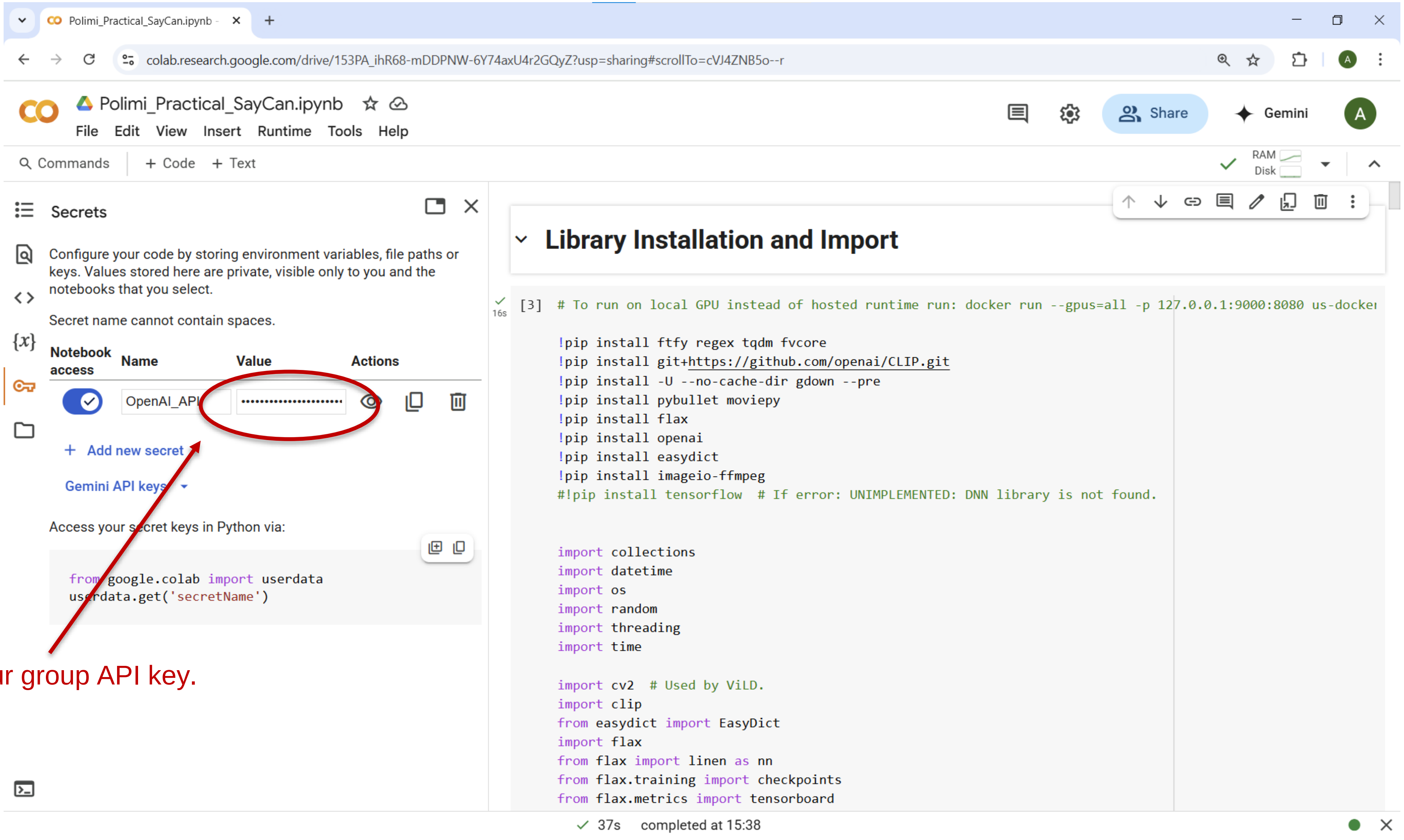

API keys - OpenAI API
platform.openai.com/api-keys
GenAI / Default project
Playground
Dashboard
Docs
API reference
DASHBOARD
Logs
Traces
Assistants
Batches
Evaluations
Fine-tuning
Storage
Usage
API keys
Cookbook
Forum
Help
API keys
+ Create new secret key
As an owner of this project, you can view and manage all API keys in this project.
Do not share your API key with others or expose it in the browser or other client-side code. To protect your account's security, OpenAI may automatically disable any API key that has leaked publicly.
View usage per API key on the Usage page.
NAME
SECRET KEY
CREATED
LAST USED
CREATED BY
PERMISSIONS
Colab Polimi Student
sk-...xHQA
24 Apr 2025
24 Apr 2025
Alessandro Leanza
All
Polimi_key
sk-...S74A
29 Jan 2025
24 Apr 2025
Ang
All

3. Practice



API keys - OpenAI API
platform.openai.com/usage
GenAI / Default project
Playground
Dashboard
Docs
API reference
DASHBOARD
Logs
Traces
Assistants
Batches
Evaluations
Fine-tuning
Storage
Usage
API keys
Cookbook
Forum
Help
Usage
View legacy dashboard
Default project
04/13/25 - 04/28/25
Export
Total Spend
$0.00
Group by
1d
Apr 13
Apr 28
Total tokens
12,874
Total requests
12
API capabilities
Spend categories
Chat Completions >
12 requests
12.874K input tokens
Images >
0 requests
0 images
Users
Services
API Keys
Ang 10
Alessandro Leanza 2

3. Practice



Put here your group API key.

4. Project (1st trace): LLM-Assisted Process Planning for CNC Machining

Objective:
Students will design a simple Python pipeline where a **LLM** assists in **mechanical process planning** for manufacturing a mechanical part via **CNC machining**.

Task:
The idea is to use LLMs to *augment* traditional planning: the LLM could help, for instance, by:
• Interpreting a **part description** (e.g., a prompt describing a bracket or a gear),
• Suggesting **process steps** (e.g., roughing, finishing, drilling),
• Proposing **setup plans** (e.g., how to fix the part on a machine),
• Selecting **appropriate tools** (e.g., end mill, drill bit),
• Estimating **operation sequences** and **machine parameters** (basic ones, like speed, feed).

Students must build a Python Notebook that:
• Accepts a text input (natural language description of a part, or even a simple CAD feature list),
• Calls an LLM API (e.g., OpenAI or similar) to generate a suggested process plan based on the input context,
• Parses and structures the output (e.g., as a table: step, tool, operation, parameters): prompt engineering and LLM fine-tuning are optional but they can be useful,
• Optionally refines the LLM output based on mechanical constraints (e.g., max spindle speed, material),
• Outputs a clean, structured mechanical process plan.

!! Please, do not fine-tune the models using the shared project APIs, they are only for inference.

Required Elements:
1. LLM Call: Clearly show how the model is queried and how prompts are structured.
2. Postprocessing: Clean, validate, and possibly reformat the LLM response.
3. Validation Logic: Some mechanical sanity checks (e.g., flagging impossible spindle speeds, wrong tool choices).
4. Notebook Output: Present results clearly (tables, comments).
5. Reflection Section: Short analysis of how reliable/helpful the LLM was, and where human oversight was needed.

4. Project (2nd trace): Extending SayCan for Enhanced LLM-Guided Robot Planning

Objective:
Starting from the provided project notebook, students will **modify and extend SayCan** to improve **at least one** of the following aspects:

Tasks (choose at least one):

1. Feedback and Plan Adjustment

1. Take the generated plan from the LLM call.
2. Execute or simulate the actions.
3. Feed back new information into the LLM as updated input.
4. Analyze: **Does the plan change based on feedback?**

2. Advanced LLM Grounding

1. Develop a more complex grounding strategy: give more low-level information to the LLM.
2. Include additional robot state information (e.g., objects geometry, precise positions) in the prompt or in the LLM context.

3. Environment and Object Detection Enhancements

1. Modify the environment configuration to create new scenarios or new objects and skills.
2. Train **new primitives** using **CLIPort** for these scenarios.
3. *(Optional)* Replace manual object definitions with the **ViLD object detector (or others such as YOLO)** to dynamically detect objects.

- ➢ Hint: In "Affordance Scoring", change affordance=1. What happens?
- ➢ Hint: In "Setup Scene", try to modify found_objects and see what happens.
- ➢ Hint: Try to modify the parameters in "Task and Config".
- ➢ Hint: In "Setup Scene", try to comment found_objects manually defined and use ViLD. Modify the random seed to have different configurations of objects for ViLD.
- ➢ Hint: Try to define new primitives with CLIPort in PyBullet and train new skills. Try to implement a new task planning with the LLM.
- ➢ Hint: This notebook is based on OpenAI API that grants access to LogProp (a specific feature of OpenAPI API). Try to develop a planner based on the open-source Llama 3 model using other features, such as, normal textual responses from the LLM.