

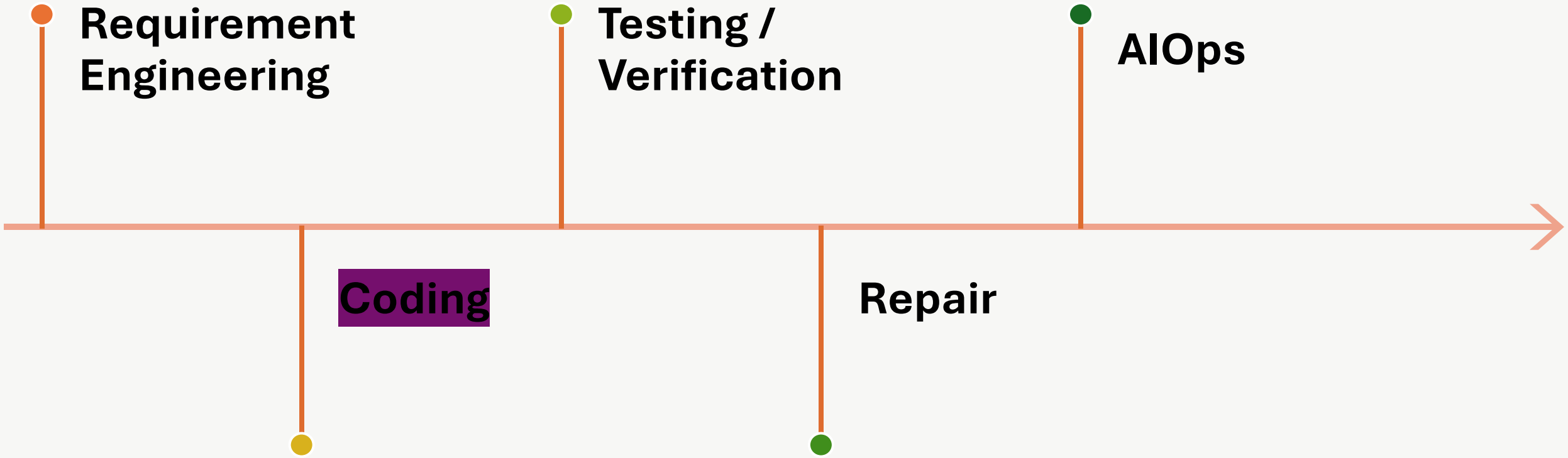
香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC4801
AI-assisted Software Engineering

Lecture 09: Coding II

Instructor: Jinsheng Ba | 2026 Fall

AI-Assisted Software Engineering



Three Pillars



Model



Harness



Benchmarks

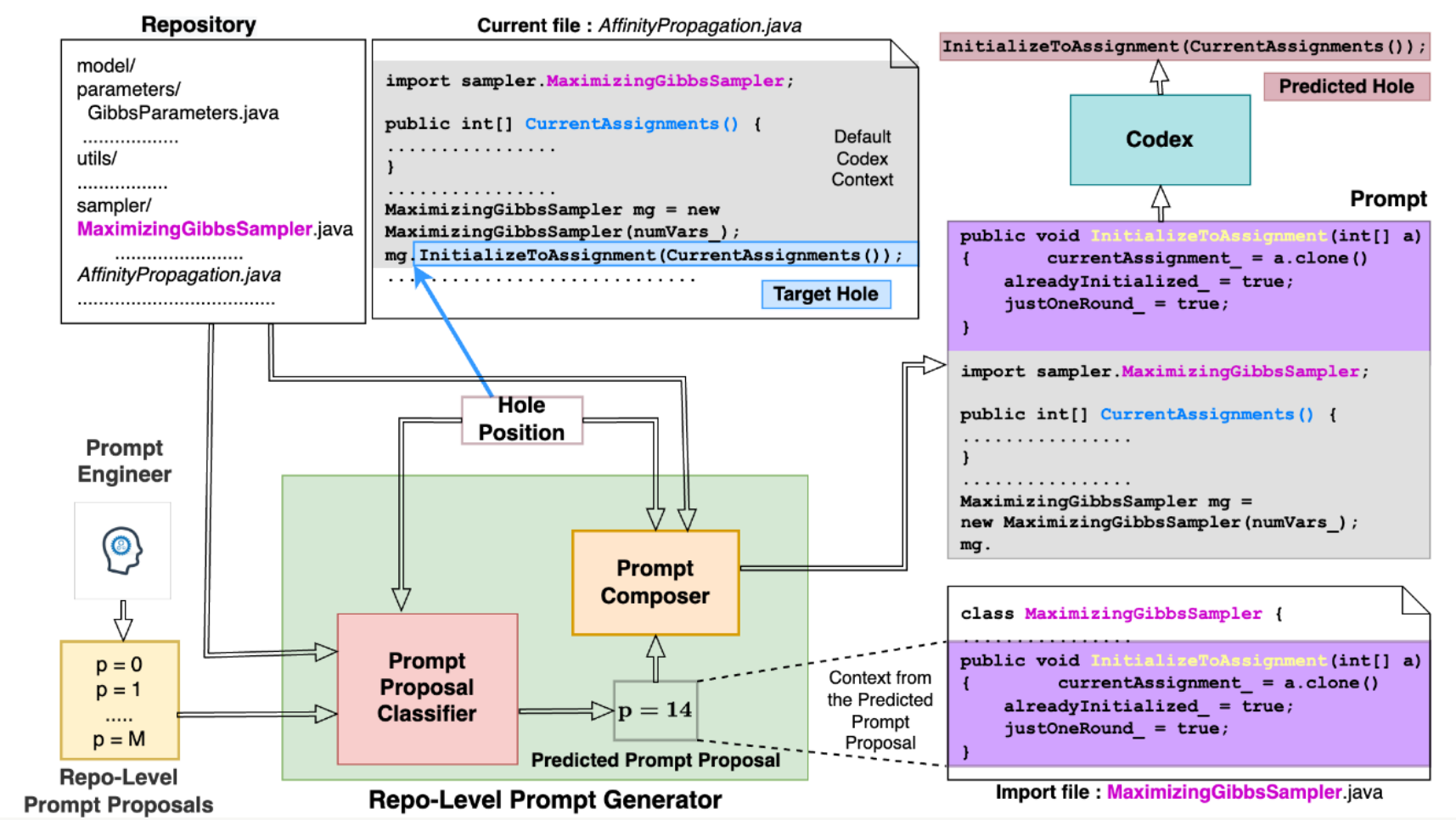
How to tackle repository level and long context code?

Challenges

- Too long context
- File dependencies
  - Shared utilities, configurations
- Repo structures
  - Naming conventions
  - Coding styles

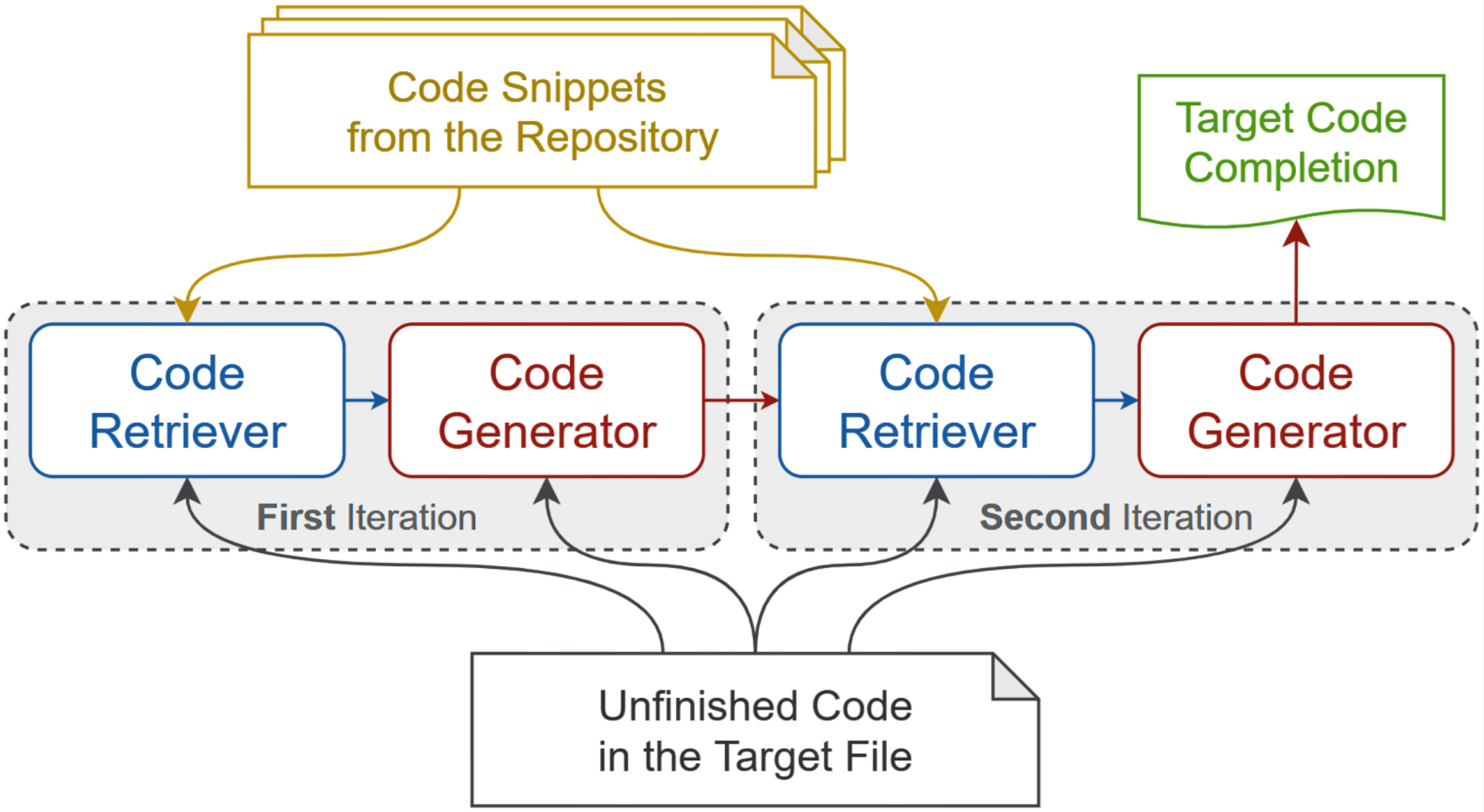
Repo-Level Prompt Generation (RAG)

- It generates repo structure with the
- RAG is used.



https://arxiv.org/pdf/2206.12839

RepoCoder: Iterative Retrieval and Generation

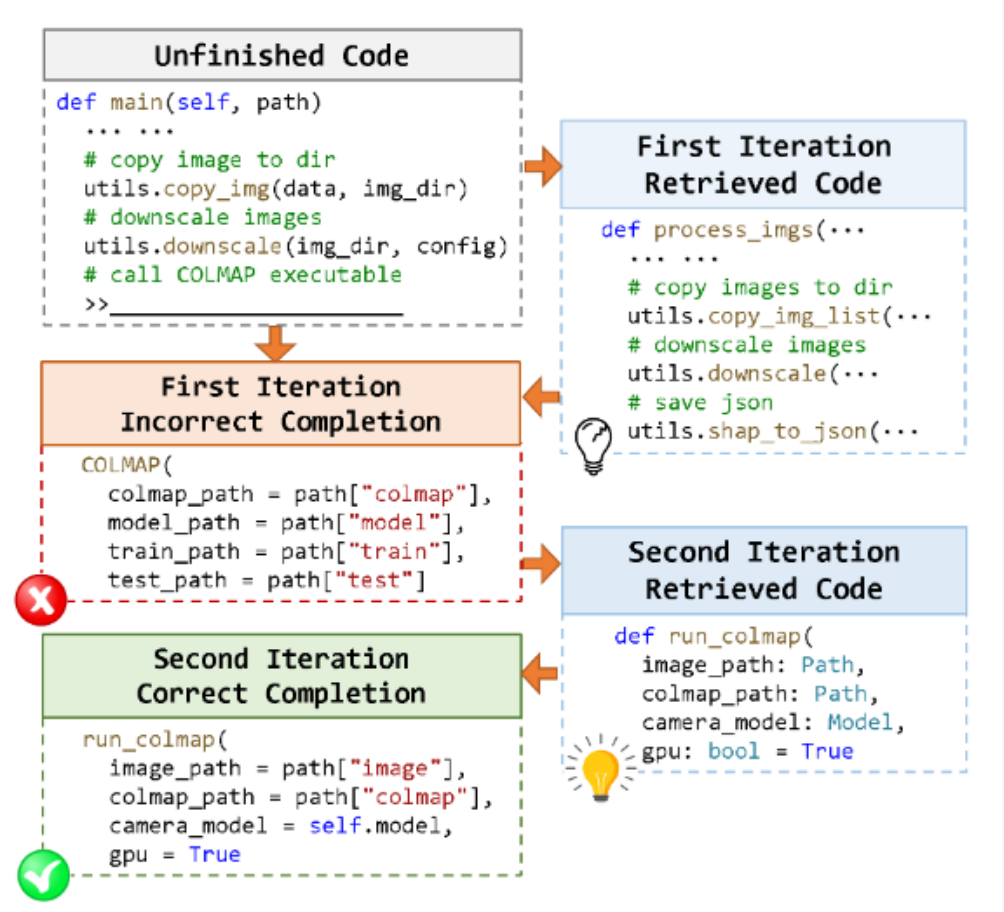


https://arxiv.org/pdf/2303.12570

RepoCoder: Iterative Retrieval and Generation

It streamlines the repository-level code completion process by incorporating a **similarity-based** retriever and a pre-trained code language model.

RAG is used!



https://arxiv.org/pdf/2303.12570

Hallucination

LLM-generated code may use the packages or functions that do not exist.

Hallucination

It is prevalent.

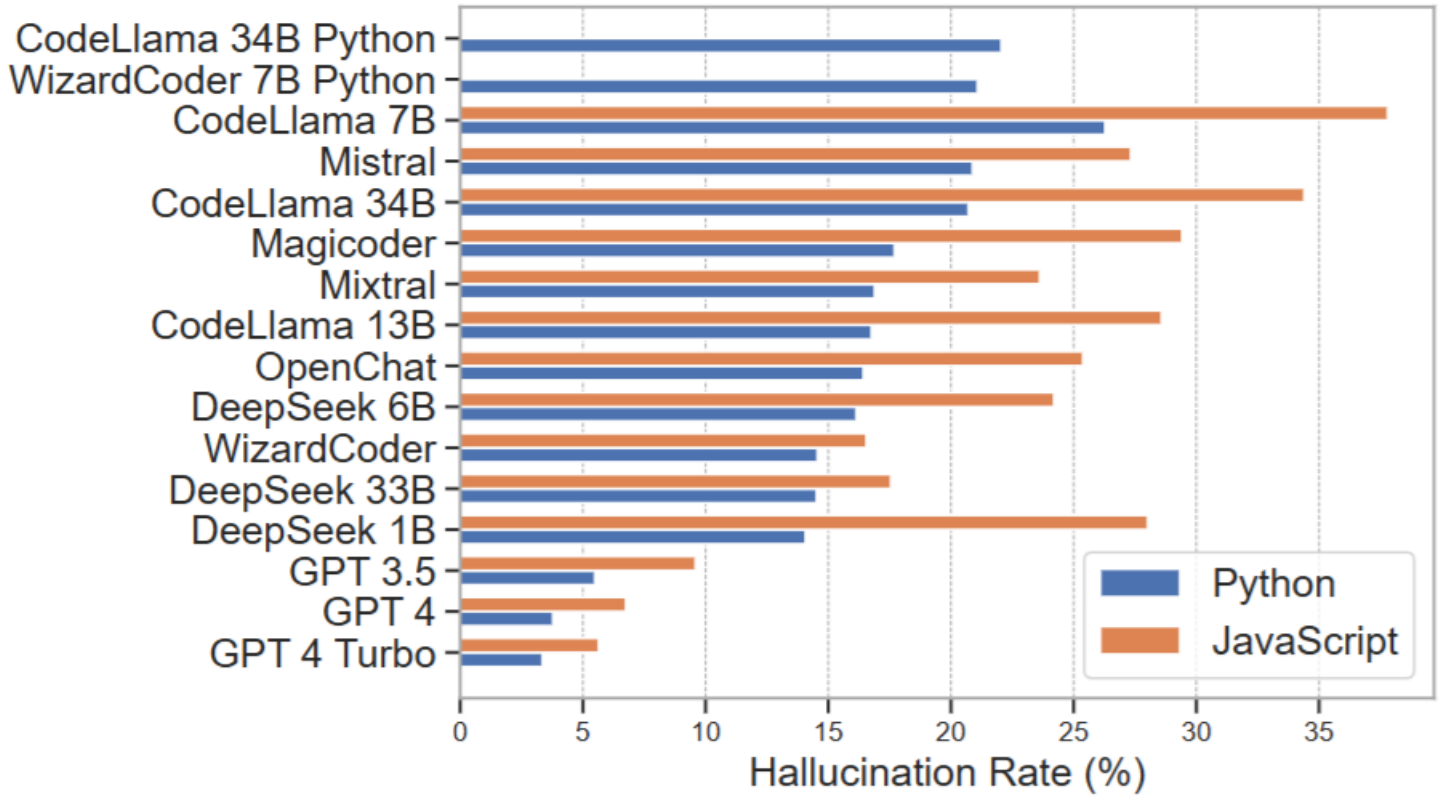
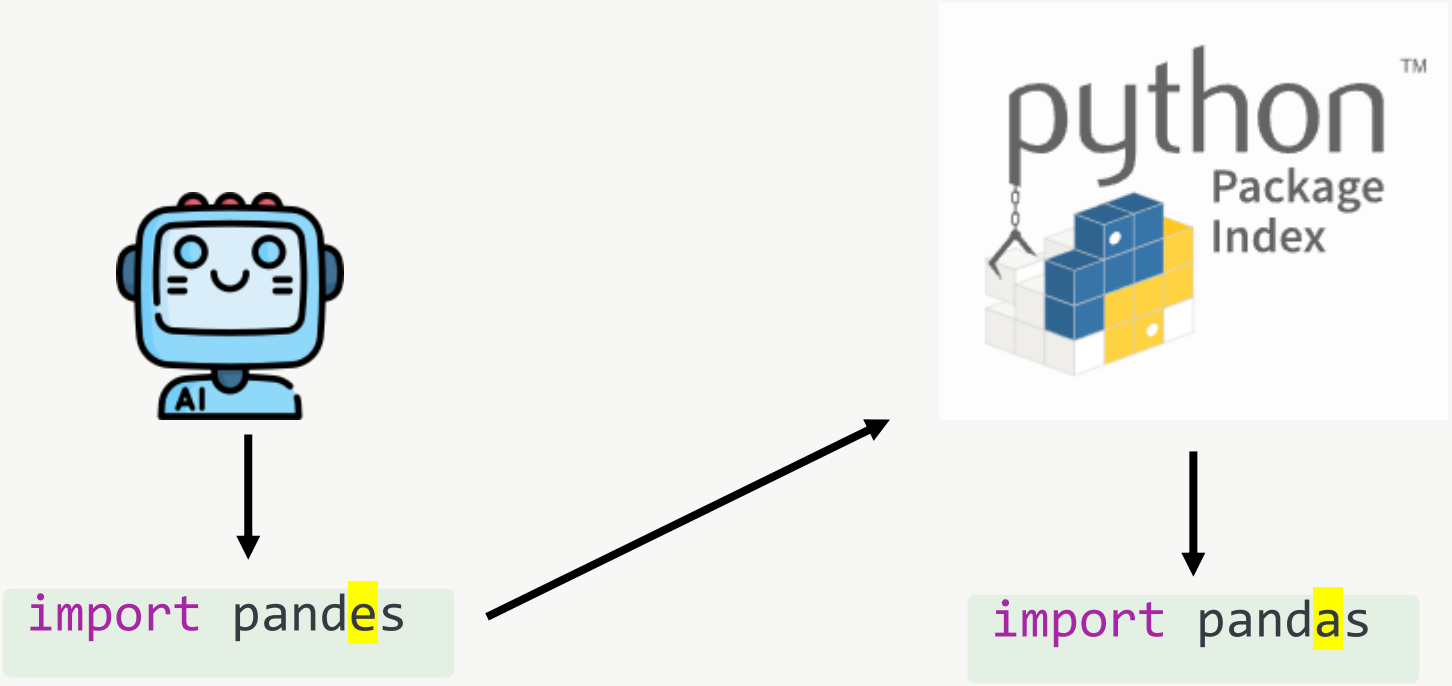


Figure 2: Observed hallucination rates of the tested models.

https://www.usenix.org/system/files/usenixsecurity25-spracklen.pdf

Fixing

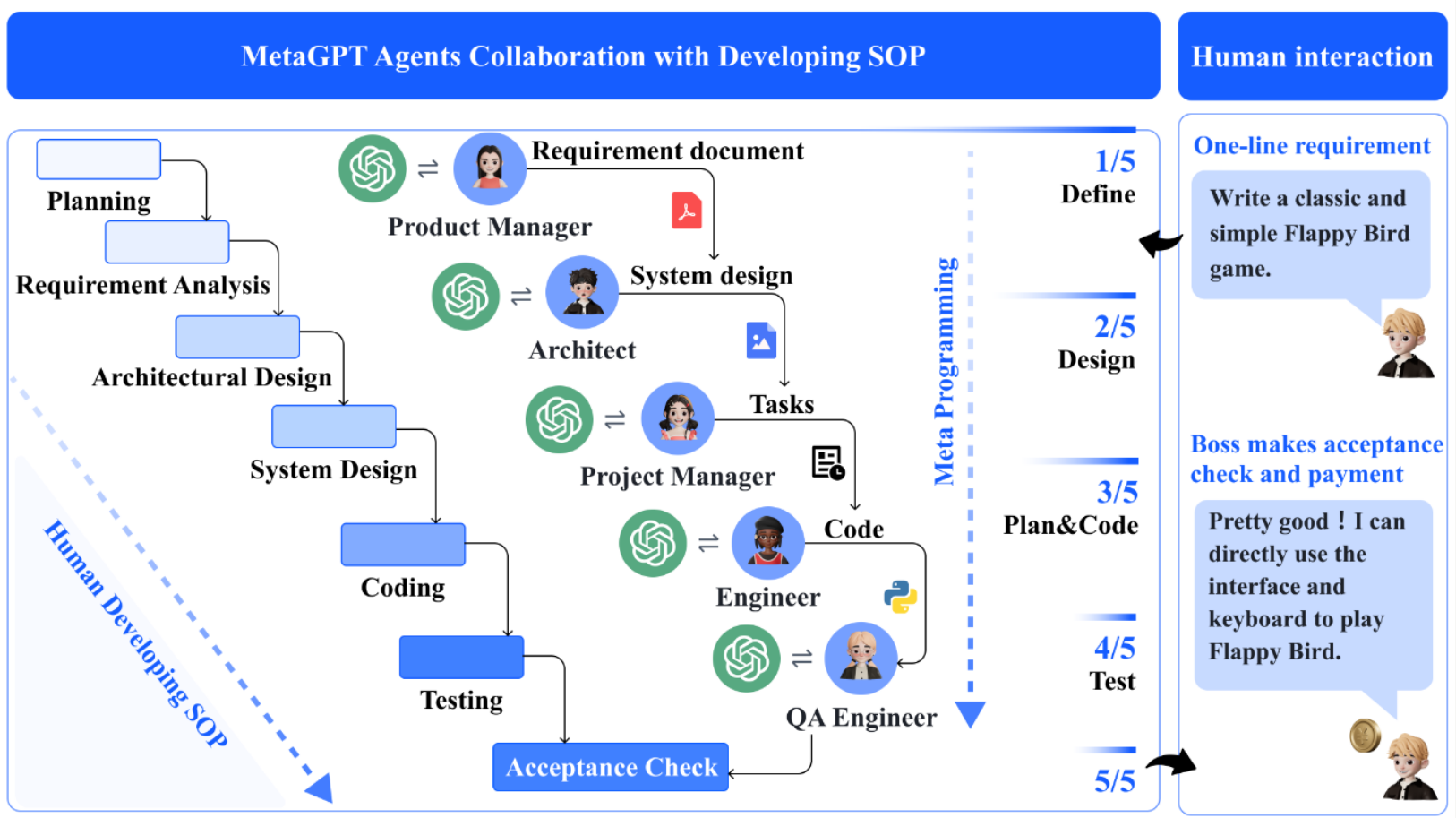


Autonomous Agents

- It dominates these techniques

MetaGPT: Multi-Agent Collaboration

Multi-agents for software engineering development cycle

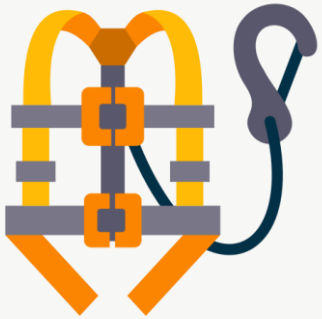


https://arxiv.org/pdf/2308.00352

Three Pillars



Model

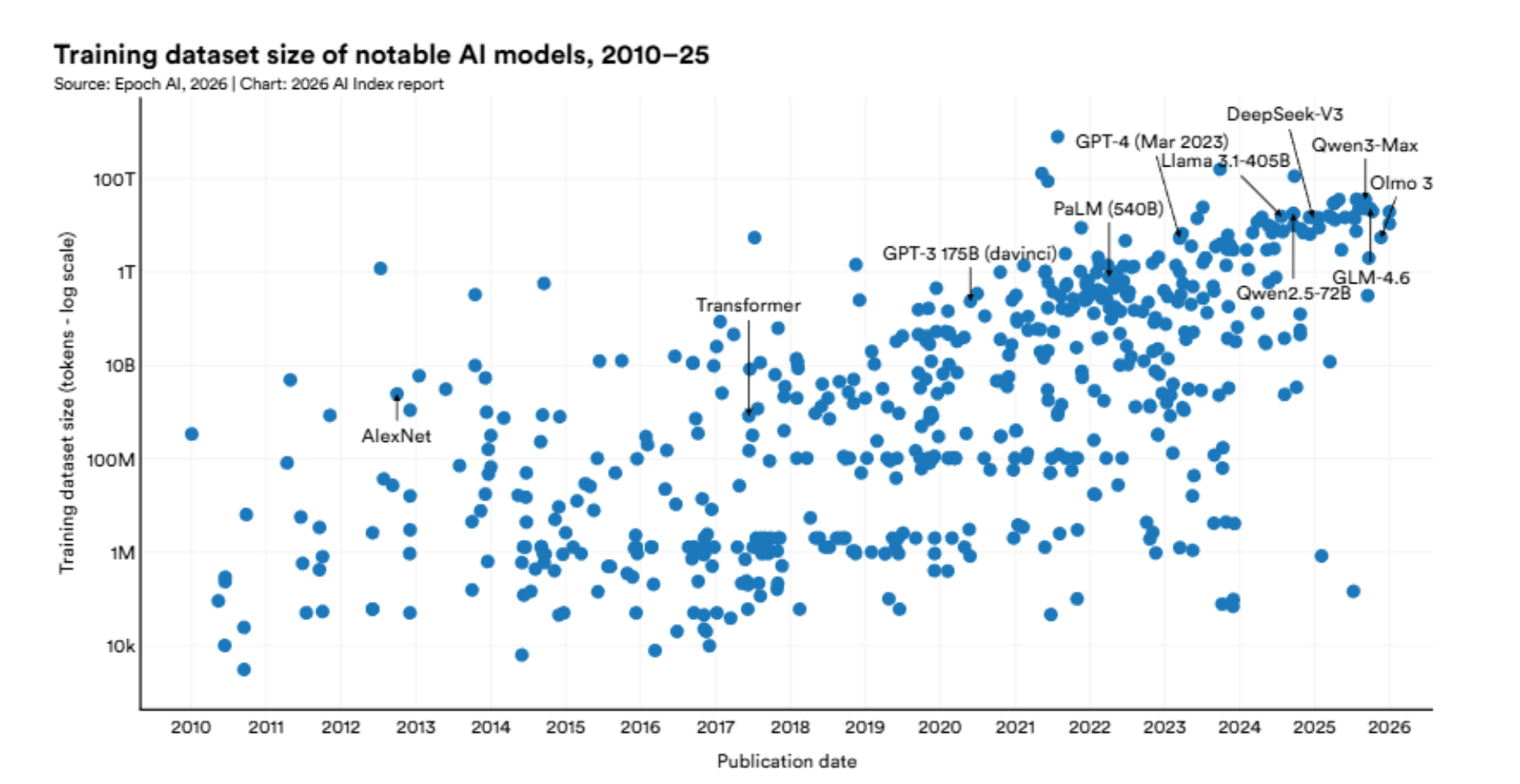


Harness



Benchmarks

We Require High-Quality Data



https://hai.stanford.edu/assets/files/ai_index_report_2026.pdf

How to Obtain High-quality Data?

- High quality
- Labor-intensive
- Privacy concern

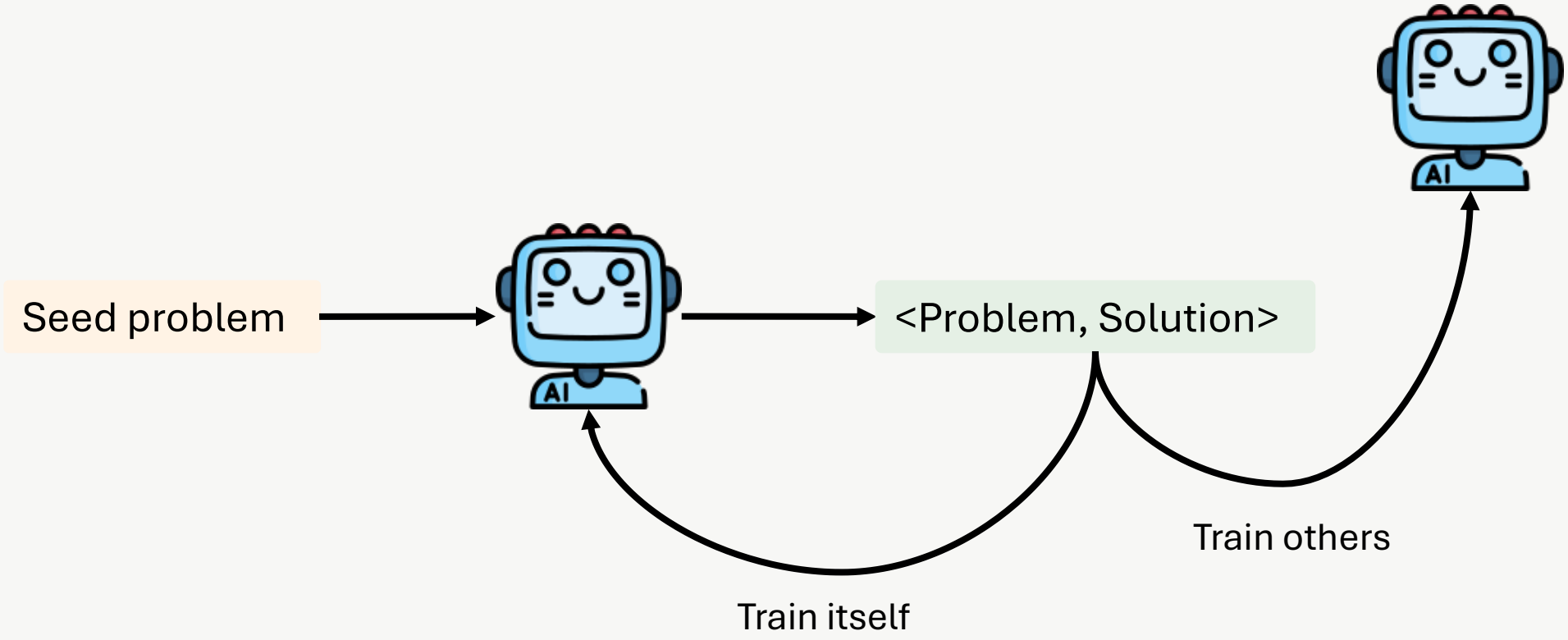


Benchmarks

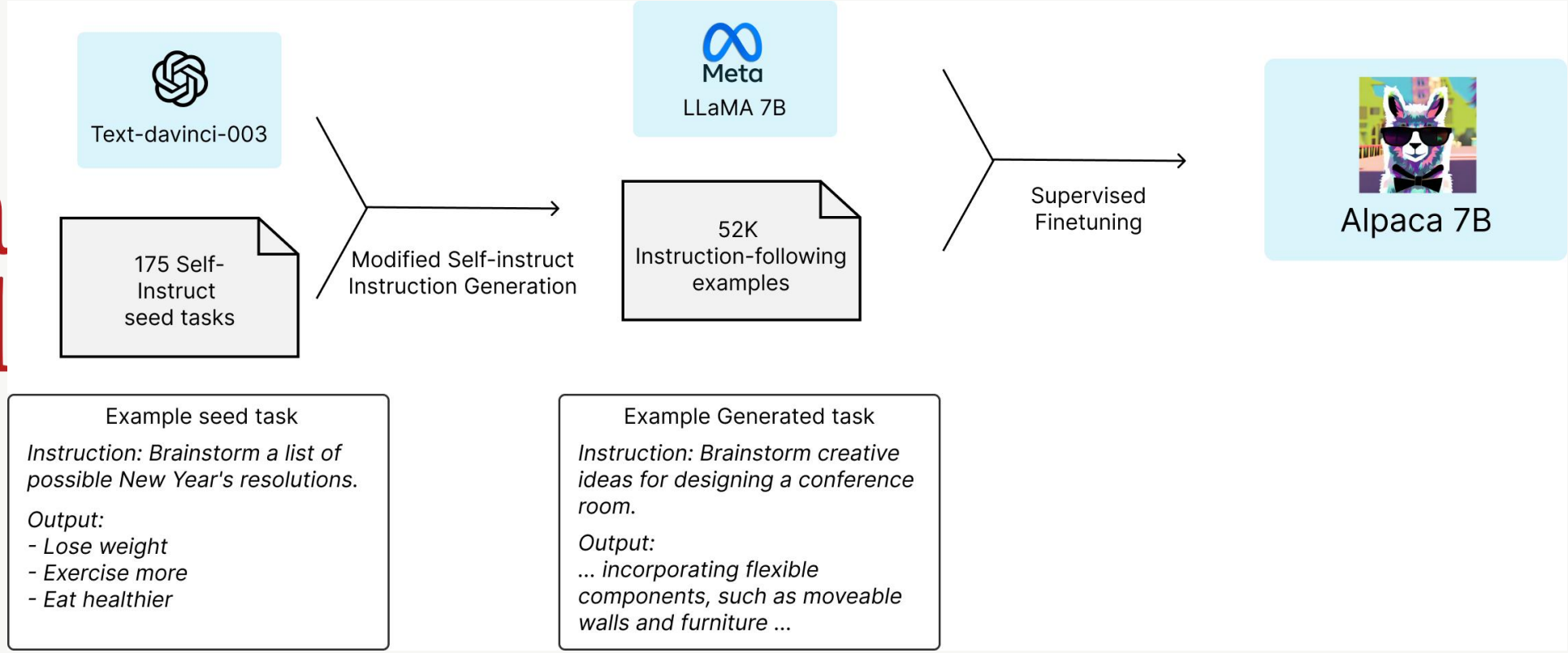
How to Obtain High-quality Data?

Synthetic data

Self-Instruct

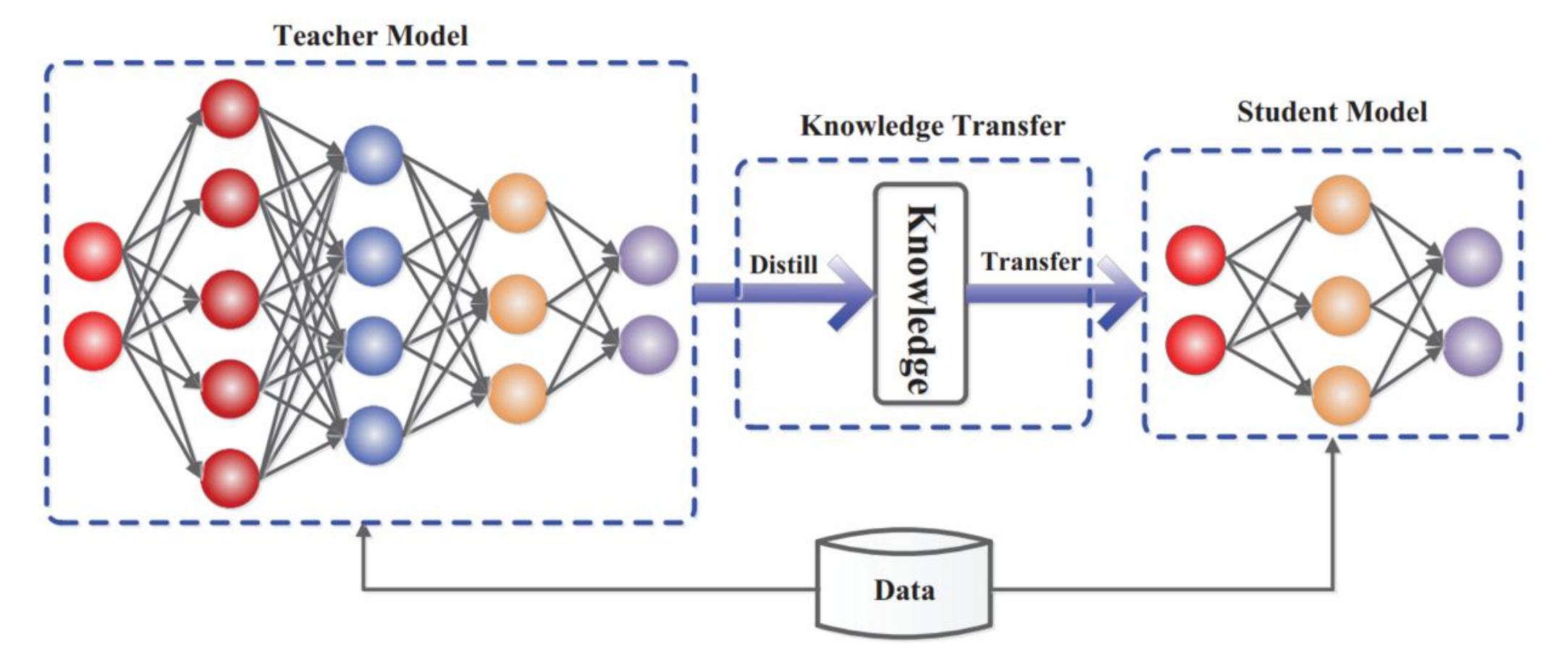


Alpcaca: a model fine-tuned from LLaMA 7B



https://crfm.stanford.edu/2023/03/13/alpaca.html

Knowledge Distillation



https://huggingface.co/blog/Kseniase/kd

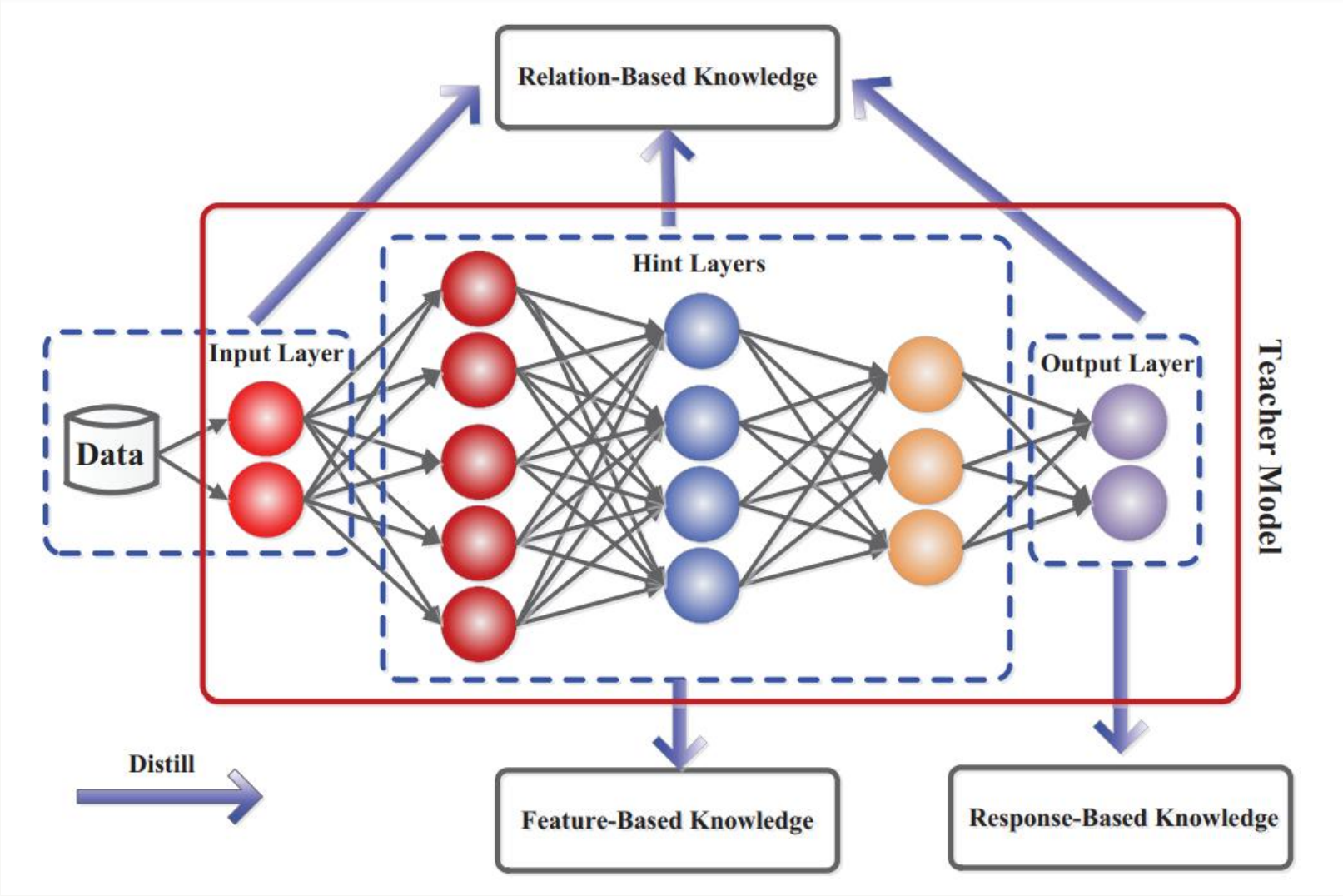
Example: Is This Email Spam?

- Input: “You won a free phone — claim it now!”
- Teacher (large model): Spam 90% | Promotion 8% | Normal 2%
- Student (small model): learns the teacher’s full pattern of guesses.
  - “Promotion” is a closer alternative than “Normal.”
- A hard label would only say: Spam.
- The teacher’s softer scores explain how it reached that answer.

How the Student Learns

- 1. Train the teacher on examples with correct labels.
- 2. Ask the teacher for scores on many inputs.
- 3. Soften the scores with a higher temperature (T).
- 4. Train the student to match those scores and the true labels.
  - Loss = teacher-matching loss + true-label loss
- Result: one compact model that imitates useful teacher behavior.

What to Learn



Gou, Jianping, et al. "Knowledge distillation: A survey." *International journal of computer vision* 129.6 (2021): 1789-1819.

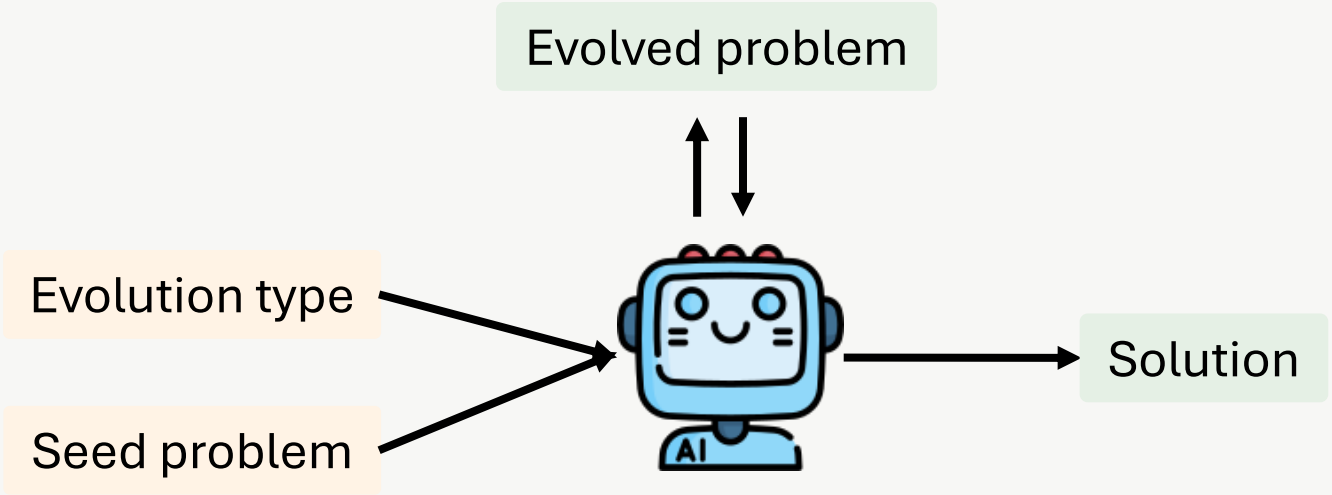
What to Learn

- Response-based distillation (outputs as knowledge)
- Feature-based distillation (intermediate layers as knowledge)
- Relation-based distillation (relationships as knowledge): The student model learns to mimic relationships between different parts of the teacher model – either between layers or between different data samples.

Why Distill? What to Watch For

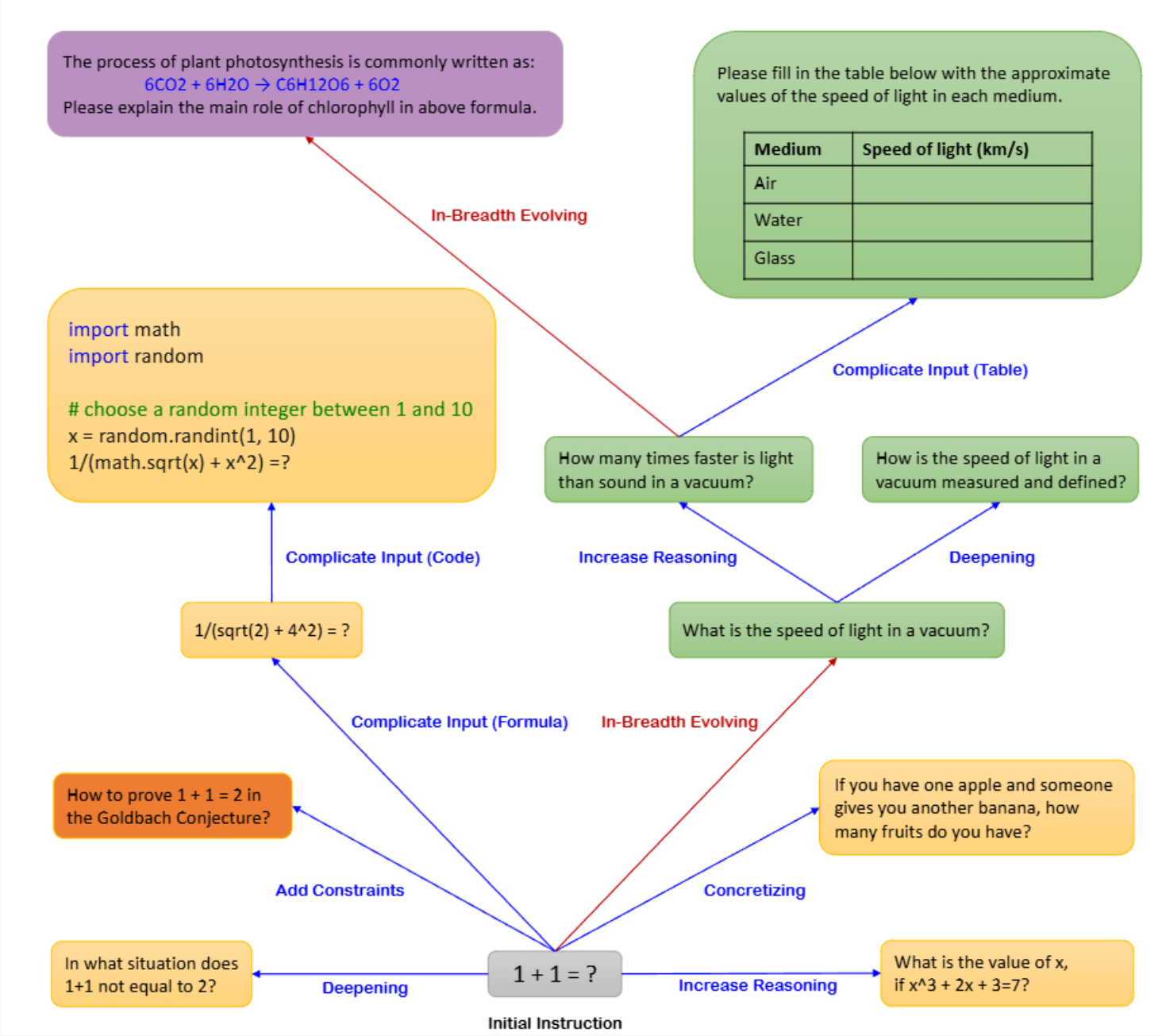
- Useful when serving a large model is too slow or expensive.
- The student can be smaller, faster, and easier to deploy.
- It may still lose some capability; distillation is not a guarantee.
- A student can also copy teacher mistakes or bias.
- For coding: teach a small code model from a stronger model's solutions and confidence.

Evol-Instruct



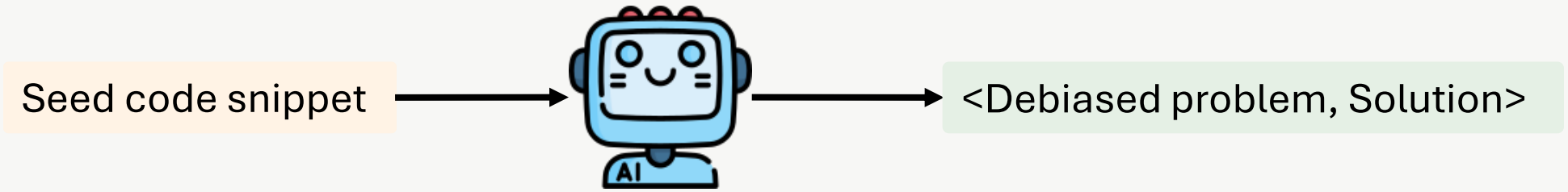
Incrementally transforms simple instructions into more complex variants.

Make the instructions more complicated

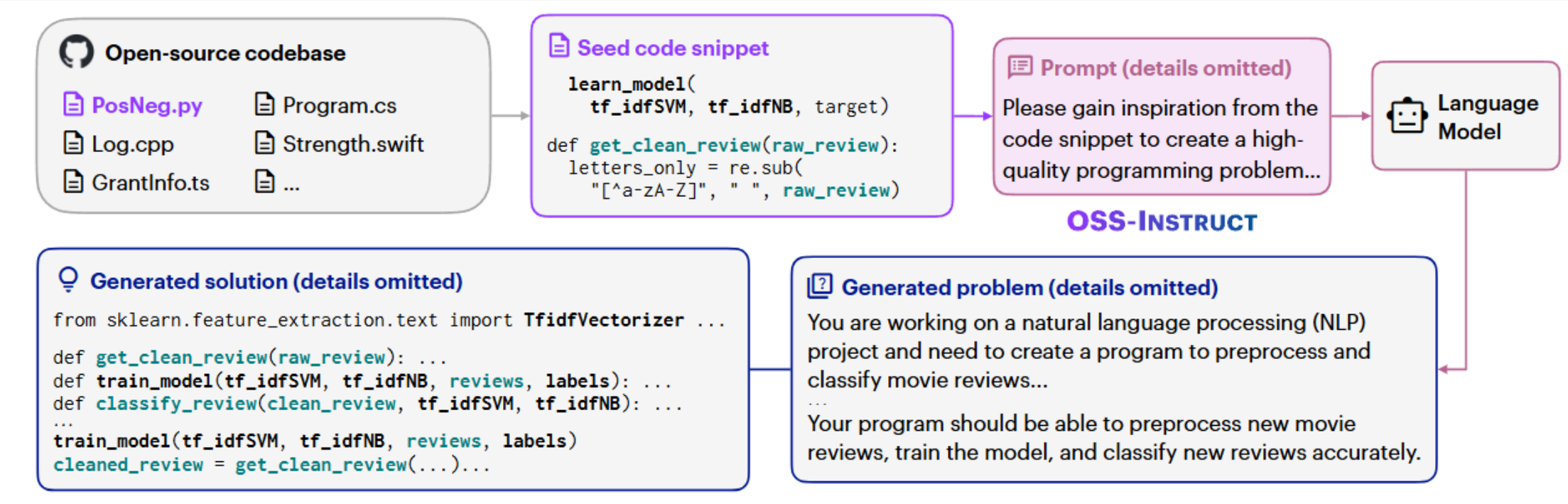


https://arxiv.org/pdf/2304.12244

OSS-Instruct

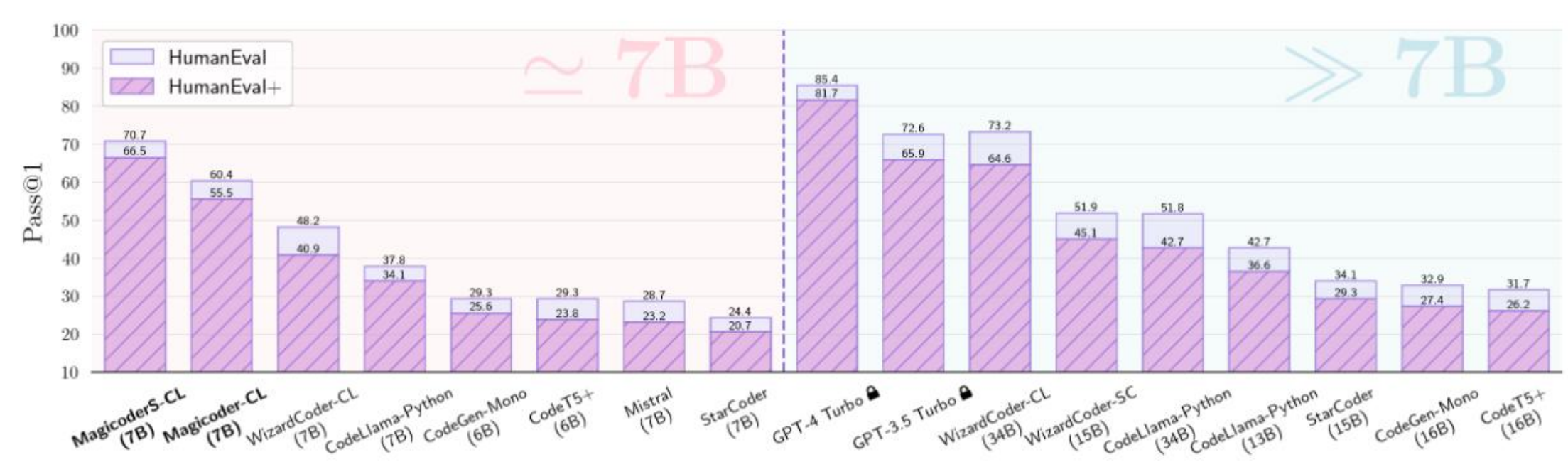


Using Open-source code data



https://arxiv.org/pdf/2312.02120

Using Open-source code data



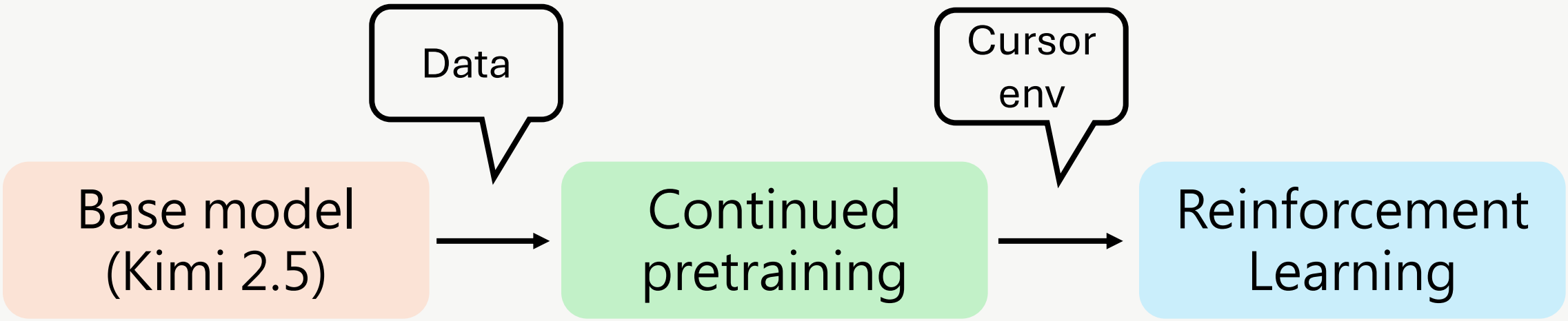
https://arxiv.org/pdf/2312.02120

Now, let’s come to the model part

Coding LLMs

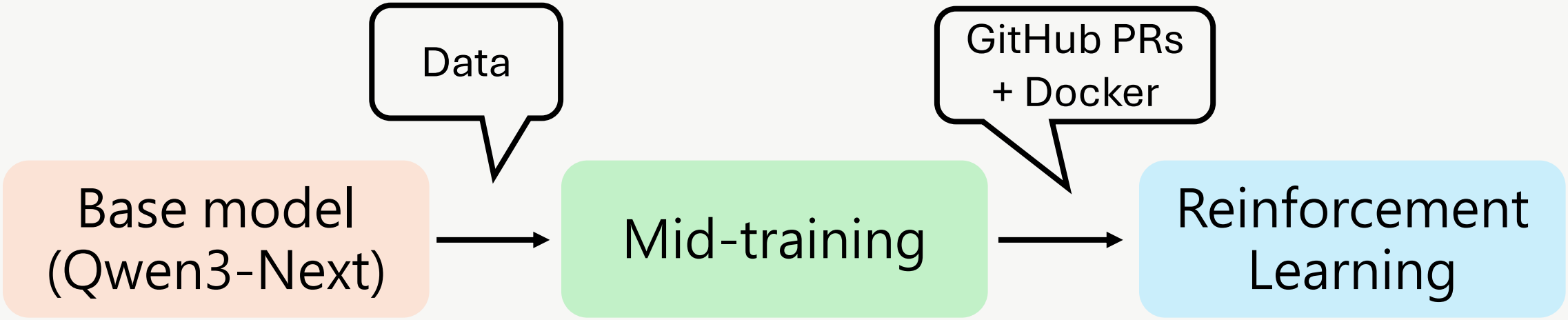
- Training a LLM from scratch is super expensive
- A practical way to fine-tune a base model

Composer 2 (The Model in Cursor)



Composer 2 Technical Report: https://arxiv.org/pdf/2603.24477

Qwen3-Coder-Next



https://arxiv.org/pdf/2603.00729

Qwen3-Coder-Next

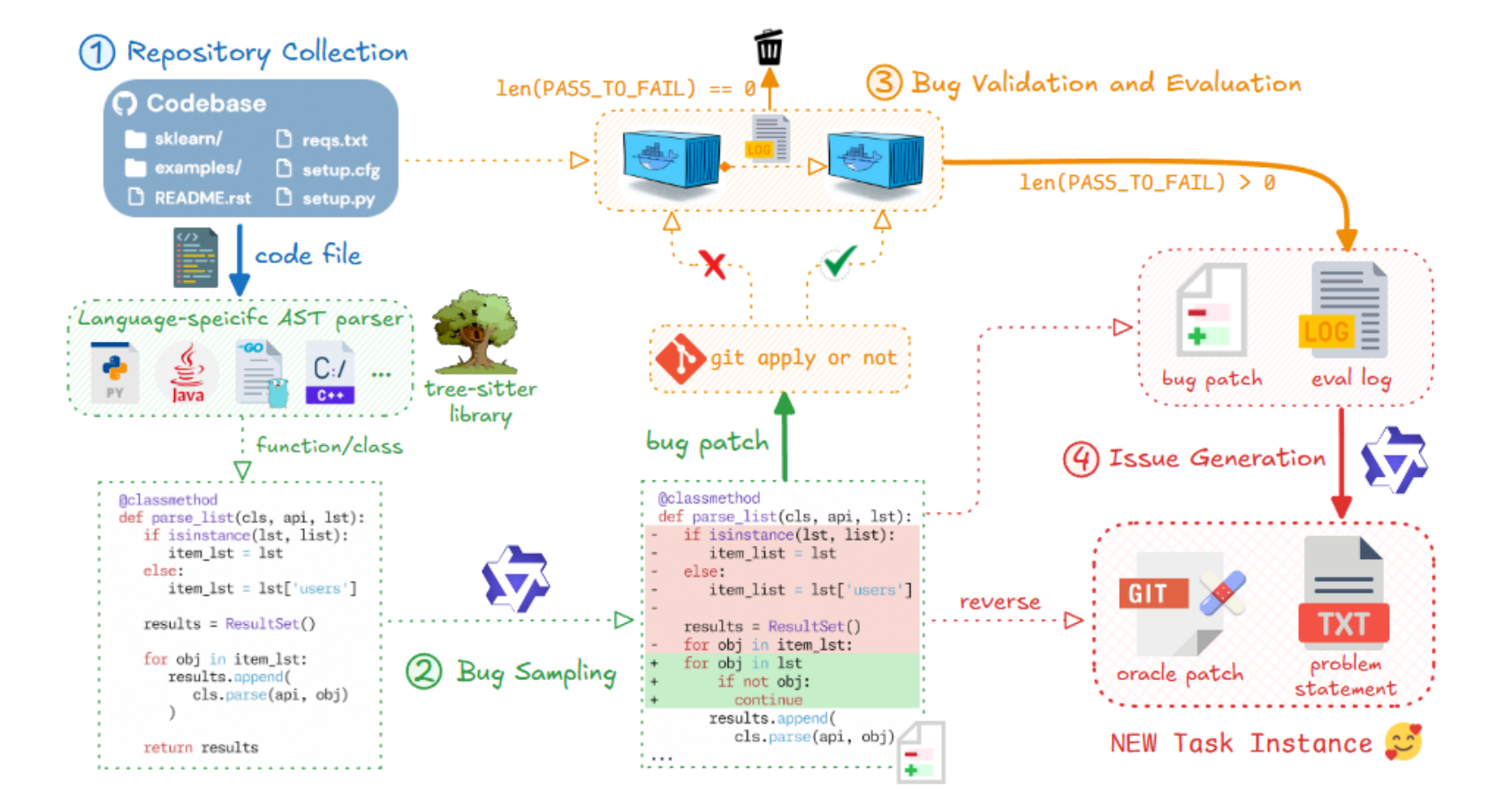


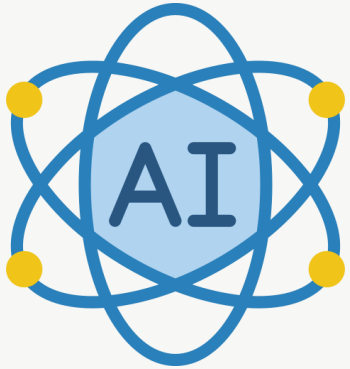
Figure 2: Our pipeline for synthesizing bugs to scale up the number of software engineering tasks.

https://arxiv.org/pdf/2603.00729

Reinforcement Learning

- We have a lot of RL algorithms for LLMs
  - PPO, DPO, GRPO, GSPO
- Mature RL training frameworks
  - https://github.com/verl-project/verl
  - https://github.com/OpenRLHF/OpenRLHF
  - https://github.com/huggingface/trl
- For SE tasks, rewards are critical

Rewards



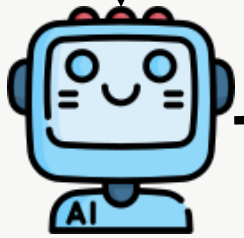
Implicit rewards
(Use another LLM to judge)



Verifiable rewards
(Execution result feedback)

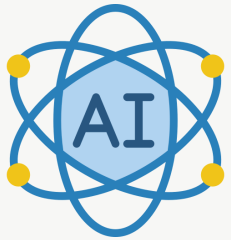
Reward Model

Implement a discount calculation function




```python
def discount_price(price):
    return price * 0.8
```

```python
def discount_price(price):
    if (price < 0):
        raise ValueError("Negative")
    return price * 0.8
```



0.2

0.8

Reward model

Verifiable rewards

Execution Outcome	Reward Score	Interpretation
All Tests Pass	+1.0	Perfect functional correctness
Some / All Tests Fail	-1.0	Logic error or edge case failure
Compilation Error	-1.0	Invalid syntax
Timeout / Memory Exceeded	-1.0	Inefficient algorithmic complexity

Sparse Reward

The Problem: This is known as a "sparse reward." If the LLM generates a 500-line program and makes a single off-by-one error on line 499, it receives a -1.0. It gets no partial credit and no information about *where* it failed, making exploration incredibly difficult.

Multi-Granularity and Fine-Grained Feedback

Instead of a reward on final result, provide step-by-step rewards.

If an error occurs on line 6, the model is penalized specifically for the tokens that led to that error, while unexecuted branches are "masked" so perfectly valid syntax is preserved.

```
1: def calculate_metrics(data):
2:   if not data:
3:     return 0                  <--- [ Valid path: Do not penalize ]
4:
5:   total = sum(data)
6:   average = total / len(dta) <--- [ Typo: NameError thrown here! ]
7:   return average              <--- [ Unexecuted: Mask out from penalty ]
```

Static Analysis as Reward

Linter Finding (e.g., Ruff / Cppcheck)	Severity Mapping	Reward Penalty
SyntaxError (Code won't run)	error	-10.0
SQL Injection Vulnerability	high	-5.0
Variable declared but unused	normal	-1.0
Missing docstring / bad spacing	low	-0.2

https://openreview.net/pdf?id=ZNDLv4qwqA

Reasoning Process Reward (CoT)

```
[ Step 1: Evaluate Reasoning (PRM) ]
                    ---> Score: +0.8 (Highly logical)
```

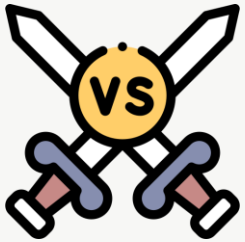


```
[ Step 2: Evaluate Final Code ]
                ---> Outcome: Fails Compilation (0)
```



```
[ Final Reward Calculation ]
              ---> ( +0.8 ) * ( 0 ) = 0.0 Total Reward
```

https://arxiv.org/pdf/2508.05170

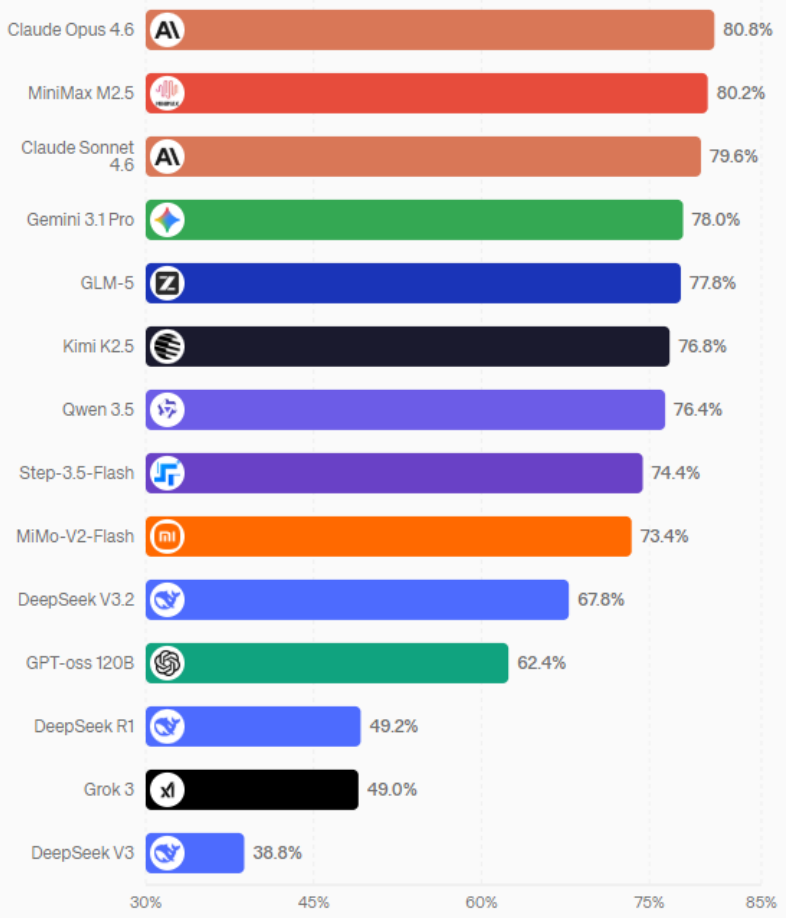
General models  Coding models

Best Coding LLMs by Benchmark

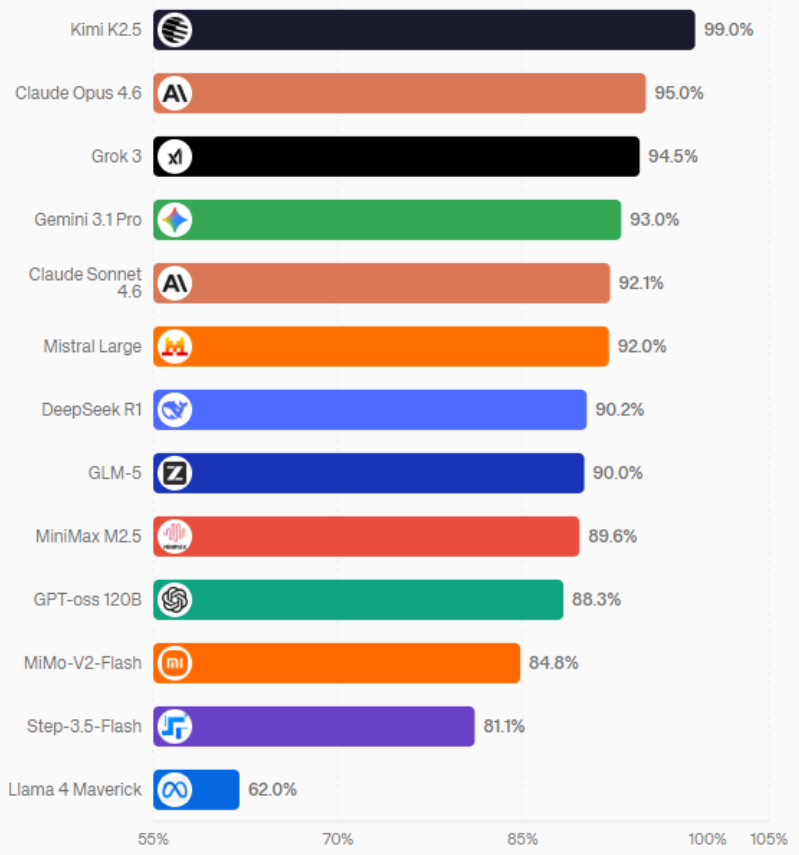
How does each model perform on real-world software engineering, code generation, competitive programming, and terminal-based coding tasks? Hover any bar for details.

Coding | Reasoning

Best at Software Engineering

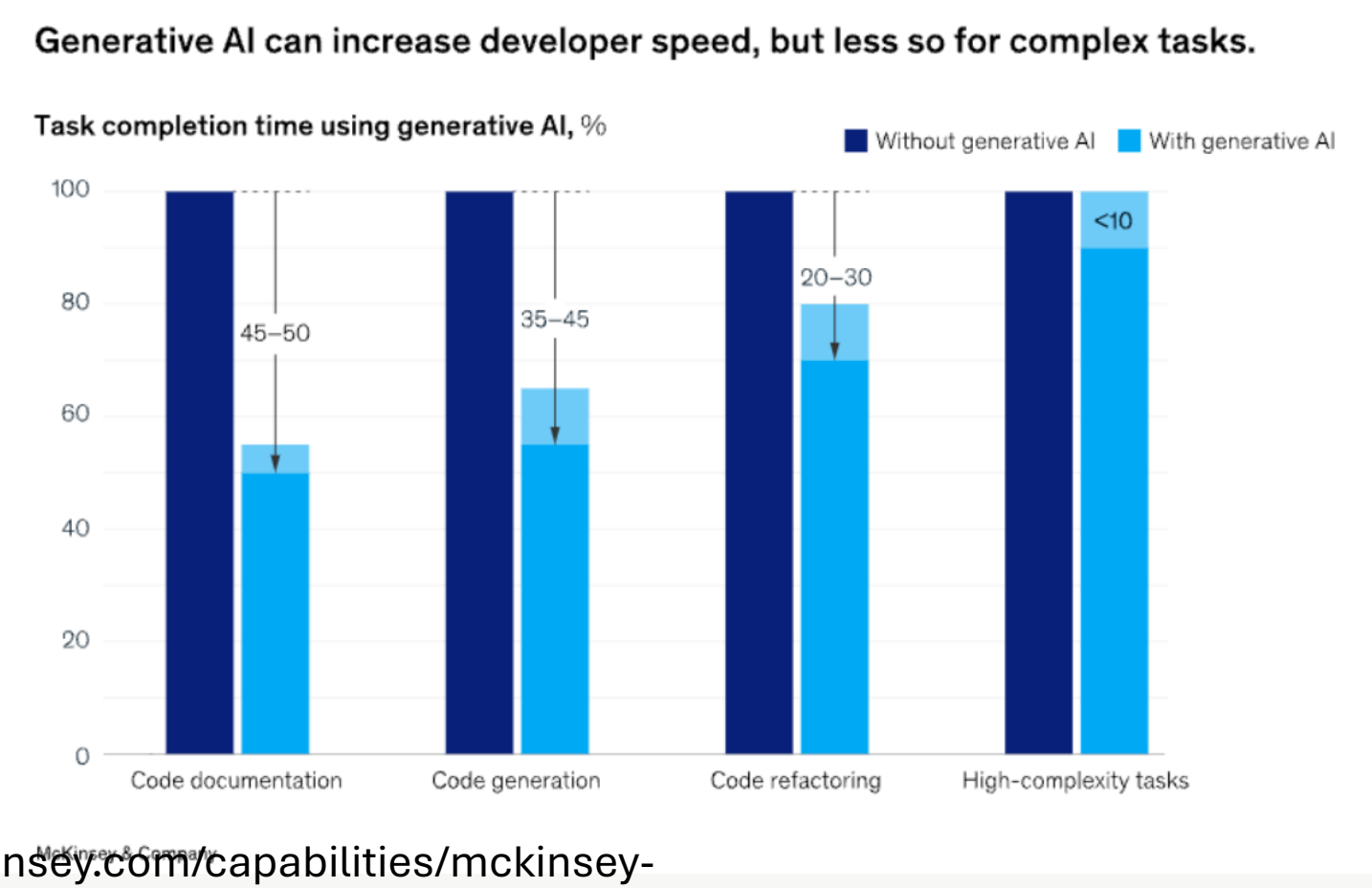


Best for Code Generation

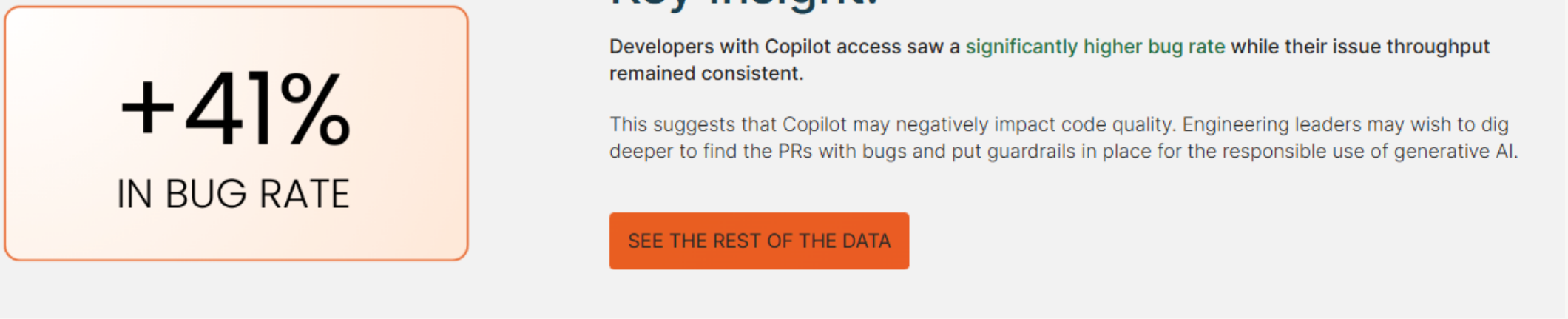


https://onyx.app/best-llm-for-coding

Does LLM really improve the efficiency?



https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai#/



https://resources.uplevelteam.com/gen-ai-for-coding



香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Questions

Next class: Team Collaboration