



# Executable Code Actions Elicit Better LLM Agents

---

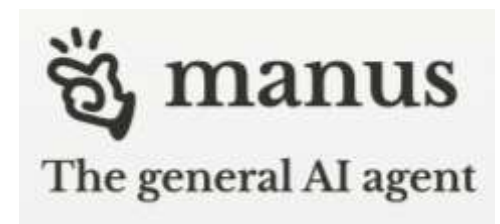
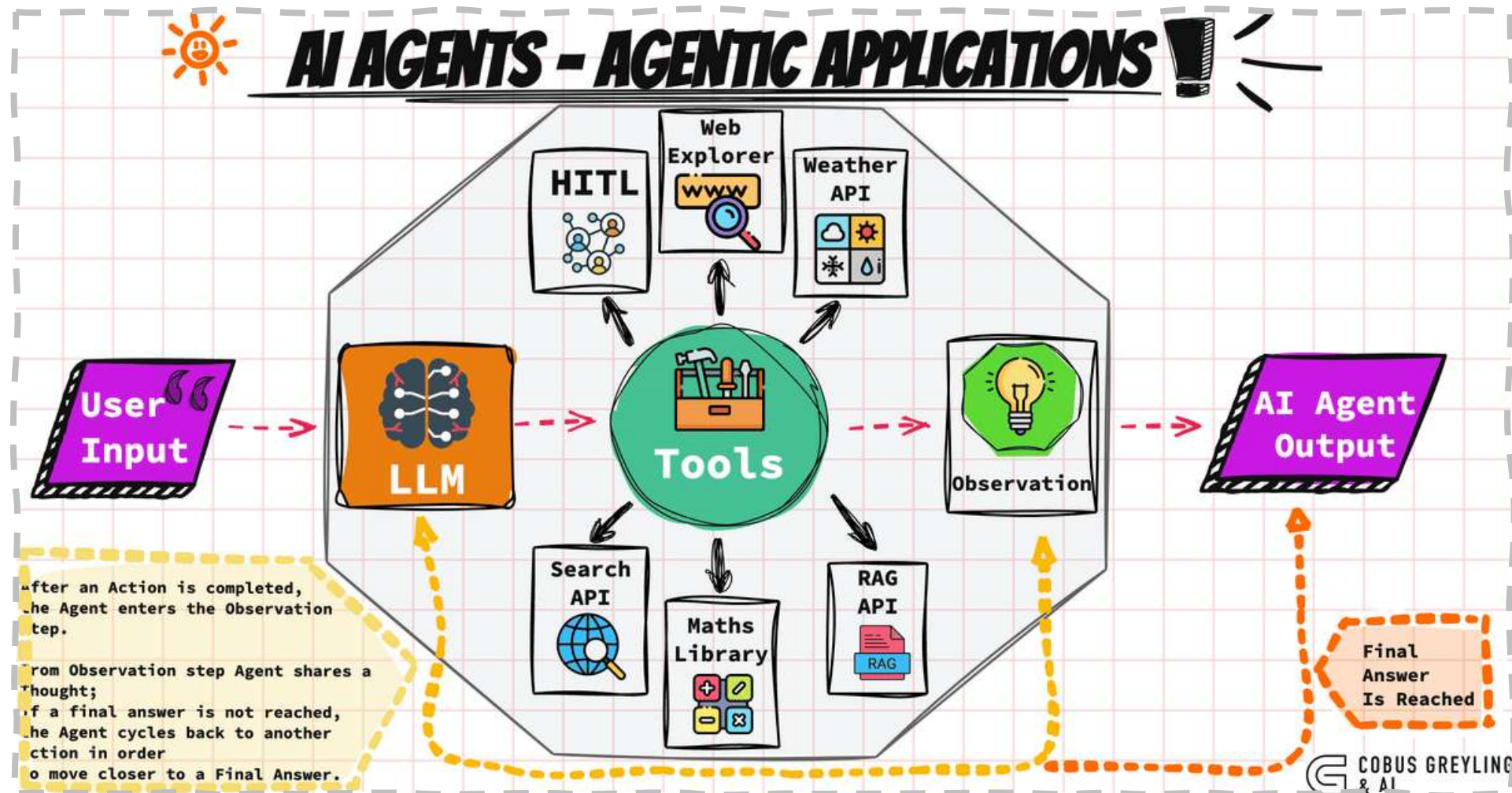
Xingyao Wang<sup>1</sup> Yangyi Chen<sup>1</sup> Lifan Yuan<sup>1</sup> Yizhe Zhang<sup>2</sup> Yunzhu Li<sup>1</sup> Hao Peng<sup>1</sup> Heng Ji<sup>1</sup>

<sup>1</sup>Department of Computer Science, University of Illinois Urbana-Champaign <sup>2</sup>Apple. Correspondence to: Xingyao Wang <xingyao6@illinois.edu>, Heng Ji <hengji@illinois.edu>.

*ICML 2024*

# What is Agent?

An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.



COBUS GREYLING & AI

# Introduction to CodeAct

LLM agents are typically prompted to produce actions by generating JSON or text in a pre-defined format, which is usually limited by constrained action space (e.g., the scope of pre-defined tools) and restricted flexibility (e.g., inability to compose multiple tools).

This work proposes to use executable Python **code** to consolidate LLM agents' actions into a unified **action** space.

**Instruction:** Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

### Available APIs

[1] lookup\_rates(country: str) -> (float, float)

[2] convert\_and\_tax(price: float, exchange\_rate: float, tax\_rate: float) -> float

[3] estimate\_final\_price(converted\_price: float, shipping\_cost: float) -> float

[4] lookup\_phone\_price(model: str, country: str) -> float

[5] estimate\_shipping\_cost(destination\_country: str) -> float

### LLM Agent using [Text/JSON] as Action

**Think** I should calculate the phone price in USD for each country, then find the most cost-effective country.

**Action** Text: lookup\_rates, Germany  
JSON: {"tool": "lookup\_rates", "country": "Germany"}

**Environment** 1.1, 0.19

**Action** Text: lookup\_phone\_price, CodeAct 1, Germany  
JSON: {"tool": "lookup\_phone\_price", "model": "CodeAct 1", "country": "Germany"}

**Environment** 700

**Action** Text: convert\_and\_tax, 700, 1.1, 0.19  
JSON: {"tool": "convert\_and\_tax", "price": 700, "exchange\_rate": 1.1, "tax\_rate": 0.19}

**Environment** 916.3

[... interactions omitted (look up shipping cost and calculate final price) ...]

**Action** Text: lookup\_rates, Japan  
JSON: {"tool": "lookup\_rates", "country": "Japan"}

[... interactions omitted (calculate final price for all other countries) ...]

**Response** The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Fewer Actions Required!

### CodeAct: LLM Agent using [Code] as Action

**Think** I should calculate the phone price in USD for each country, then find the most cost-effective country.

**Action**

```
countries = ['USA', 'Japan', 'Germany', 'India']
final_prices = {}

for country in countries:
    exchange_rate, tax_rate = lookup_rates(country)
    local_price = lookup_phone_price("xAct 1", country)
    converted_price = convert_and_tax(
        local_price, exchange_rate, tax_rate
    )
    shipping_cost = estimate_shipping_cost(country)
    final_price = estimate_final_price(converted_price, shipping_cost)
    final_prices[country] = final_price

most_cost_effective_country = min(final_prices, key=final_prices.get)
most_cost_effective_price = final_prices[most_cost_effective_country]
print(most_cost_effective_country, most_cost_effective_price)
```

Control & Data Flow of Code Simplifies Complex Operations

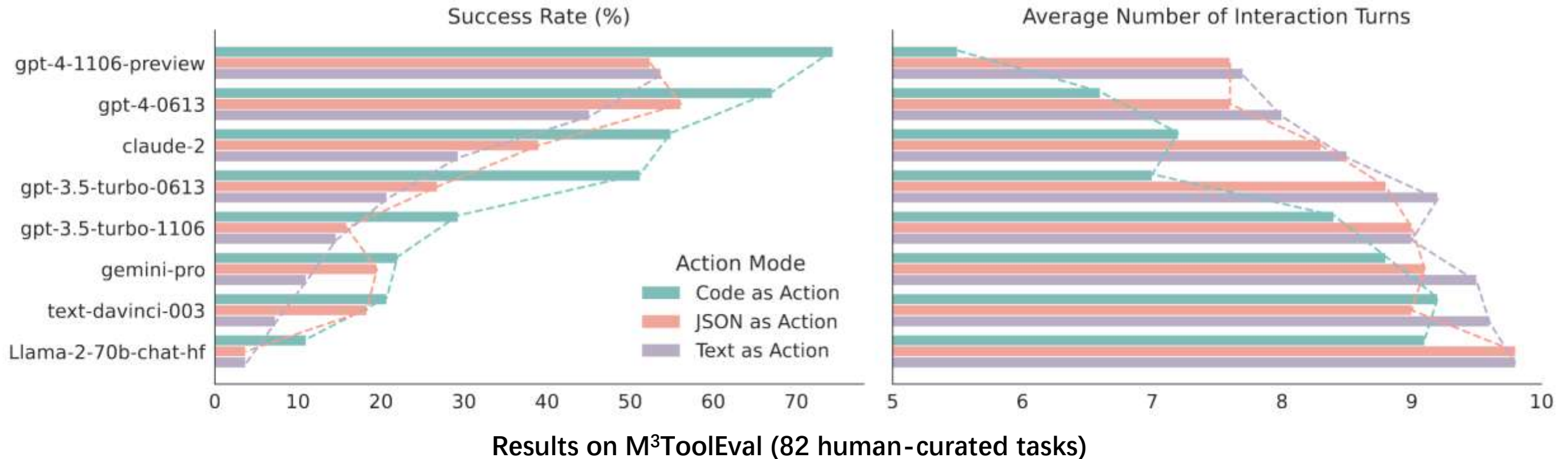
Re-use 'min' Function from Existing Software Infrastructures (Python library)

**Environment** 1.1, 0.19

**Response** The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

# Advantages

The extensive experiments with 17 LLMs show that, CodeAct achieves up to a **20% absolute improvement** over baselines on the success rate of solving the problems while requiring up to **30% fewer actions**.



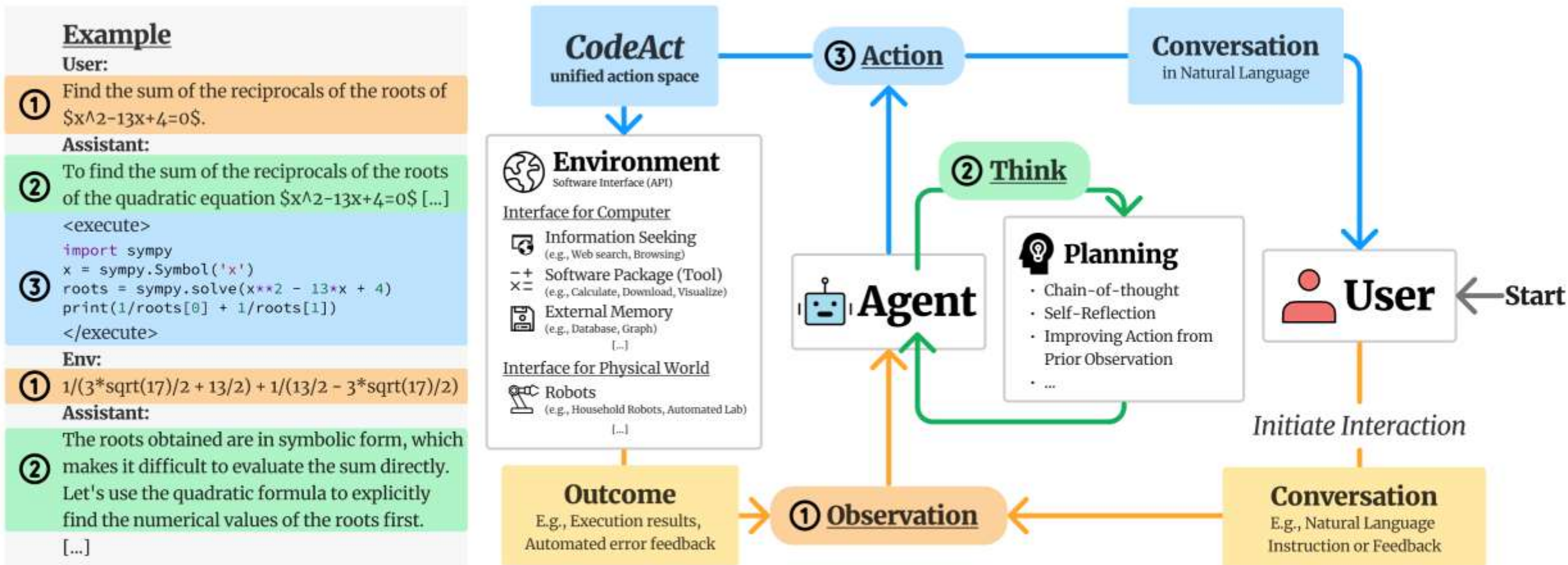


Figure 2: General agent multi-turn interaction framework that describes the role of CodeAct and motivates the construction of our data mixture. CodeActInstruct focuses on the *agent-environment* interactions and specifically filters for the self-improved *planning* behavior, while general conversation data we include focuses on *agent-user* interaction (§3.1).

```
<|im_start|>system
```

A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions.

The assistant can interact with an interactive Python (Jupyter Notebook) environment and receive the corresponding output when needed. The code should be enclosed using "<execute>" tag, for example: <execute> print("Hello World!") </execute>.

The assistant should attempt fewer things at a time instead of putting too much code in one <execute> block. The assistant can install packages through PIP by <execute> !pip install [package needed] </execute> and should always import packages and define variables before starting to use them.

The assistant should stop <execute> and provide an answer when they have already obtained the answer from the execution result. Whenever possible, execute the code for the user using <execute> instead of providing it.

The assistant's response should be concise, but do express their thoughts.

```
<|im_end|>
```

# Results

Table 2: Atomic API call correctness on API-Bank. The best performance is **bolded**, and the second-best is underlined.

Format of Action	Correctness (% , $\uparrow$ )		
	CodeAct	JSON	Text
<i>Open-source LLMs</i>			
CodeLlama-7b-Instruct-hf	<u>12.5</u>	12.0	<b>17.0</b>
CodeLlama-13b-Instruct-hf	<u>11.8</u>	7.8	<b>14.0</b>
CodeLlama-34b-Instruct-hf	<b>17.3</b>	12.0	<u>16.8</u>
Llama-2-7b-chat-hf	<b>28.8</b>	11.3	<u>25.8</u>
Llama-2-13b-chat-hf	<b>38.1</b>	8.5	<u>37.3</u>
Llama-2-70b-chat-hf	<u>35.6</u>	14.3	<b>37.6</b>
Mistral-7B-Instruct-v0.1	<u>2.5</u>	2.3	<b>3.0</b>
lemur-70b-chat-v1	<b>58.6</b>	46.6	<u>56.1</u>
<i>Closed-source LLMs</i>			
claude-2	<b>76.7</b>	59.4	<u>73.7</u>
claude-instant-1	<b>75.2</b>	64.9	<u>73.2</u>
gemini-pro	70.4	<b>73.2</b>	<u>71.2</u>
gpt-3.5-turbo-0613	<b>74.4</b>	<u>73.9</u>	73.4
gpt-3.5-turbo-1106	<u>75.4</u>	<b>78.4</b>	73.4
gpt-4-0613	<u>75.4</u>	<b>82.0</b>	74.4
gpt-4-1106-preview	<u>76.7</u>	<b>82.7</b>	73.4
text-davinci-002	<b>69.2</b>	<u>59.6</u>	57.4
text-davinci-003	<u>75.4</u>	<b>76.9</b>	69.7
<b>Frequency of Best-Performing Format <math>\uparrow</math></b>			
Open-source	<b>4</b>	0	<u>4</u>
Closed-source	<u>4</u>	<b>5</b>	0
<b>Overall</b>	<b>8</b>	<u>5</u>	4

Table 3: Success rates (higher the better) and average turns required per instance (lower the better) on M<sup>3</sup>ToolEval. The best results for each model are **bolded**, and the second-best ones are underlined.

Format of Action	Success Rate (% , $\uparrow$ )			Avg. Turns ( $\downarrow$ )		
	CodeAct	JSON	Text	CodeAct	JSON	Text
<i>Open-source LLMs</i>						
CodeLlama-7b-Instruct-hf	<b>4.9</b>	<u>2.4</u>	<u>2.4</u>	<b>9.7</b>	<u>9.9</u>	<u>9.9</u>
CodeLlama-13b-Instruct-hf	<b>4.9</b>	<b>4.9</b>	<b>4.9</b>	<u>9.8</u>	<u>9.8</u>	<b>9.7</b>
CodeLlama-34b-Instruct-hf	<b>2.4</b>	<u>0.0</u>	<u>0.0</u>	<b>9.9</b>	<u>10.0</u>	<u>10.0</u>
Llama-2-7b-chat-hf	0.0	<u>1.2</u>	<b>2.4</b>	<b>8.9</b>	<u>9.5</u>	9.6
Llama-2-13b-chat-hf	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>	<b>9.7</b>	<u>10.0</u>	<u>10.0</u>
Llama-2-70b-chat-hf	<b>11.0</b>	<u>3.7</u>	<u>3.7</u>	<b>9.1</b>	<u>9.8</u>	<u>9.8</u>
Mistral-7B-Instruct-v0.1	0.0	<b>3.7</b>	<u>1.2</u>	10.0	<b>9.8</b>	<u>9.9</u>
lemur-70b-chat-v1	<u>13.4</u>	<b>15.9</b>	12.2	<b>9.1</b>	<u>9.3</u>	9.4
<i>Closed-source LLMs</i>						
claude-2	<b>54.9</b>	<u>39.0</u>	29.3	<b>7.2</b>	<u>8.3</u>	8.5
claude-instant-1	20.7	<b>31.7</b>	<u>24.4</u>	<u>8.8</u>	<b>8.6</b>	8.9
gemini-pro	<b>22.0</b>	<u>19.5</u>	11.0	<b>8.8</b>	<u>9.1</u>	9.5
gpt-3.5-turbo-0613	<b>51.2</b>	<u>26.8</u>	20.7	<b>7.0</b>	<u>8.8</u>	9.2
gpt-3.5-turbo-1106	<b>29.3</b>	<u>15.9</u>	14.6	<b>8.4</b>	<u>9.0</u>	<u>9.0</u>
gpt-4-0613	<b>67.1</b>	<u>56.1</u>	45.1	<b>6.6</b>	<u>7.6</u>	8.0
gpt-4-1106-preview	<b>74.4</b>	52.4	<u>53.7</u>	<b>5.5</b>	<u>7.6</u>	7.7
text-davinci-002	<u>4.9</u>	<u>4.9</u>	<b>8.5</b>	<u>9.7</u>	9.8	<b>9.6</b>
text-davinci-003	<b>20.7</b>	<u>18.3</u>	7.3	<u>9.2</u>	<b>9.0</b>	9.6
<b>Frequency of Best-performing Format <math>\uparrow</math></b>						
Open-source	<b>5</b>	<u>4</u>	3	<b>6</b>	<u>1</u>	<u>1</u>
Closed-source	<b>7</b>	<u>1</u>	<u>1</u>	<b>6</b>	<u>2</u>	1
<b>Overall</b>	<b>12</b>	<u>5</u>	4	<b>12</b>	<u>3</u>	2

To improve open-source LLMs’ CodeAct capability, the authors introduce **CodeActInstruct**, an instruction finetuning dataset that contains agent-environment interaction trajectories.

Table 4: Statistics of our training mixture and comparison with prior work. Please refer to §3.1 for details about CodeActInstruct and general conversation data. Token statistics are computed using Llama-2 tokenizer.

Data Mixture	Data Type	Data Name	# of Data Instances	# of Total Tokens	Avg. Tokens Per Instance
<b>Prior Work</b>	-	FireAct (Chen et al., 2023a)	2,063	542,176	262.81
	-	AgentInstruct (Zeng et al., 2023)	1,866	2,517,785	1349.30
<b>CodeActInstruct (Ours)</b>	Information Seeking	HotpotQA (Yang et al., 2018)	1,664	2,472,227	1485.71
	Software Packages (Tool)	MATH (Math, (Hendrycks et al., 2021b))	1,732	1,719,467	992.76
	Software Packages (Tool)	APPS (Code, (Hendrycks et al., 2021a))	647	1,235,472	1909.54
	External Memory	WikiTableQuestion (Pasupat & Liang, 2015)	1,065	1,316,246	1235.91
	Robot Planning	ALFWorld (Shridhar et al., 2020)	2,031	3,838,269	1889.84
		<b>Total</b>	<b>7,139</b>	<b>10,581,681</b>	<b>1482.24</b>
<b>General Conversation</b>	Single-Turn Reasoning	OpenOrca (Sub-sampled, (Lian et al., 2023))	50,000	14,034,152	280.68
	Multi-Turn Conversations	ShareGPT (Sub-sampled, (Anonymous, 2023))	10,000	17,933,861	1793.39
	Multi-Turn Conversations	ShareGPT (GPT-4, (OpenChat, 2023))	4,583	18,195,878	3970.30
	Multi-turn Reasoning	CapyBara (LDJnr, 2023)	4,647	4,982,435	1072.18
		<b>Total</b>	<b>69,230</b>	<b>55,146,326</b>	<b>796.57</b>

CodeActAgent has some training domains overlapping with MINT’s evaluation, hence the authors report separate numbers for MINT’s in- and out-of-domain performance.

Table 5: Evaluation results for CodeActAgent. The best results among all open-source LLMs are **bolded**, and the second-best results are underlined. ID and OD stand for in-domain and out-of-domain evaluation correspondingly. Overall averaged performance normalizes the MT-Bench score to be consistent with other tasks and excludes in-domain tasks for fair comparison.

Model	Size	Agent Tasks					Generic Tasks				Overall Average
		Code as Action			Text as Action (OD)		(OD)				
		MINT (ID)	MINT (OD)	M <sup>3</sup> ToolEval (OD)	Miniwob++	SciWorld	MMLU	HumanEval	GSM8K	MTBench	
<i>Open-source LLMs (LLaMA-2-based)</i>											
Llama2 Base	7B	-*	-*	-*	-*	-*	45.3	12.8	14.6	-*	-*
Llama2 Chat	7B	3.2	11.0	<u>0.0</u>	0.0	5.9	48.0	13.9	27.7	6.3	21.1
FireAct (Chen et al., 2023a)	7B	0.0	0.3	<u>0.0</u>	0.0	6.8	44.1	3.5	12.4	4.5	14.0
AgentLM (Zeng et al., 2023)	7B	8.7	6.1	<u>0.0</u>	<u>28.9</u>	13.7	48.7	15.4	24.6	6.1	24.8
CodeActAgent (LLaMA-2)	7B	<u>51.3</u>	<u>20.4</u>	<u>0.0</u>	25.5	<b>17.6</b>	50.6	18.1	38.3	<u>7.5</u>	<u>30.7</u>
<i>Open-source LLMs (Mistral-based)</i>											
Mistral Base	7B	-*	-*	-*	-*	-*	<b>60.1</b>	<u>30.5</u>	<u>52.1</u>	-*	-*
Mistral Instruct	7B	18.8	9.7	<u>0.0</u>	0.5	4.0	53.8	29.3	43.3	6.4	25.6
CodeActAgent (Mistral)	7B	<b>57.4</b>	<b>32.4</b>	<b>12.2</b>	<b>46.2</b>	<u>15.9</u>	<u>59.1</u>	<b>34.7</b>	<b>58.0</b>	<b>8.2</b>	<b>42.5</b>
<i>Closed-source LLMs</i>											
gpt-3.5-turbo-0613	-	33.9	38.2	51.2	66.7	21.2	70.0	48.1	57.1	7.9	54.0
gpt-4-0613	-	68.6	70.2	67.1	69.4	36.4	86.4	67.0	87.1	9.0	71.7

\* Some results are only available with instruction-tuned models.

# Example

## User

Here is a dataset about auto MPG: "https://huggingface.co/datasets/scikit-learn/auto-mpg/raw/main/auto-mpg.csv". You should download and load the dataset from the URL, take a look at it, check for missing value, do a train-test split, and train a regression model predicting its MPG. You should use everything except "mpg" and "car name" as the input features.

## CodeActAgent

To accomplish this task, I will use the pandas library to download and load the dataset, check for missing values, and perform a train-test split. Then, I will use the scikit-learn library to train a regression model.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Download and load the dataset
df = pd.read_csv(
    "https://huggingface.co/datasets/scikit-learn/auto-mpg/raw/main/auto-mpg.csv")
# Check for missing values
print(df.isnull().sum())
# Remove rows with missing values
df = df.dropna()
# Perform train-test split
X = df.drop(columns=["mpg", "car name"])
y = df["mpg"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=42)

# Train a regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Predict MPG for test data
y_pred = model.predict(X_test)
# Evaluate the model
print("Mean squared error: ", np.mean((y_test - y_pred) ** 2))
print("R^2 score: ", r2_score(y_test, y_pred))
```

Use Pandas Library to Download, Examine, and Process Data

Use Scikit-Learn Library to Split Data and Train Regression Model

## Environment

```
mpg      0
...omitted for space...
car name 0
dtype: int64

ValueError                                Traceback (most recent call last)
~/mp/spykernel_12/1373581217.py in <module>
    18 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2 [...],
    19
    20 # Train a regression model
    21 model = LinearRegression()
--> 22 model.fit(X_train, y_train)
    23
    24 # Predict MPG for test data
    25 y_pred = model.predict(X_test)

... [omitted for space] ...

ValueError: could not convert string to float: '?'
```

Self-Debug from Automated Feedback

## CodeActAgent

It seems that there is a data type issue in the dataset. The '?' character is present in the data, which is causing the error when trying to convert the data to a float. To fix this, I will remove the rows containing the '?' character.

```
# Remove rows containing '?'
df = df.replace('?', np.nan)
df = df.dropna()
X = df.drop(columns=["mpg", "car name"])
y = df["mpg"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Mean squared error: ", np.mean((y_test - y_pred) ** 2))
print("R^2 score: ", r2_score(y_test, y_pred))
```

## Environment

Mean squared error: 10.718864418638367  
R^2 score: 0.790158038676035

## CodeActAgent

The mean squared error is 10.711 and the R^2 score is 0.790, which indicates that the model has a good fit with the data [...]

## User

Can you also give me the MSE and R^2 for the training set?

## CodeActAgent

Certainly! Here are the mean squared error (MSE) and R^2 scores [...]

```
from sklearn.metrics import mean_squared_error
# Calculate MSE for training set
mse_train = mean_squared_error(y_train, model.predict(X_train))
# Calculate R^2 score for training set
r2_train = r2_score(y_train, model.predict(X_train))

print("Mean squared error (training set):", mse_train)
print("R^2 score (training set):", r2_train)
```

## Environment

Mean squared error (training set): 10.979255842373478  
R^2 score (training set): 0.826801578671067

## CodeActAgent

The mean squared error for the training set is 10.979 and the R^2 score is 0.826, which indicates [...]

## User

Can you visualize the regression coefficient? You may access them via model.coef\_ and their names via model.feature\_names\_in\_

## CodeActAgent

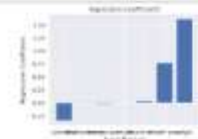
Certainly! Here is a visualization of the regression coefficients [...]

```
import matplotlib.pyplot as plt
# Get regression coefficients
coef = model.coef_
# Get names of input features
feature_names = model.feature_names_in_
# Plot regression coefficients
plt.bar(feature_names, coef)
plt.xlabel('Input Feature')
plt.ylabel('Regression Coefficient')
plt.title('Regression Coefficients')
# Display the plot
plt.show()
```

Use Matplotlib Library for Data Visualization

## Environment

<Figure size 640x480 with 1 Axes>



## CodeActAgent

The regression coefficients are plotted as a bar chart, with each input feature on the x-axis and the corresponding coefficient on the y-axis [...]

## User

Can you rotate the x ticks by 45 degrees for better readability?

## CodeActAgent

Certainly! Here is the updated bar chart with the x-ticks rotated by 45 degrees for better readability:

```
[...same code omitted...]
# Plot regression coefficients
plt.bar(feature_names, coef, tick_labels=False)
plt.xticks(feature_names, feature_names,
           rotation=45, ha='right', fontsize=12)
[...same code omitted...]
```

## Environment

```
<Figure size 640x480 with 1 Axes>

AttributeError                                Traceback (most recent call last)
Cell In[1], line 10
     7 feature_names = model.feature_names_in_
     9 # Plot regression coefficients
--> 10 plt.bar(feature_names, coef, tick_labels=False)
    11 plt.xticks(feature_names, feature_names,
              rotation=45, ha='right', fontsize=12)
    12 plt.xlabel('Input Feature')
[...]
```

Self-Debug for Data Visualization

## CodeActAgent

It seems that the tick\_labels argument is not a valid argument for the bar() function in Matplotlib. Instead, we can use the tick\_params() function to rotate the x-ticks [...]

```
[...same code omitted...]
plt.bar(feature_names, coef)
plt.tick_params(axis='x',
               labelrotation=45, ha='right', fontsize=12)
[...same code omitted...]
```

[3 rounds of self-debugging omitted for space]

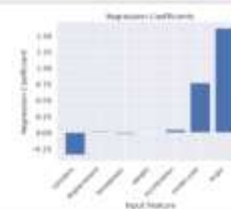
## CodeActAgent

I apologize for the confusion. It seems that the set\_xticks() function is not a valid function in Matplotlib. Instead, we can use the xticks() function to set the x-ticks and rotate them by 45 degrees for better readability. Here is the updated bar chart with the x-ticks rotated by 45 degrees:

```
[...same code omitted...]
plt.xticks(feature_names, feature_names,
           rotation=45, ha='right', fontsize=12)
[...same code omitted...]
```

## Environment

<Figure size 640x480 with 1 Axes>



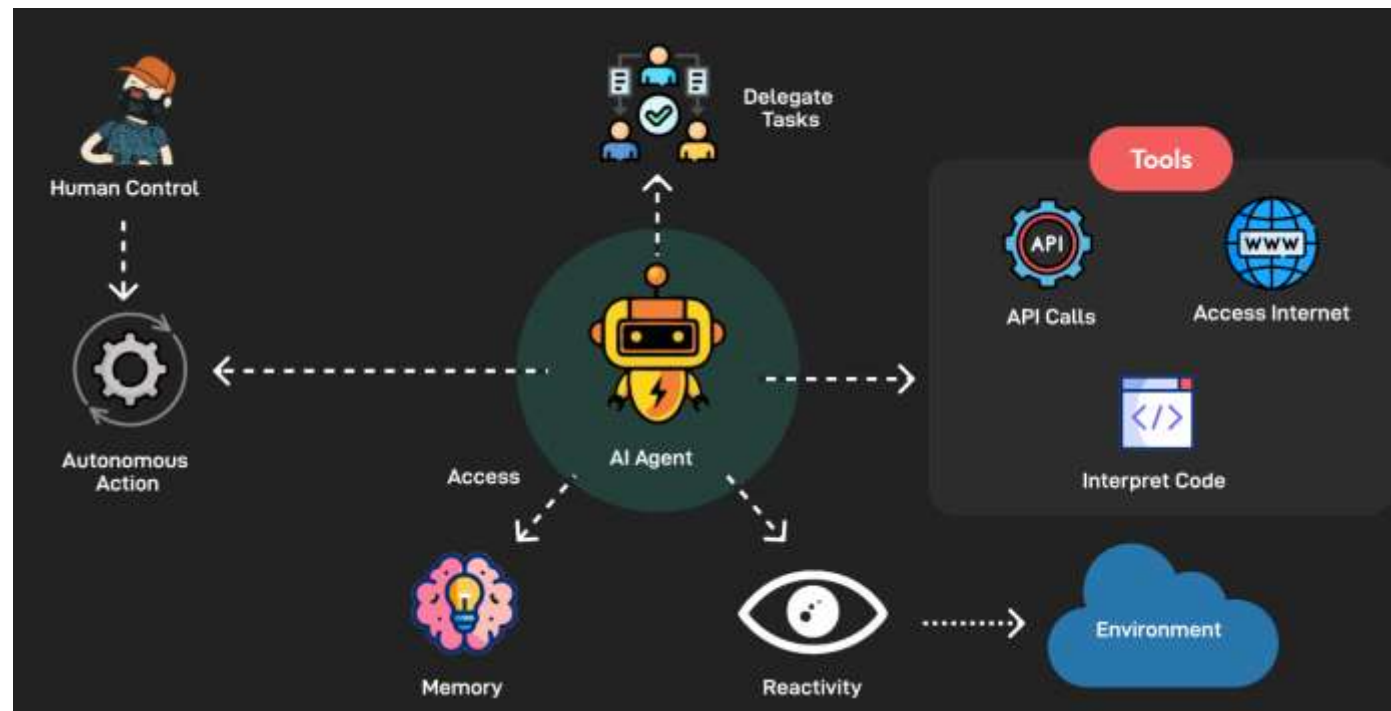
## User

[Stops Interaction]

# Conclusion

## To Build Better Agents

- **Executable Code as Actions:** Executable Python code is a superior choice for agent actions compared to JSON or Text. It leverages the control and data flow inherent in programming, allowing agents to dynamically adjust actions based on observations and feedback from the environment.
- **Data and Fine-Tuning:** Fine-tuning on high-quality, multi-turn interaction data is also crucial.
- **Specifically Designed Prompts:** A well-designed prompt tailored for the agent-environment interaction framework is essential. It guides the agent to effectively use executable code actions and interact with the environment, ensuring seamless integration and task execution.



*Thanks*