

Picking the best AI tool for the job

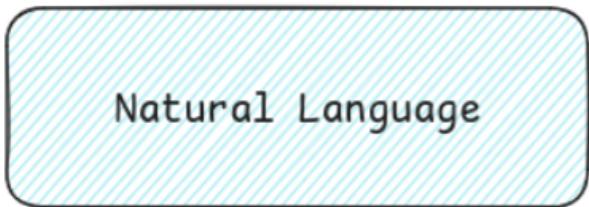
AI-ASSISTED CODING FOR DEVELOPERS



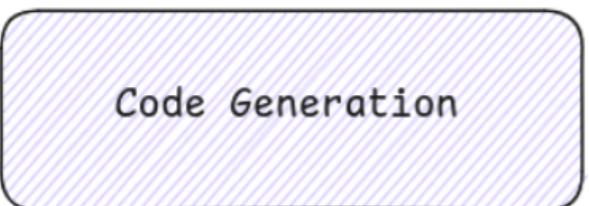
Francesca Donadoni

AI Curriculum Manager, DataCamp

Benchmarks

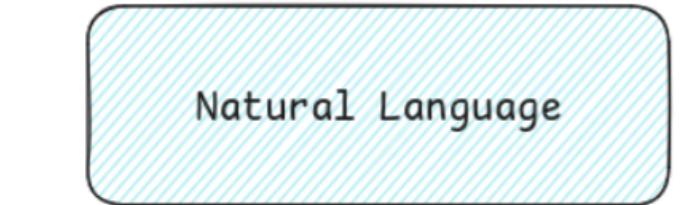


MMLU benchmark
<https://arxiv.org/pdf/2009.03300>

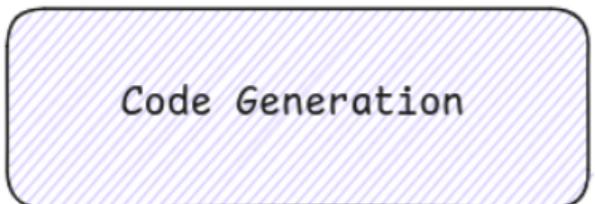


HumanEval benchmark
<https://arxiv.org/pdf/2107.03374>

Benchmarks

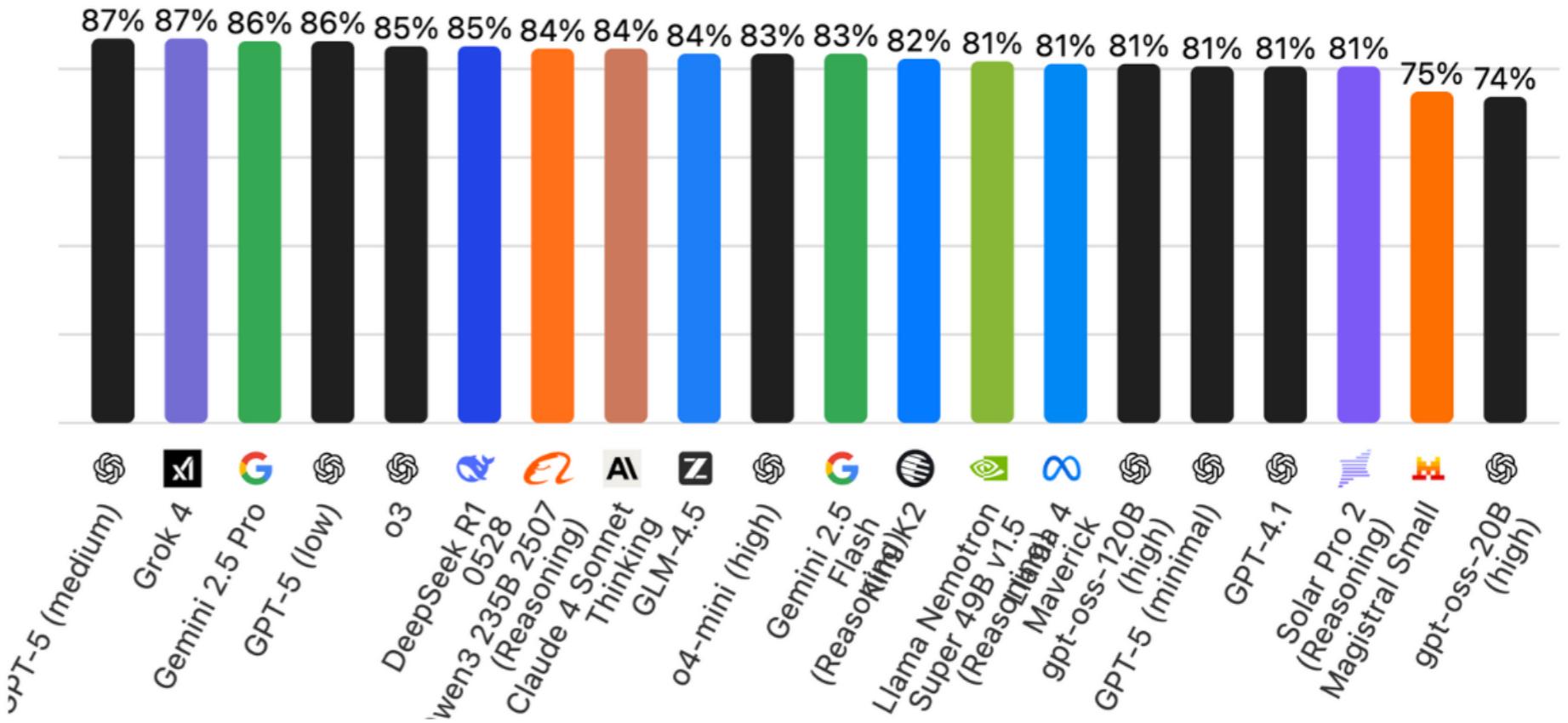


MMLU benchmark
<https://arxiv.org/pdf/2009.03300>



HumanEval benchmark
<https://arxiv.org/pdf/2107.03374>

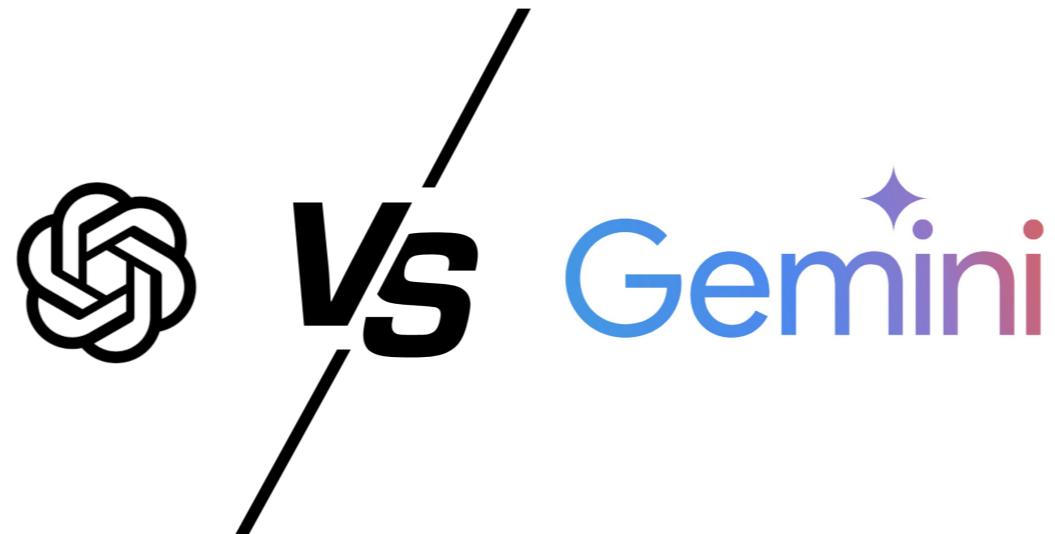
MMLU-Pro (Reasoning & Knowledge)



¹ <https://artificialanalysis.ai/models>

Model comparison

"Generate Python unit tests using pytest for the following function. Include cases for empty input, SQL keywords in the input, and special characters. Assume a mock database connection: <paste function implementation>"



Model comparison

gpt-4.1-2025-04-14

Certainly! Below are `pytest` unit tests for the `insert_user_data_insecure` function.

These tests use the `unittest.mock` library to mock the `sqlite3.connect` method and its returned objects, so no real database is touched.

The tests cover:

- Empty input
- SQL keywords in input
- Special characters in input

Python

```
import pytest
from unittest.mock import patch, MagicMock

from your_module import insert_user_data_insecure # Replace with
your actual module name

@pytest.fixture
def mock_sqlite():
```

gemini-2.5-pro

Of course. Here are the pytest unit tests for the provided function.

First, let's place the function you provided into a file named `vulnerable_app.py`.

Python

```
# vulnerable_app.py

import sqlite3

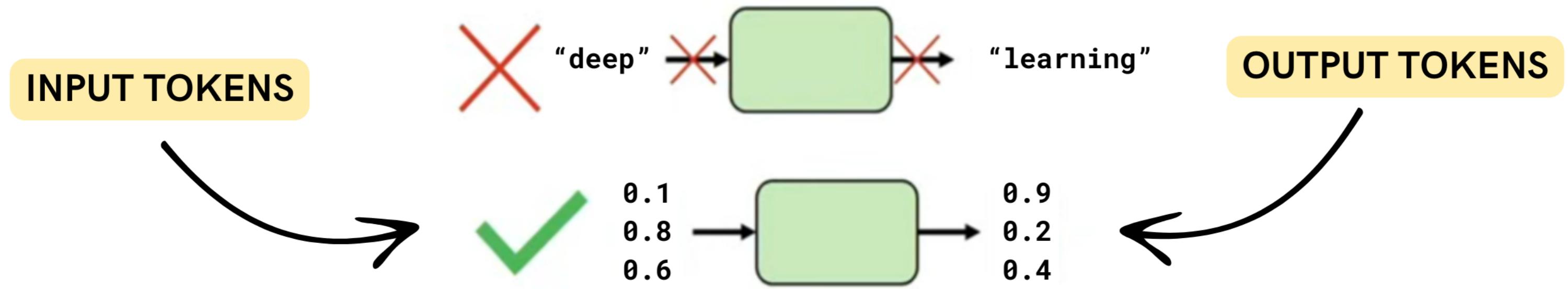
def insert_user_data_insecure(db_path, form_data):
    """
    Inserts user data into the database using an insecure method
    (f-string formatting), which is vulnerable to SQL injection.
    """

    name = form_data.get('name', '')
    email = form_data.get('email', '')
    message = form_data.get('message', '')

    try:
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
```

Tokenization

- LLMs work on numerical data (tokens)



Cost



Token Cost Breakdown (model: gpt)
Input tokens: 146 → \$0.000292

Gemini

Token Cost Breakdown (model: gemini)
Input tokens: 146 → \$0.000182

	GPT	Gemini
Input tokens	\$2 / M tokens	\$1.25 / M tokens
Output tokens	\$8 / M tokens	\$10 / M tokens

Cost



Token Cost Breakdown (model: gpt)
Input tokens: 146 → \$0.000292
Output tokens: 681 → \$0.005448

Total cost: \$0.005740



Token Cost Breakdown (model: gemini)
Input tokens: 146 → \$0.000182
Output tokens: 2079 → \$0.020790

Total cost: \$0.020973

	GPT	Gemini
Input tokens	\$2 / M tokens	\$1.25 / M tokens
Output tokens	\$8 / M tokens	\$10 / M tokens

Reasoning models



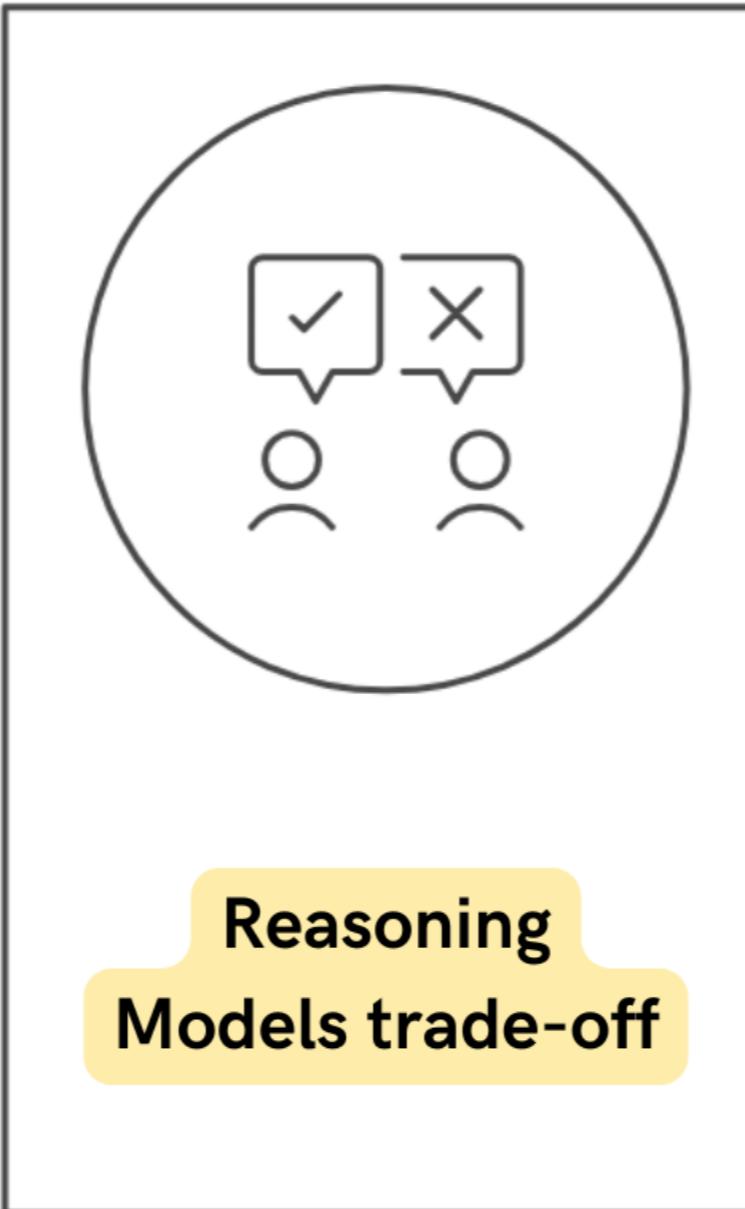
Transparency



Accuracy in the responses



Insights into the model



Higher cost



System interference



Slower response



Output quality



GPT Output

```
Traceback (most recent call last):
  File "/home/avalenz1/datacamp/model-comparison/gpt-model.py", line 4, in <module>
    from your_module import insert_user_data_insecure # Replace with your actual module name
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'your_module'
```

Gemini Output

Gemini

```
===== test session starts =====
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0 -- /home/avalenz1/anaconda3/bin/python
cachedir: .pytest_cache
rootdir: /home/avalenz1/datacamp/model-comparison/gemini
plugins: mock-3.14.1, anyio-4.9.0
collected 4 items

test_vulnerable_app.py::test_insert_user_data_standard PASSED [ 25%]
test_vulnerable_app.py::test_insert_empty_input PASSED [ 50%]
test_vulnerable_app.py::test_insert_with_special_characters FAILED [ 75%]
test_vulnerable_app.py::test_insert_with_sql_injection_attack PASSED [100%]
```

Output quality



GPT Output

```
Traceback (most recent call last):
  File "/home/avalenz1/datacamp/model-comparison/gpt-model.py", line 4, in <module>
    from your_module import insert_user_data_insecure # Replace with your actual module name
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'your_module'
```

Not an end-to-end solution

Gemini Output



```
===== test session starts =====
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0 -- /home/avalenz1/anaconda3/bin/python
cachedir: .pytest_cache
rootdir: /home/avalenz1/datacamp/model-comparison/gemini
plugins: mock-3.14.1, anyio-4.9.0
collected 4 items

test_vulnerable_app.py::test_insert_user_data_standard PASSED [ 25%]
test_vulnerable_app.py::test_insert_empty_input PASSED [ 50%]
test_vulnerable_app.py::test_insert_with_special_characters FAILED [ 75%]
test_vulnerable_app.py::test_insert_with_sql_injection_attack PASSED [100%]
```

Output quality



GPT Output

```
Traceback (most recent call last):
  File "/home/avalenz1/datacamp/model-comparison/gpt-model.py", line 4, in <module>
    from your_module import insert_user_data_insecure # Replace with your actual module name
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'your_module'
```

Not an end-to-end solution

Gemini Output



```
===== test session starts =====
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0 -- /home/avalenz1/anaconda3/bin/python
cachedir: .pytest_cache
rootdir: /home/avalenz1/datacamp/model-comparison/gemini
plugins: mock-3.14.1, anyio-4.9.0
collected 4 items
```

**More expensive, but a
more complete solution**

```
test_vulnerable_app.py::test_insert_user_data_standard PASSED
[ 25%]
test_vulnerable_app.py::test_insert_empty_input PASSED
[ 50%]
test_vulnerable_app.py::test_insert_with_special_characters FAILED
[ 75%]
test_vulnerable_app.py::test_insert_with_sql_injection_attack PASSED
[100%]
```

Let's practice!

AI-ASSISTED CODING FOR DEVELOPERS

Guardrails for responsible AI coding

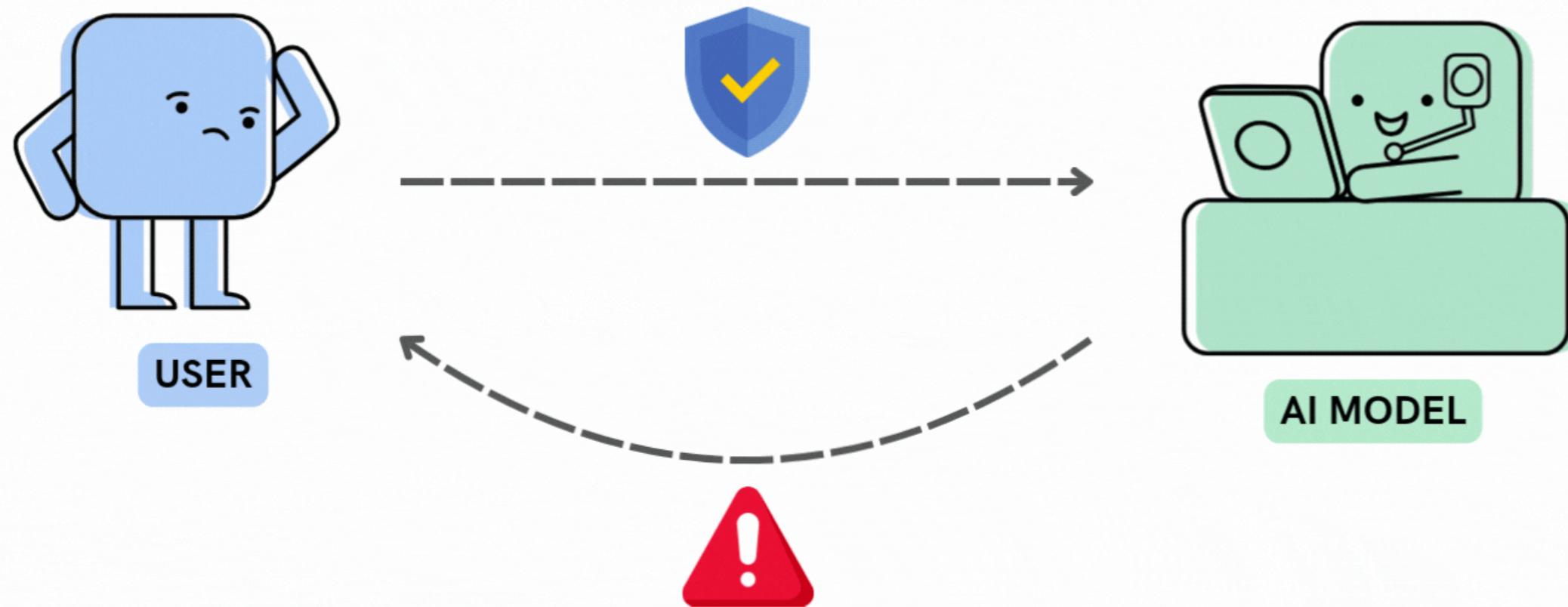
AI-ASSISTED CODING FOR DEVELOPERS



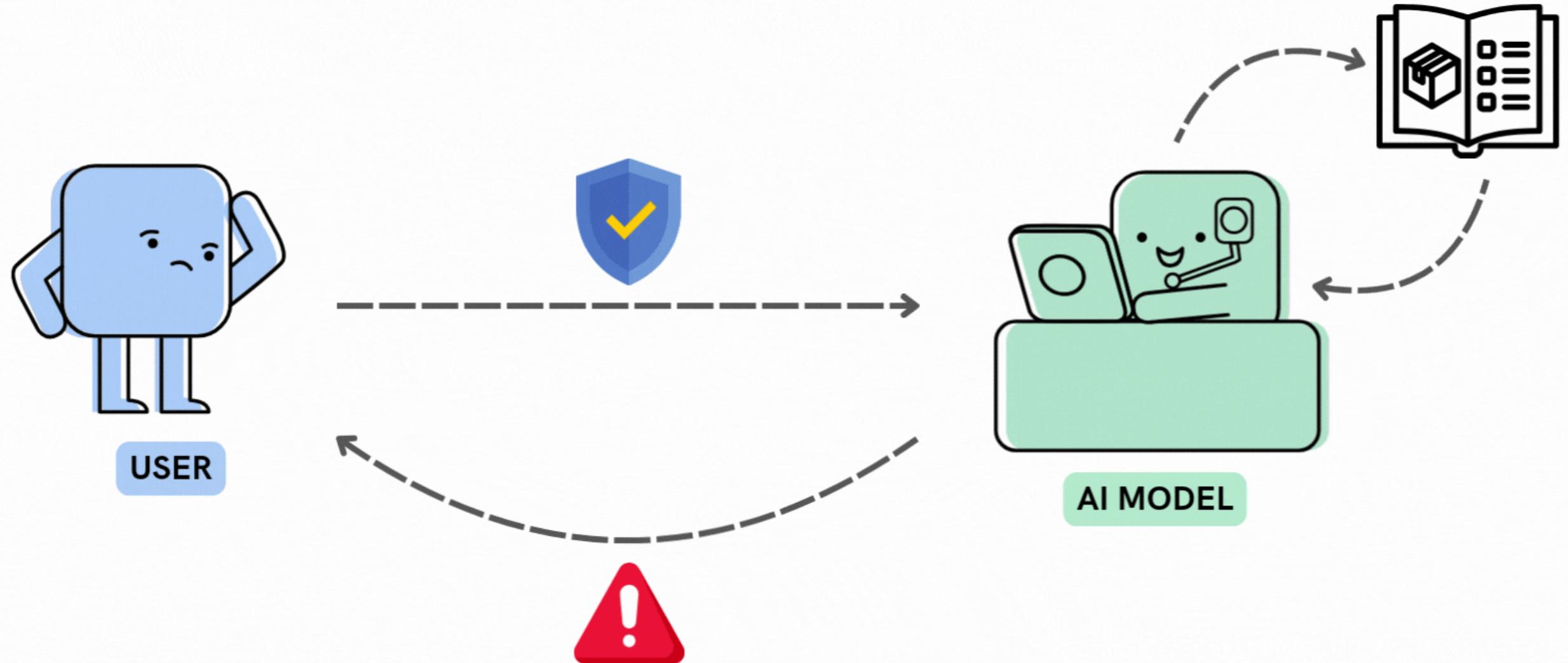
Francesca Donadoni

AI Curriculum Manager, DataCamp

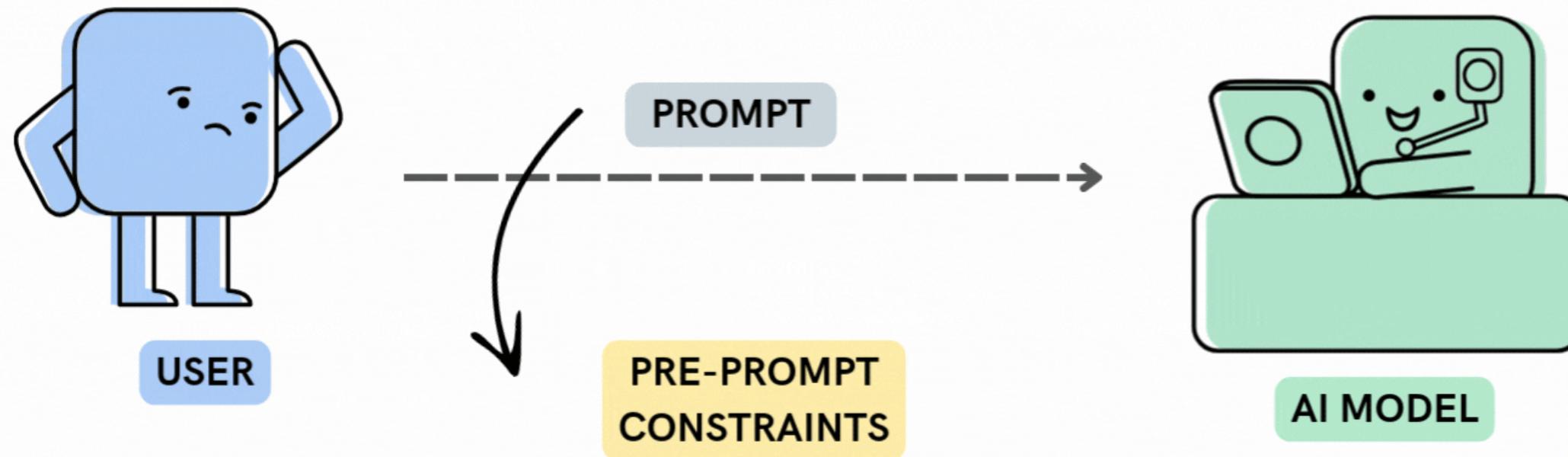
LLM-powered workflows



LLM-powered workflows



Pre-prompt constraints



Pre-prompt constraints

□ Message constraints

- System messages
- Structure, tone and limits
- Few-shot examples

□ Ethical constraints

- "*Avoid generating unsafe code*"
- "*Do not provide scripts that bypass authentication or scrape private data*"
- "*Only return examples that follow open-source licenses*"

Pre-prompt constraints

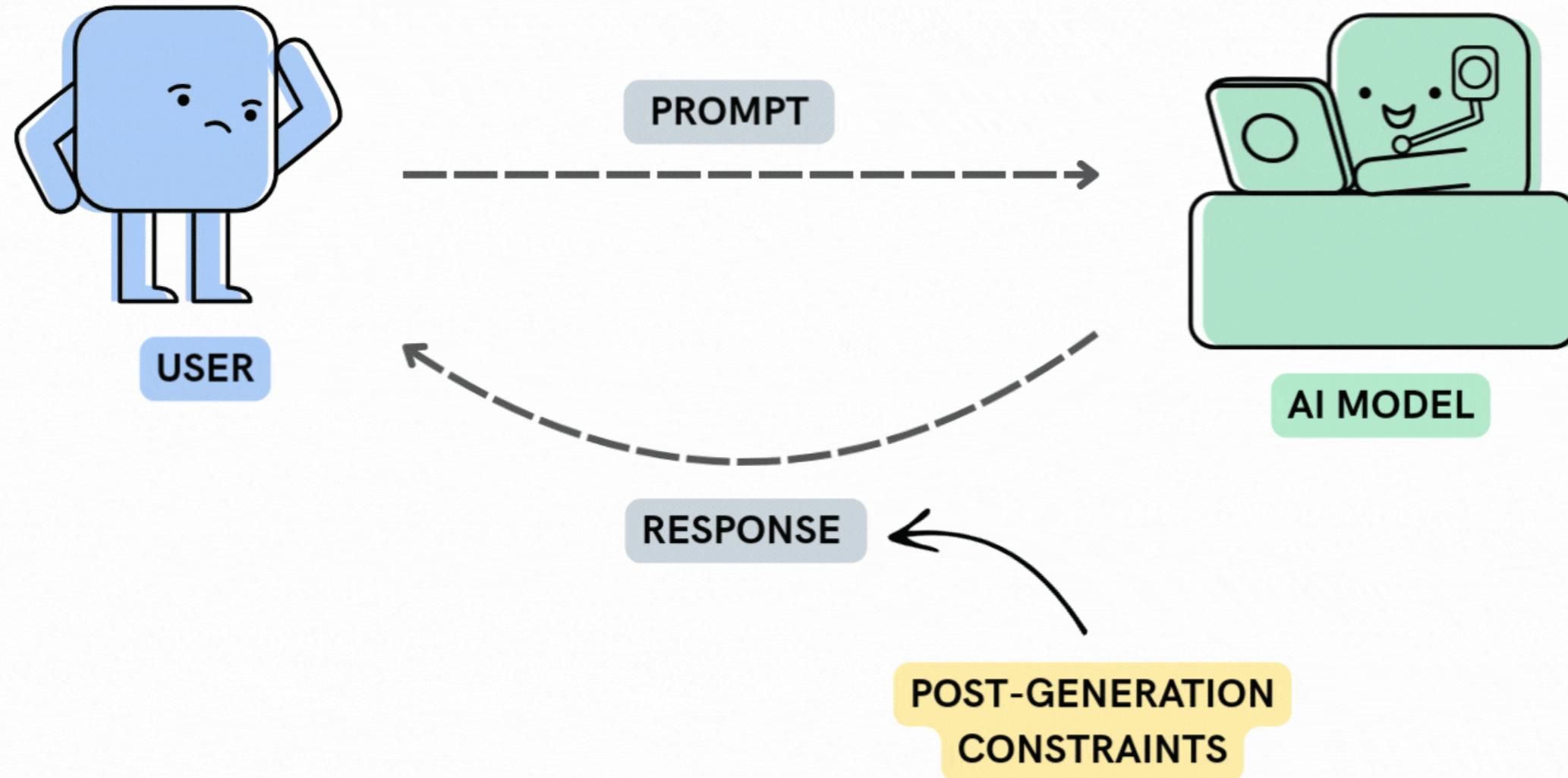
Secure Prompt

"Write a script to download images from an open-access public gallery"

Malicious Prompt

"Write a script to download images from a gallery that restricts automated access"

Post-generation constraints

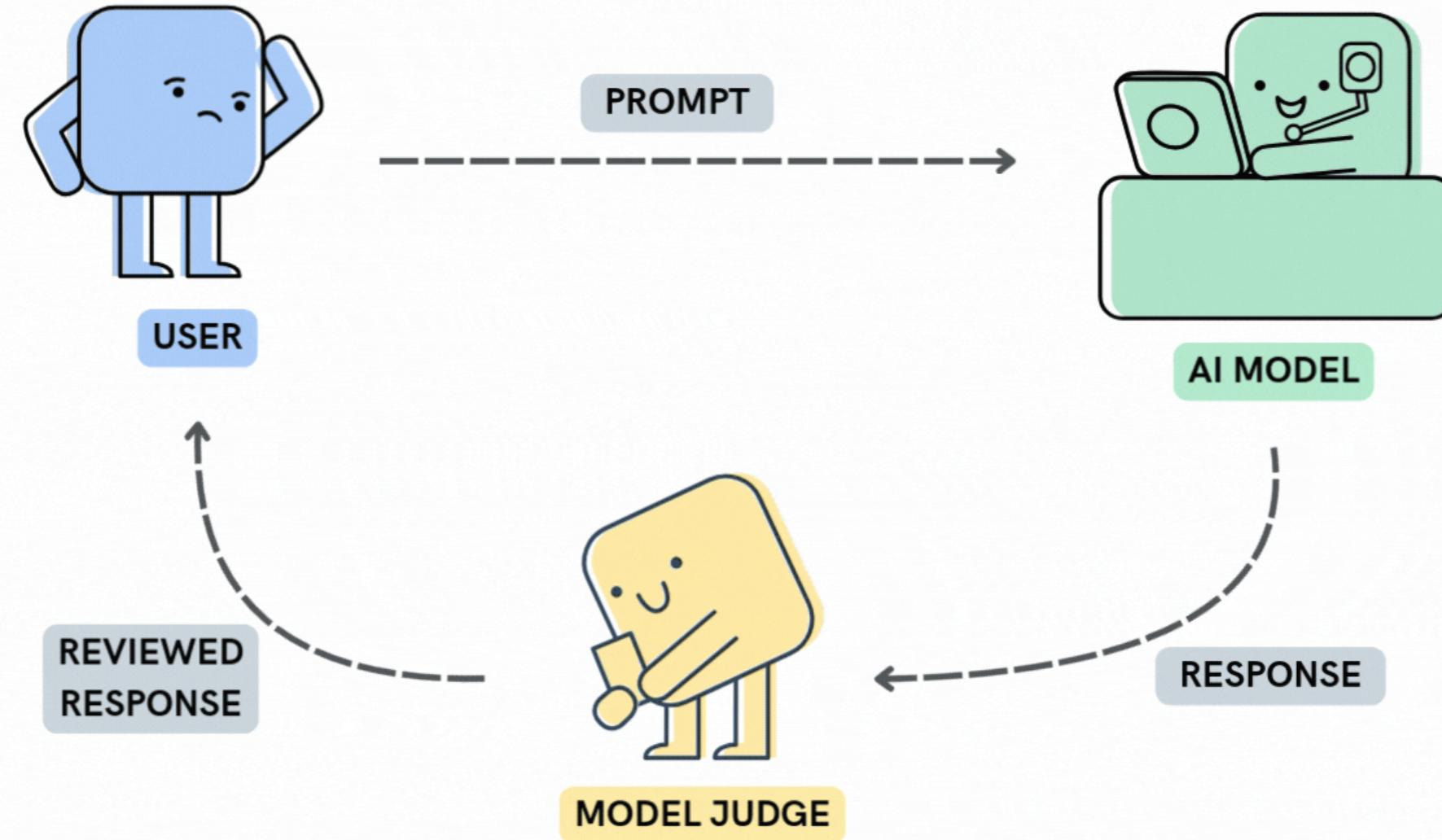


Post-generation constraints

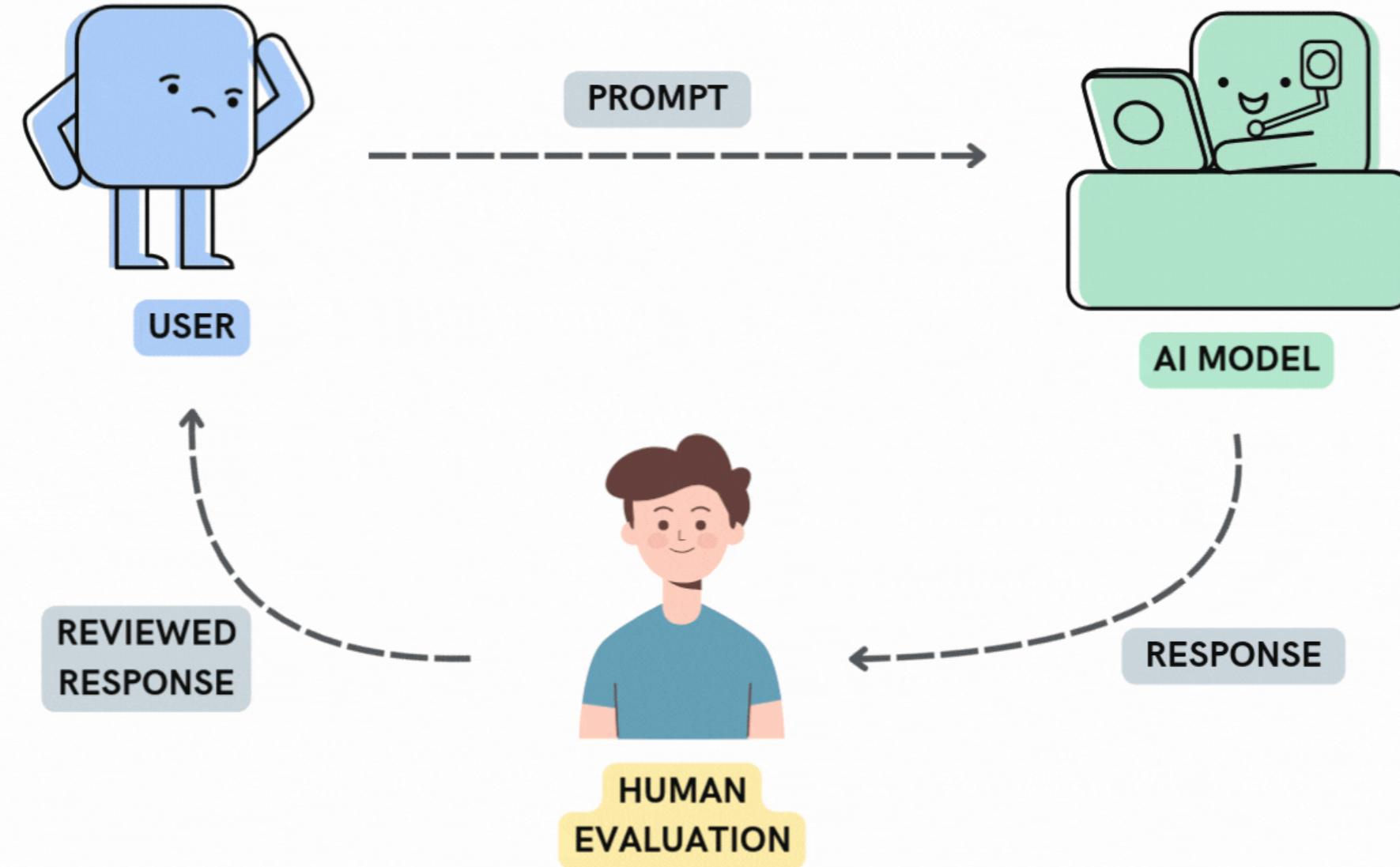
□ Output constraints

- Output validation
- Content filters
- Evaluation functions

Post-generation constraints



Post-generation constraints



Let's practice!

AI-ASSISTED CODING FOR DEVELOPERS

Optimizing AI for speed, cost and quality

AI-ASSISTED CODING FOR DEVELOPERS



Francesca Donadoni

AI Curriculum Manager, DataCamp

Metrics



- **Latency** (response time): How long it takes for the model to generate a response

Metrics



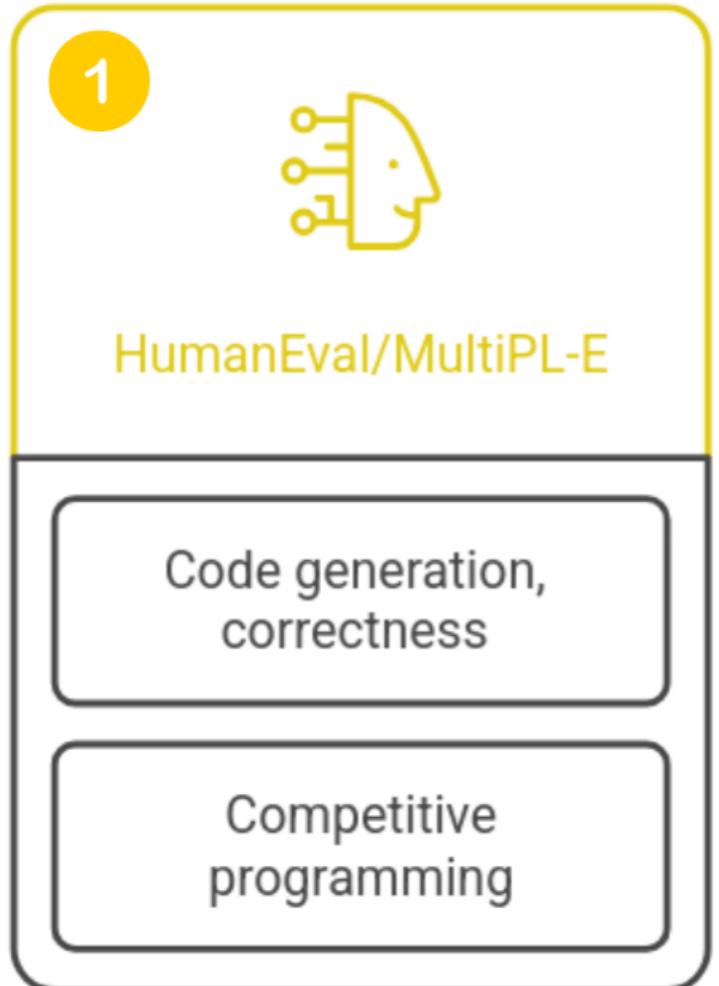
- **Latency** (response time): How long it takes for the model to generate a response
- **Token cost** (per 1M tokens): Monetary expense of using an AI model

Metrics



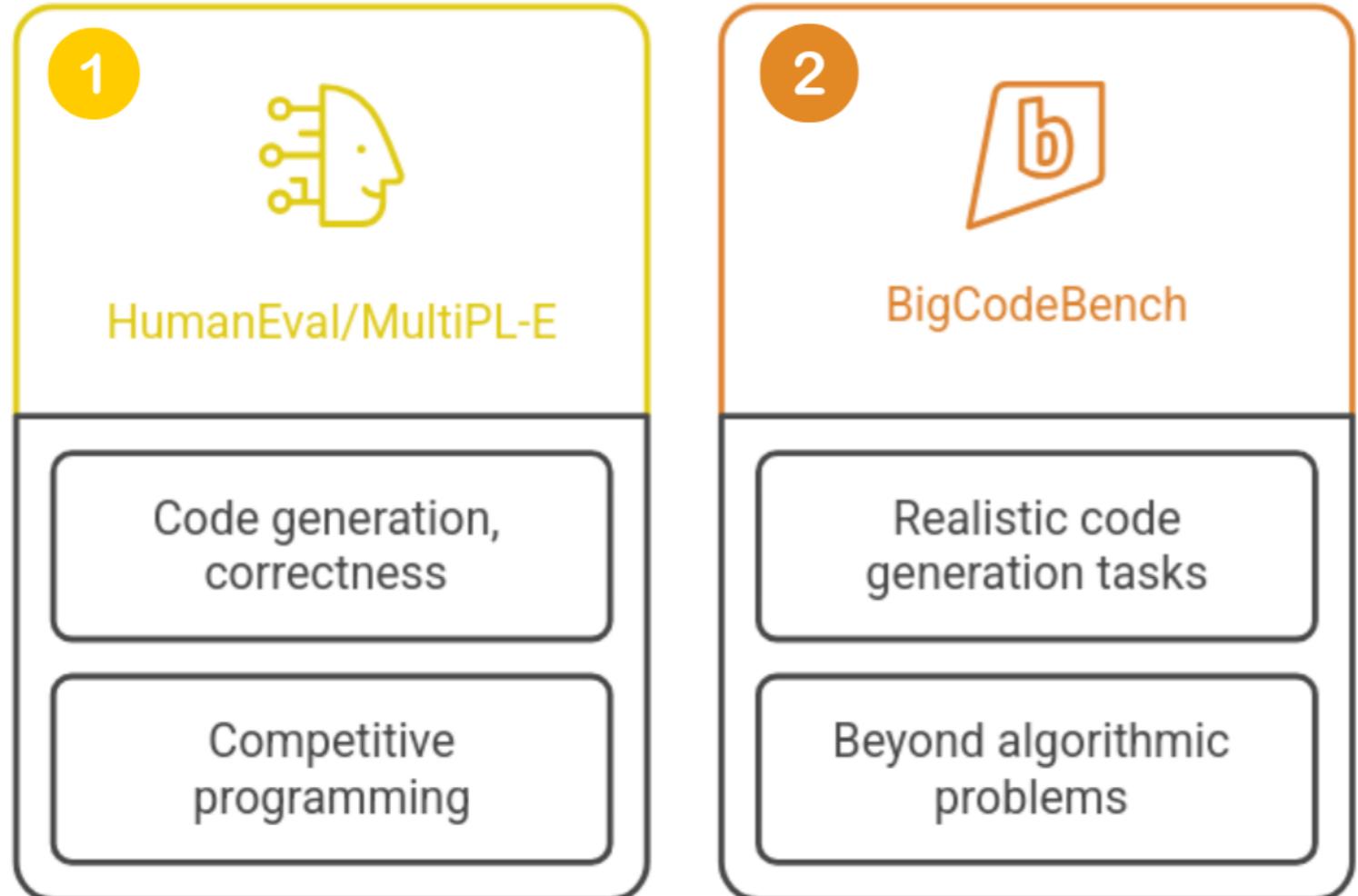
- **Latency** (response time): How long it takes for the model to generate a response
- **Token cost** (per 1M tokens): Monetary expense of using an AI model
- **Quality** (pass rate on tests/coverage): How often the generated code works as intended

Model benchmarking



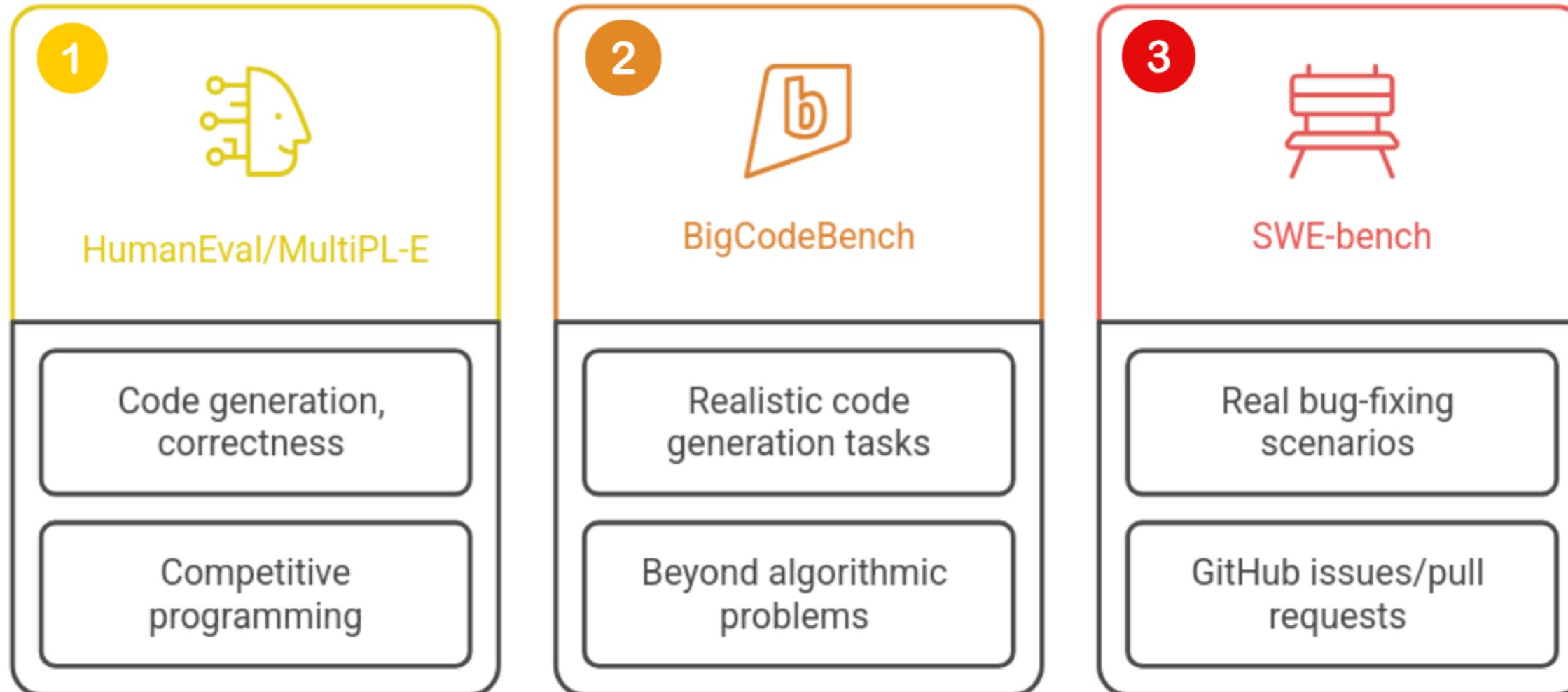
¹ <https://github.com/openai/human-eval>

Model benchmarking



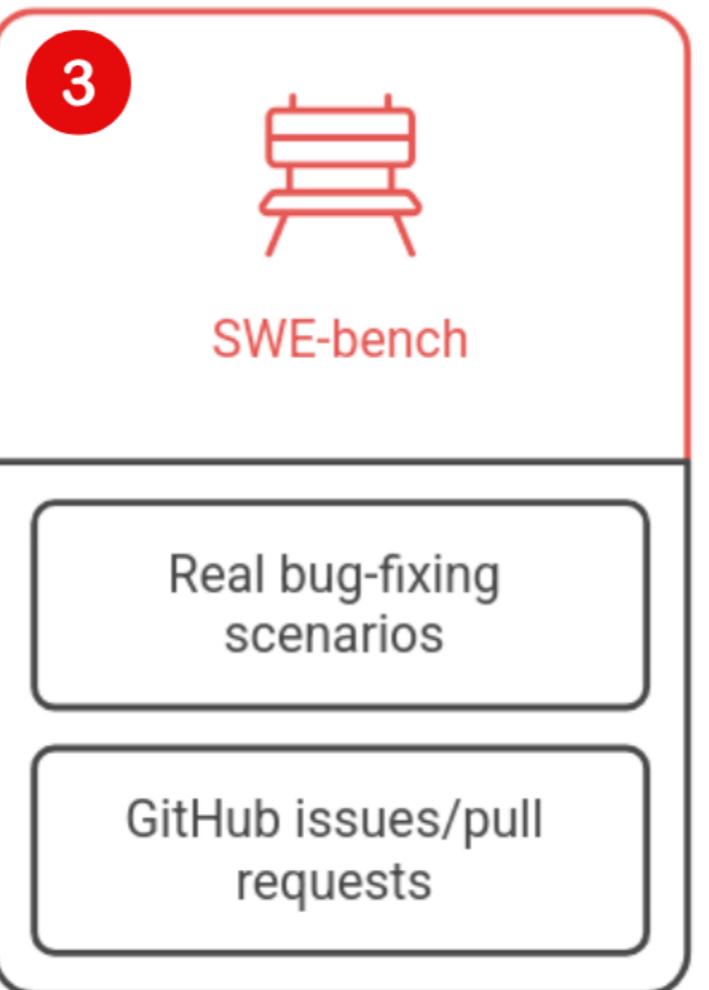
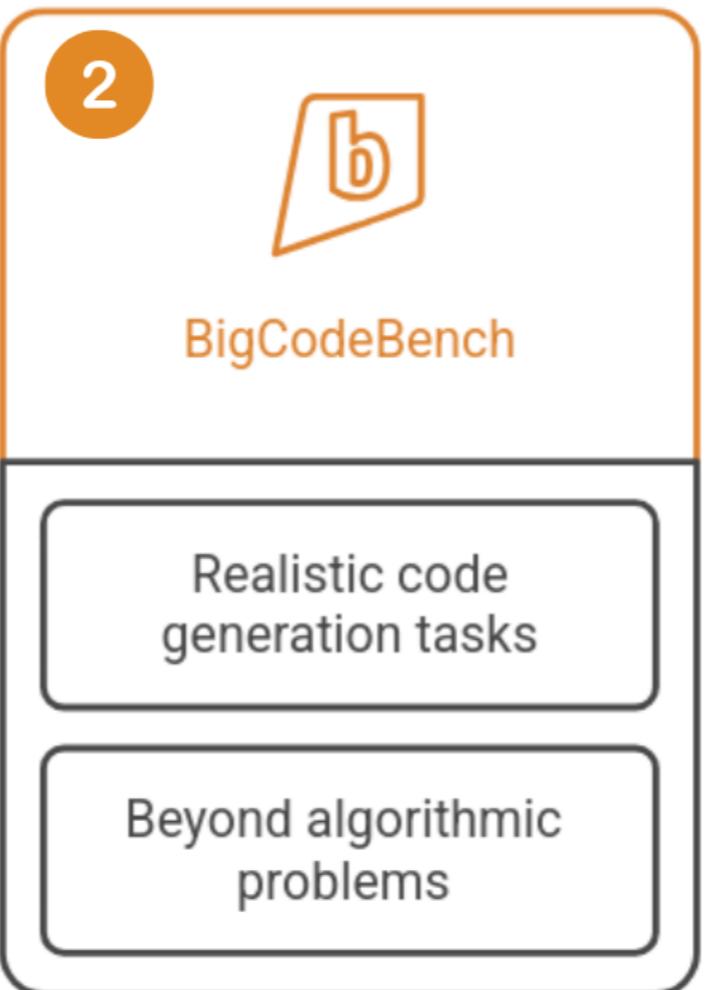
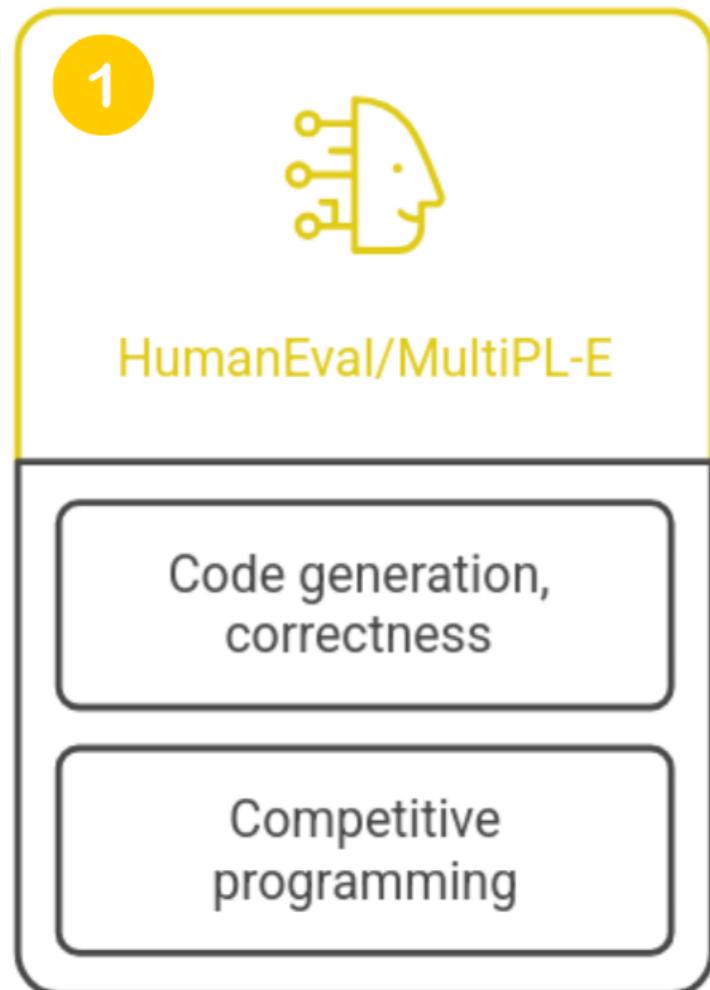
¹ <https://github.com/bigcode-project/bigcodebench>

Model benchmarking



¹ <https://github.com/SWE-bench/SWE-bench>

Model benchmarking



¹ <https://github.com/JohnnyPeng18/Coffe>

Prompt versioning

```
{  
    "id": "generate_average_function",  
    "version": "v2.1",  
    "template": "Write a Python function that takes {{input_type}} and returns  
{{output_description}}. Make sure to {{instruction}}.",  
    "variables": {  
        "input_type": "a list of numbers",  
        "output_description": "the average as a float",  
        "instruction": "include a docstring and handle empty lists by raising a ValueError"  
    }  
}
```



Prompt versioning

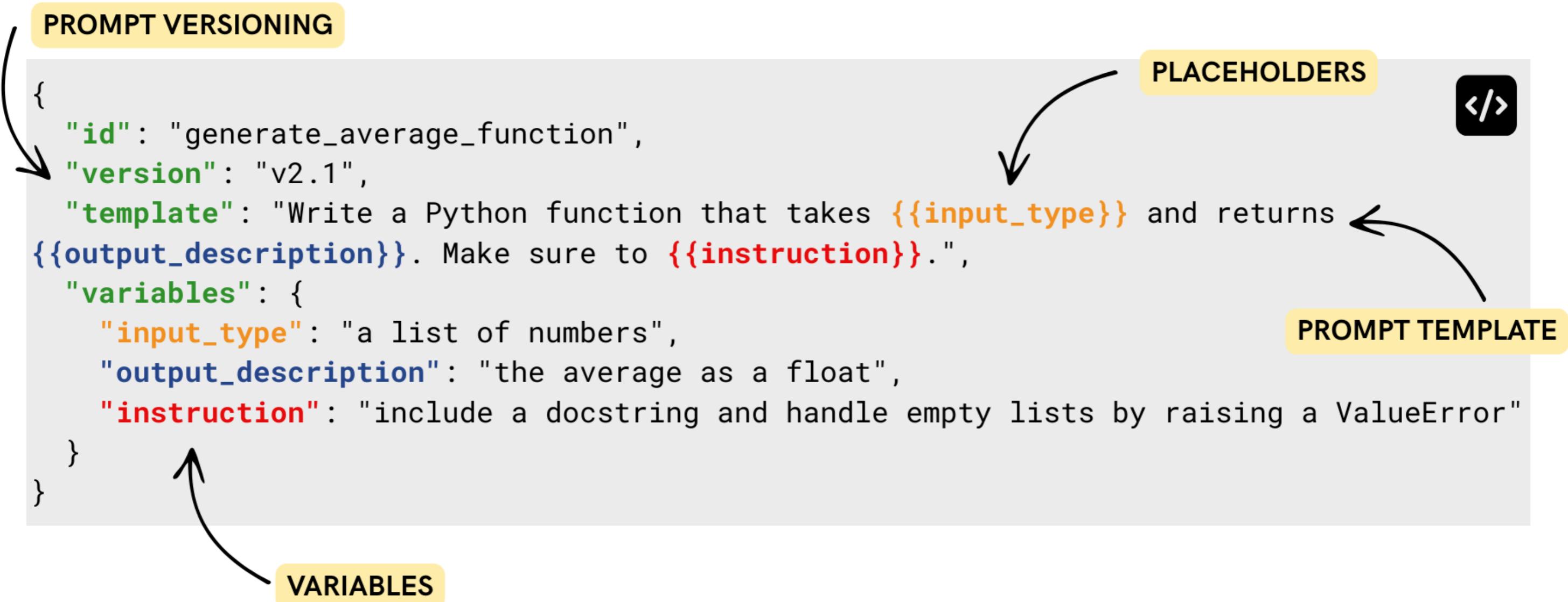
PROMPT VERSIONING

```
{  
    "id": "generate_average_function",  
    "version": "v2.1",  
    "template": "Write a Python function that takes {{input_type}} and returns  
{{output_description}}. Make sure to {{instruction}}.",  
    "variables": {  
        "input_type": "a list of numbers",  
        "output_description": "the average as a float",  
        "instruction": "include a docstring and handle empty lists by raising a ValueError"  
    }  
}
```

</>

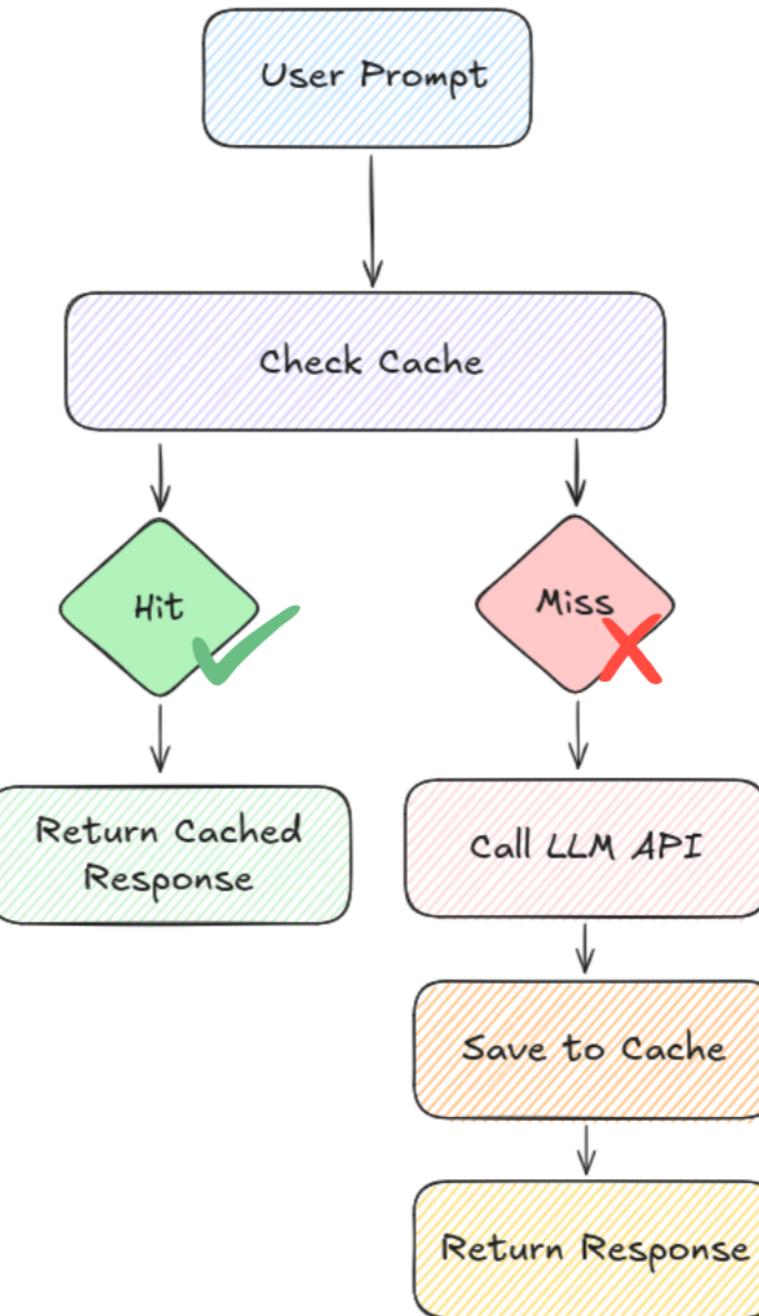
PROMPT TEMPLATE

Prompt versioning



Prompt caching

- Caching works by storing:
 - Prompt
 - Input
 - Model
 - Temperature



Let's practice!

AI-ASSISTED CODING FOR DEVELOPERS

Congratulations!

AI-ASSISTED CODING FOR DEVELOPERS



Francesca Donadoni

AI Curriculum Manager, DataCamp

Recap

Recap

- Deep reasoning:
o-series, Claude Opus
or Gemini Ultra.
- Quick assistance:
o-mini or
Gemini Flash.
- General coding:
GPT or
Claude Sonnet.

Let's practice!

Components of a Prompt

INSTRUCTION

- Fix this Python function

CONTEXT

- to handle division by zero errors.

CONSTRAINT

- Keep the structure the same, and only change the except block.

CHAPTER #1

Recap

The collage consists of four cards:

- Recap**: Shows three icons: a person thinking (Deep reasoning), a person at a computer (Quick assistance), and a person at a computer (General coding). Text: "Deep reasoning: o-series, Claude Opus or Gemini Ultra.", "Quick assistance: o-mini or Gemini Flash.", "General coding: GPT or Claude Sonnet.". A button says "Let's practice!".
- Reasoning Models**: Features the **deepseek** logo. Text: "Reasoning models perform Chain-of-Thought reasoning by default, self-verify intermediate steps and iterate over their own reasoning." An inset shows a screenshot of the deepseek interface with a play button and a text input field: "Write a Python function that checks if a given string is a palindrome".
- Chain-of-Thought**: Shows a question: "Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?". It shows a step-by-step reasoning process: "A: Let's think step by step.", "(Output) 8 X", "A: That means there are 8 golf balls. Half of the golf balls are blue. That means there are 4 blue golf balls. ✓". A callout notes: "CoT forces the model to explicitly compute the intermediate steps, leading to the correct response".
- OWASP Top 10**: A circular diagram showing the ten most critical web application security risks: Server-Side Request Forgery, Broken Access Control, Insufficient Logging, Cryptographic Failures, Data Integrity Failures, Injection, Authentication Failures, Insecure Design, Outdated Components, and Security Misconfiguration.

CHAPTER #1

CHAPTER #2

Recap

Recap

Deep reasoning:
o-series, Claude Opus
or Gemini Ultra.

Quick assistance:
o-mini or
Gemini Flash.

General coding:
GPT or
Claude Sonnet.

Let's practice!

Reasoning Models

Reasoning models perform Chain-of-Thought reasoning by default, self-verify intermediate steps and iterate over their own reasoning.

Chain-of-Thought

#1st CHAIN-OF-THOUGHT APPROACH

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: Let's think step by step.
(Output) There are 16 balls in total. Half of the balls are golf balls. That means there are 8 golf balls. Half of the golf balls are blue. That means there are 4 blue golf balls. ✓

(Output) 8 X

CoT forces the model to explicitly compute the intermediate steps, leading to the correct response

Post-Generation Constraints

LLM-as-a-Judge

Prompt Versioning

Tagging & storing best-performing prompts

PROMPT VERSIONING

```
    {  
        "id": "generate_average_function",  
        "version": "v2.1",  
        "template": "Write a Python function that takes {{input_type}} and returns {{output_description}}. Make sure to {{instruction}}.",  
        "variables": {  
            "input_type": "a list of numbers",  
            "output_description": "the average as a float",  
            "instruction": "include a docstring and handle empty lists by raising a ValueError"  
        }  
    }
```

PROMPT TEMPLATE

PLACEHOLDERS

VARIABLES

Components of a Prompt

INSTRUCTION
Fix this Python function

CONTEXT
to handle division by zero errors.

CONSTRAINT
Keep the structure the same, and only change the except block.

OWASP Top 10

OWASP Top 10

CHAPTER #3

CHAPTER #1

CHAPTER #2

Thank you!

AI-ASSISTED CODING FOR DEVELOPERS