

Coding Agents

CS6960 MultiModal LLM Agents

Syeda Mishra Saiara

Announcements

- HW2 due in 2 weeks
 - Reach out to us if you have any questions
- Projects
 - Deadline for group formation - Feb 10 (Today), fill out your group information if you haven't done so already
 - Rubric posted for proposal (see announcement)
 - Reach out to us if you have any questions/concerns
- Participation Quiz will be opened before the student presentation

Any Questions

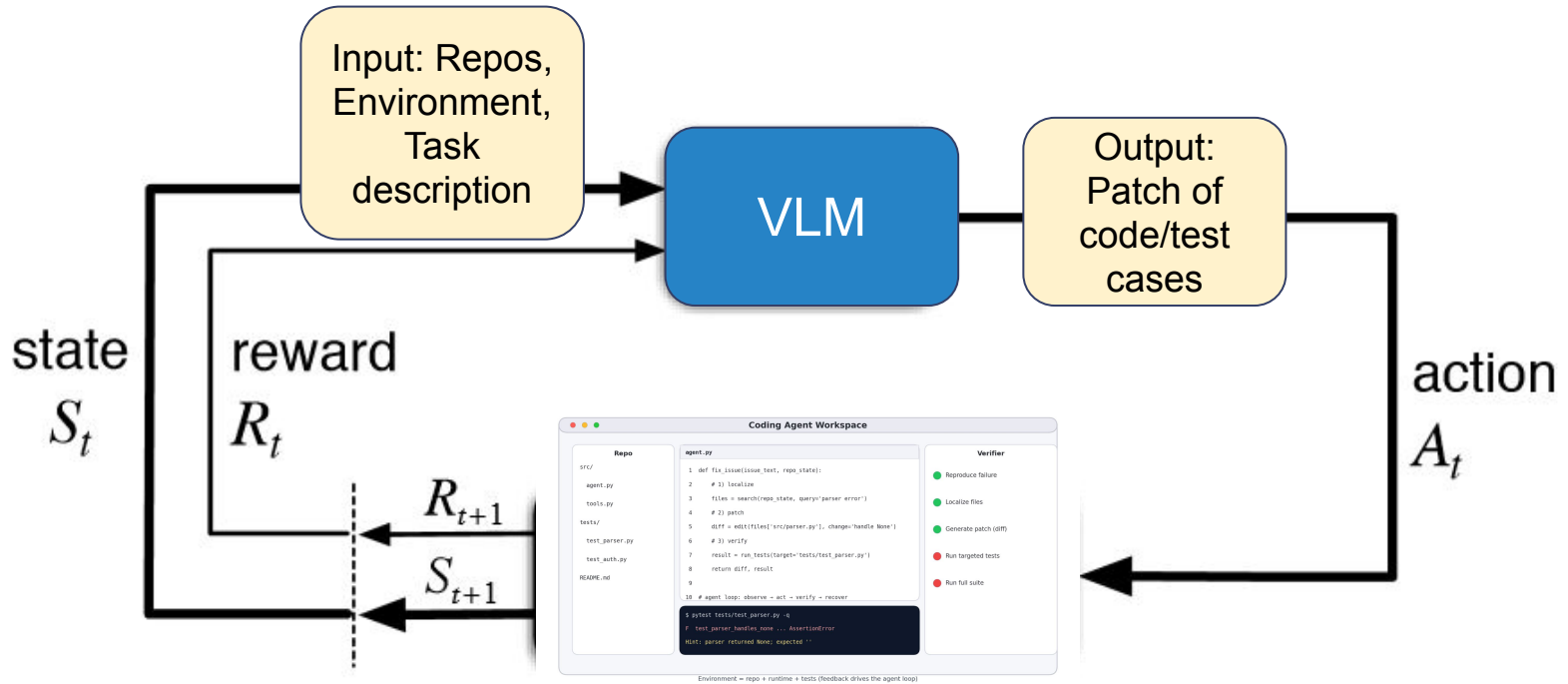
From code generation to coding agents

- A “Coding Agent” isn’t just **code generation**
- Closed-loop controller:
 - Reads context, takes action, validates, recovers
 - More like docker container
- Repo - level difficulty:
 - Long context, multi-file edits, dependencies, environment mismatch

What is a Coding Agent

- Input :
 - Goal: Issue/task
 - Repositories
 - Environment
- Output :
 - Patch that meets a verifier (tests/builds/checks)
- Loop: Observe > Propose > Act > Observe (until done)

What is a Coding Agent?



Challenges in Development Agents

- Defining Environment
- Designing Observations/ Actions
- Code generations
- File localization
- Planning and error recovery
- Safety

Code environments

- Code repositories (GitHub, Gitlab, Bitbucket)
- Task management SW (Jira)
- Office & Communication
- **Testing environments:**
 - Focused mostly on coding environments

Code environments (SWEBench)

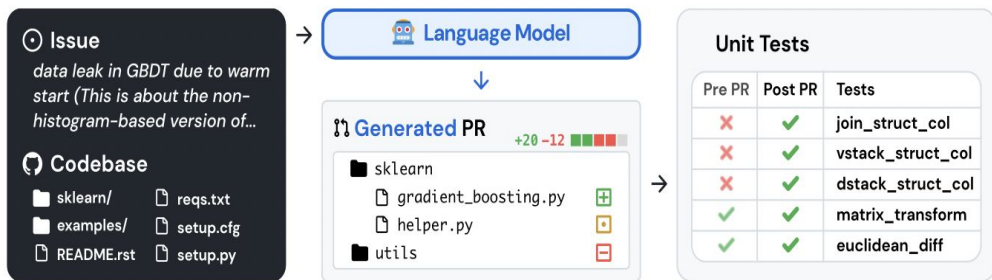


Figure 1: SWE-bench sources task instances from real-world Python repositories by connecting GitHub issues to merged pull request solutions that resolve related tests. Provided with the issue text and a codebase snapshot, models generate a patch that is evaluated against real tests.

- Codebase with description of an issue
- LLM tasked to edit codebase to address the issue
- Requires understanding and collaborating changes in -
 - Functions, classes, files simultaneously
- Process extremely long contexts and perform complex reasoning that goes far beyond traditional code generation tasks.

Action Spaces

- Understand repository structure
- Read in existing code
- Modify or produce code
- Running codes
- Debugging

Action Spaces (CodeACT)

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 984.00 in USD.

Success Rate (%)

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

Environment 1.1, 0.19

Response The most cost-effective country to purchase the smartphone model is Japan with price 984.00 in USD.

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

Average Number of Interaction Turns

- Provided with all APIs of the tools it wants to work with

Action Spaces (CodeACT)

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

Fewer Actions Required!

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.

Success Rate (%)

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 = {}

Control & Data Flow of Code Simplifies Complex Operations

```
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)
```

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.

Average Number of Interaction Turns

- Python interpreter

Action Spaces (CodeACT)

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.

Success Rate (%)

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)
```

Environment 1.1, 0.19

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

Average Number of Interaction Turns

- Execute code actions

Action Spaces (CodeACT)

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.

Success Rate (%)

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)
```

Environment 1.1, 0.19

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



Fewer Actions Required!

Control & Data Flow of Code Simplifies Complex Operations

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

Average Number of Interaction Turns

- Dynamically reverse prior actions

Action Spaces (CodeACT)

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.

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



Fewer Actions Required!

Environment 1.1, 0.19

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

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

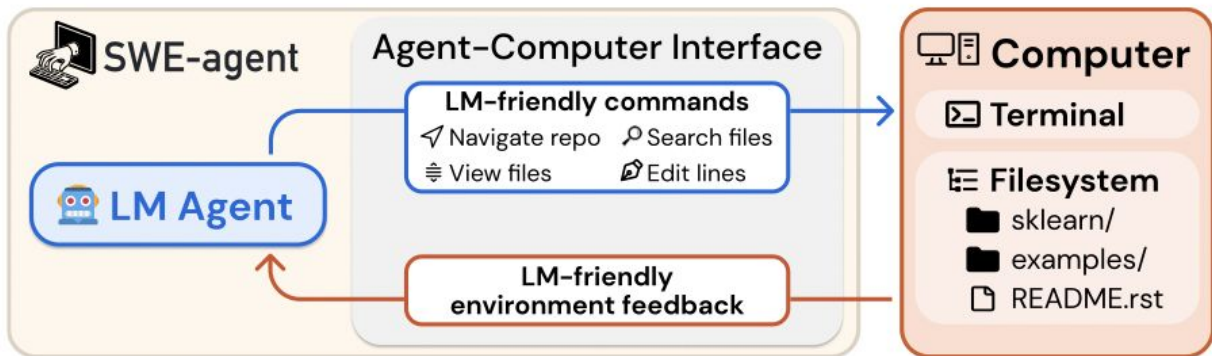
Success Rate (%)

Average Number of Interaction Turns

- Emit new actions upon new observations through multi-turn interactions

Observations

- Loop: Observe > Localise > Patch (action) > Verify > Recover



Observations

- Loop: Observe > Localise > Patch (action) > Verify > Recover

Observation (File Viewer)

```
[File: solvers/diophantine.py (3093 lines total)]  
(400 lines above)  
401:     elif total_degree == 3:  
402:         diop_type = "cubic_thue"  
403:  
404:     elif (total_degree > 3):  
405:         if all(coeff[k] == 1):  
406:             diop_type = 'general_sum_even'  
407:  
408:     if diop_type is not None:  
409:         return var, coeff, diop_type  
(2684 lines below)
```

(a) Observation from the file viewer.

Response (Thought + Edit Action)

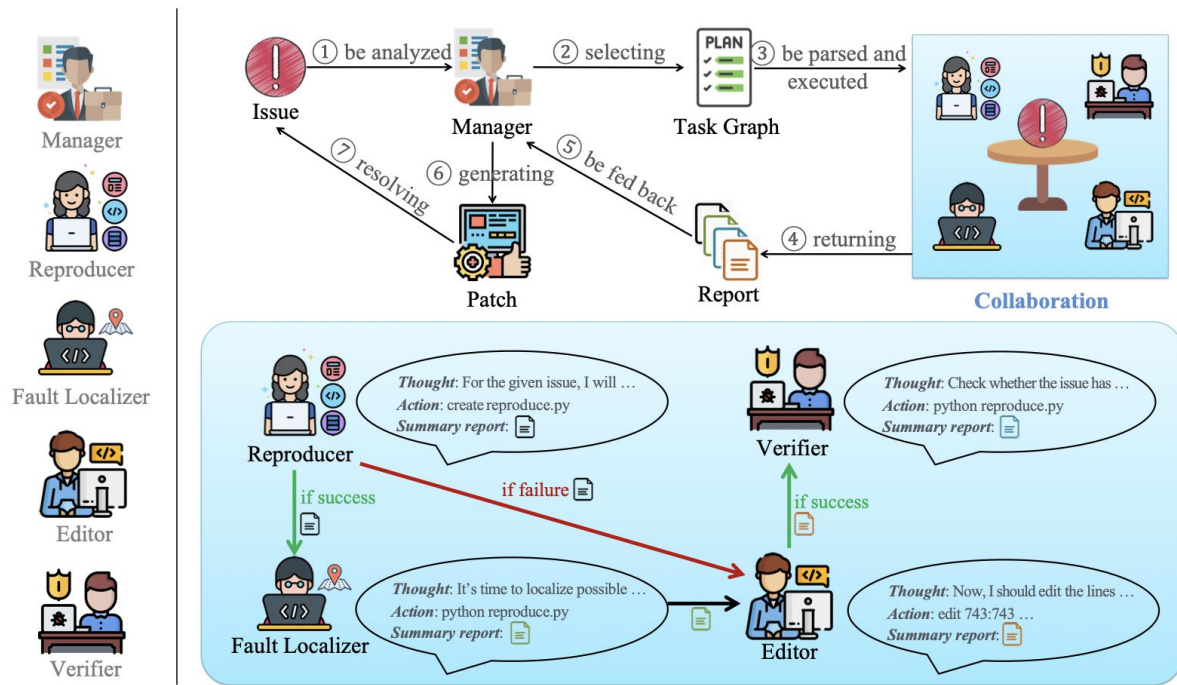
We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

```
edit 404:407  
    elif (total_degree > 3 and  
          len(set(k.exp[k.is_Pow])) == 1):  
        if all(coeff[k] == 1):  
            diop_type = 'general_sum_even'  
end_of_edit
```

(b) Action using the edit interface.

- Uses ReAct here

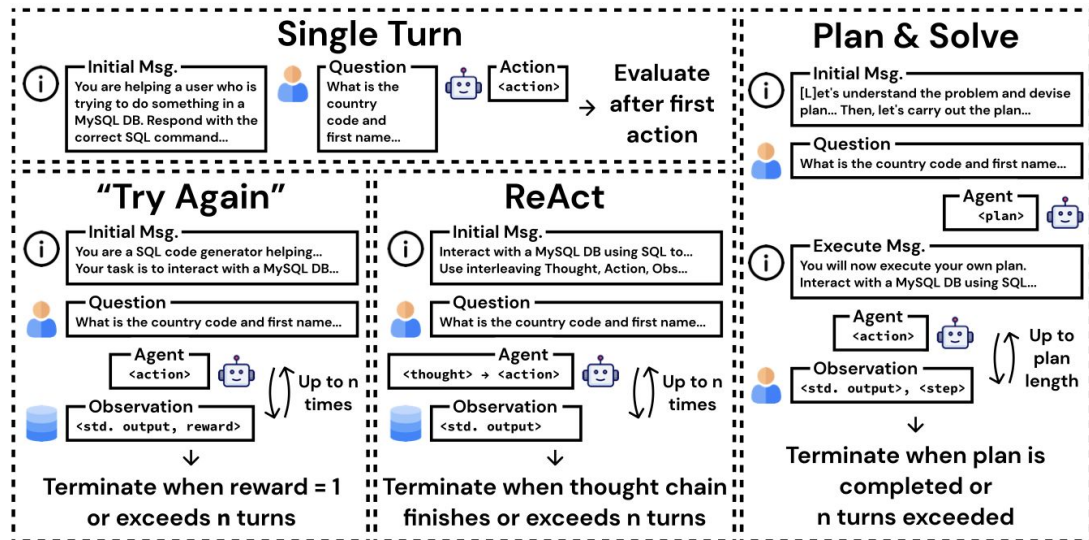
Planning & Error Recovery (CodeR)



- CodeR adopts multi-agent framework
- Takes pre-defined task graphs to Repair & Resolve reported bugs
- Adds new features within code repository

Figure 1: Multi-Agent framework of CODER with task graphs.

Planning & Error Recovery (InterCode)



- Disconnection between generated code and final execution
- InterCode uses RL environment with *code as actions*, and *execution feedbacks as observations*

Figure 2: Overview of Prompting Strategies adjusted for evaluation on InterCode. The "Try Again" termination constraint is conditioned on reward = 1, while ReAct [51] and Plan & Solve [43] are determined by the agent itself. This is because the purpose of the "Try Again" method is to explore how capable agents are at error correction from feedback, while the other two are more concerned with the overall success of general problem-solving strategies.

Planning & Error Recovery (SelfDebugging)

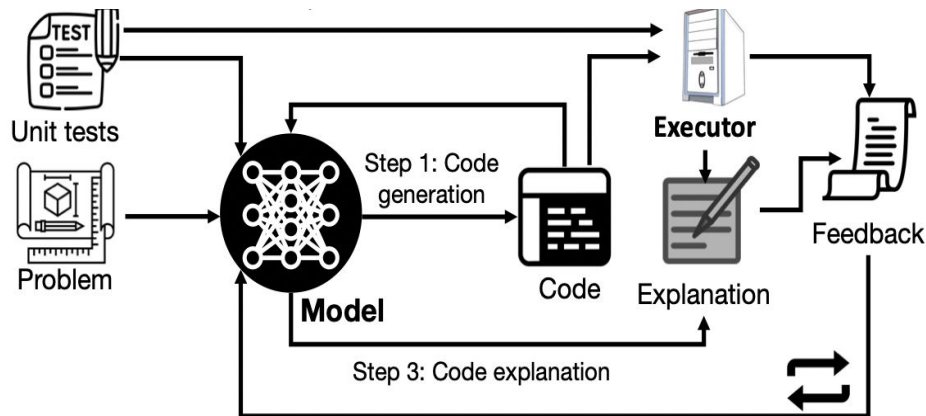


Figure 1: SELF-DEBUGGING for iterative debugging using a large language model. At each debugging step, the model first generates new code, then the code is executed and the model explains the code. The code explanation along with the execution results constitute the feedback message, based on which the model infers the code correctness and then adds this message to the feedback. The feedback message is then sent back to the model to perform more debugging steps. When unit tests are not available, the feedback can be purely based on code explanation.

- Interesting paper to give a read
- It teaches LLMs to perform without human feedback on code correctness/ error messages
- Model is able to identify mistakes by investigating the execution results
- Explains the generated code in natural language

Step 1: Question explanation

Infer the return type of the question.

[Question]

Which customers have both "On Road" and "Shipped" as order status? List the customer names.

[Question Explanation]

"List the customer names" returns 1 column. The question returns the customer names who have both "On Road" and "Shipped" as order status. So the question returns 1 column.

Step 2: Code explanation

Summarize the return type of the execution.

[SQL]

```
SELECT customers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id
WHERE orders.order_status = "On Road" OR orders.order_status = "Shipped"
```

Execution:

```
| George |
```

[SQL Explanation]

The execution of the SQL query above would return a table with 1 column.

The first column, "customers.customer_name" would contain the customer names.

With "customers JOIN orders", the table would contain the data about customers with orders.

With "WHERE orders.order_status = 'On Road' OR orders.order_status = 'Shipped'", the table filters the records to only include customers who have order status "On Road" or "Shipped".

So the SQL query returns a table with 1 column, the customer names who have the order status "On Road" or "Shipped".

Step 3: Self-Debugging with explanation

[Question]

[Explanation summary]

As in your explanation, the SQL query returns a table with 1 column, the customer names who have the order status "On Road" or "Shipped". The question returns 1 column, the customer names who have both "On Road" and "Shipped" as order status.

[Feedback]

So the SQL prediction above is wrong. Please fix the SQL.

[Revised SQL #1]

```
SELECT customers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id
WHERE orders.order_status = "On Road" AND orders.order_status = "Shipped"
```

[Explanation summary #1]

As in your explanation, the SQL query returns an empty table, because the "WHERE" clause contains mutually exclusive conditions.

The question returns 1 column, the customer names who have order status of both "On Road" and "Shipped".

[Feedback #1]

So the SQL prediction above is wrong. Please fix the SQL.

[Revised SQL #2]

```
SELECT customers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id WHERE
orders.order_status = "On Road" INTERSECT SELECT customers.customer_name FROM customers JOIN orders ON
customers.customer_id = orders.customer_id WHERE orders.order_status = "Shipped"
```

[Explanation summary #2]

As in your explanation, the SQL query returns a table with 1 column, the customer names who have the order status "On Road" and "Shipped". The question returns 1 column, the customer names who have order status of both "On Road" and "Shipped". So the question returns 1 column.

[Feedback #2]

So the SQL prediction above is correct!

Safety

- Correctness
- Coding models can accidentally push your code to the main branch
- Can delete the test cases if instructed to maybe “Pass the test cases”
- Faking personal information
- Pushing or generating malicious codes as an attack form

Can we guardrail them?



- Sandboxing
- Credentialization
- Auditing

Safety (SeCodePLT)

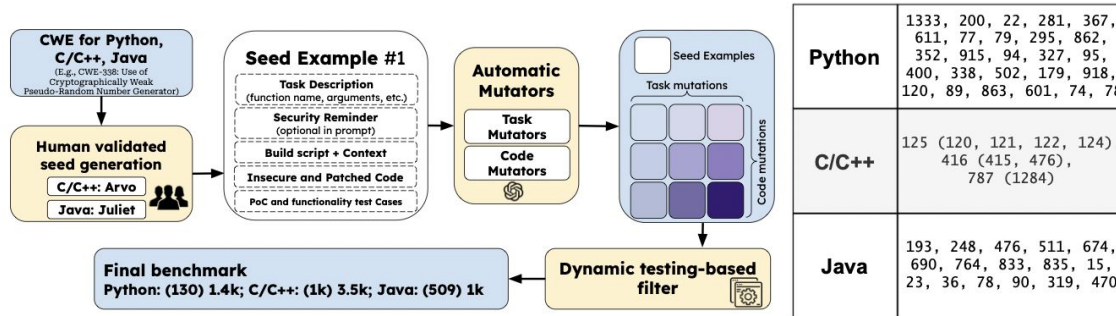


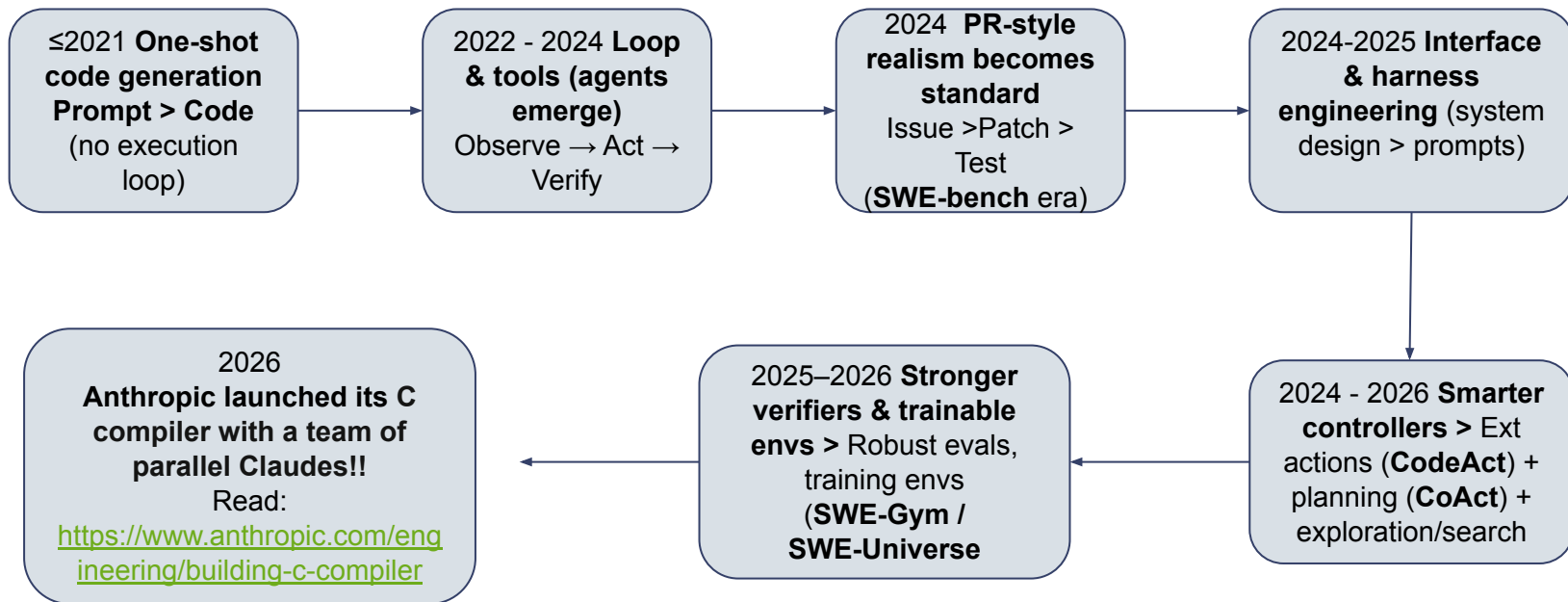
Figure 1: Our two-stage data creation pipeline.

Python	1333, 200, 22, 281, 367, 611, 77, 79, 295, 862, 352, 915, 94, 327, 95, 400, 338, 502, 179, 918, 120, 89, 863, 601, 74, 78
C/C++	125 (120, 121, 122, 124), 416 (415, 476), 787 (1284)
Java	193, 248, 476, 511, 674, 690, 764, 833, 835, 15, 23, 36, 78, 90, 319, 470

Figure 2: Risk categories.

- Evaluates **three security capabilities**: secure code generation, vulnerability detection, and patch generation
- Dynamic metrics (LLM-judging mostly does static evaluation)
- For each scenario, they provide >
 - vulnerable codepatched code, functionality tests and security tests
- Multi-language support (built for Python, C/C++, and Java)

Evolutions



Takeaway

What started as a code generation blocks in chat prompts, is going towards doing parallelism with multi agents, long running harnesses, and full fledged coding platforms for industry.

Trade off(?) Are we as a software developer giving away our jobs, or just merely making systems to make our lives easier?

Any Questions



Questions

Now for the presentations, but before that QUIZ!

Giving 2 minutes of dedicated time for participation quiz