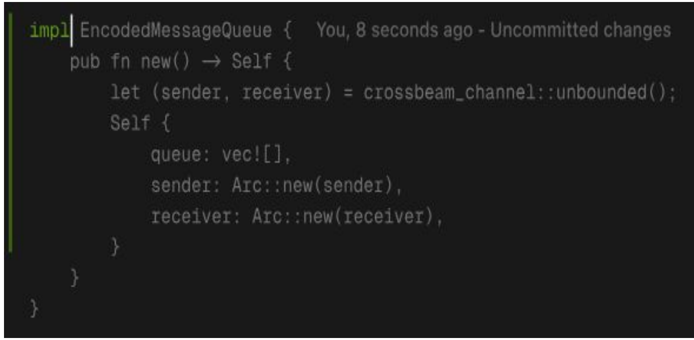
Large Language Model Coding

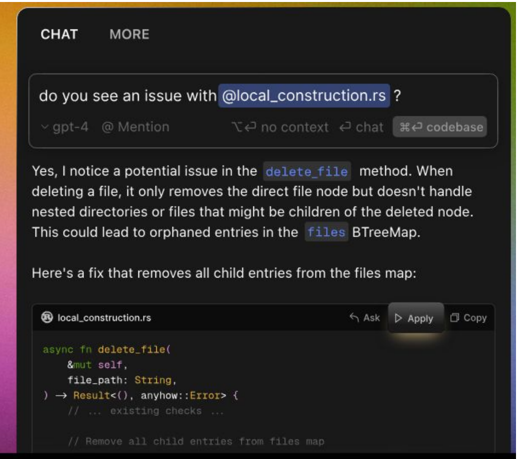
Guangya Wan, Joseph Moretto

CS 6501: Natural Language Processing
March 11, 2025

Current LLM Coding Tools

- ➢ Cursor
  - Coding assistant fork of visual studio, LLM integrated into the UI
  - Can use a variety of models, Claude 3.5 Sonnet is recommended for general coding
- ➢ Features
  - Auto predict suggestions for written code
  - Chat with the model always have the files in the context
  - CTRL K, edit existing code or create new code





Current LLM Coding Tools

- ➢ Github Copilot, integrated with a variety of IDEs
  - Github integration
- ➢ Very similar to cursor in features
- ➢ Mixed general sentiment on cursor vs copilot
- ➢ Both free with limited features

Feature	Cursor	Copilot
Terminal integration	✓	✓
Tab completion	✓	✓
Language agnostic	✓	✓
API access	✗	✓
Self hostable	✗	✗
Test generation	✓	✓
Real-time completions	✓	✓
Usage analytics	✗	✗
Explanations/Chat	✓	✓
Full codebase context	✓	✓
VS Code Support	✓	✓
JetBrains Support	✗	✓
NVIM Support	✗	✓
Models Supported	GPT-4, Claude, Custom	GPT-4, Custom Models
Pricing	Free Hobby Tier + Pro $20/mo + Business $40/user/mo	Individual $10/mo, Business $19/mo, Enterprise $39/mo

https://www.greptile.com/blog/comparing-cursor-vs-copilot

Published as a conference paper at ICLR 2023

InCoder: A Generative Model for Code Infilling and Synthesis

Daniel Fried[*♡†◇] **Armen Aghajanyan**[*♡] **Jessy Lin**[♣]
Sida Wang[♡] **Eric Wallace**[♣] **Freda Shi**[△] **Ruiqi Zhong**[♣]
Wen-tau Yih[♡] **Luke Zettlemoyer**[♡†] **Mike Lewis**[♡]
Facebook AI Research[♡] University of Washington[†]
UC Berkeley[♣] TTI-Chicago[△] Carnegie Mellon University[◇]
dfried@cs.cmu.edu, {armenag,mikelewis}@fb.com

Introduction and Purpose

- Much progress has been made on generative coding
- Not all coding is done left to right
  - Refinement and editing often take place
- Introduce InCoder
- Can generate left to right, but can also infill arbitrary regions of code
- This contribution allows for a more effective coding assistant through infilling

Infilling and Synthesis

- Most neural models utilize left to right generation (casual)
  - autoregressive language modeling objective
  - Predicts next token using all previous tokens
- Bert utilizes a masked language modeling objective
  - Predicts next token using surrounding context
- InCoder utilizes a causal masking objective
  - Predicts next token using all previous tokens, but masks future tokens

Training

- Casual masking procedure samples spans of contiguous tokens to mask
- Each span (k) is replaced with mask token, <Mask:k>
  - Prepended to the document
- End-of-mask token <EOM> token appended
- Left and Right context
- Maximize log probability of masked document

log P ([Left; <Mask:0>; Right; <Mask:0>; Span; <EOM>])

Masking Example

Training

Original Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Masked Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        <MASK:0> in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
<MASK:0> word_counts = {}
        for line in f:
            for word in line.split():
                if word <EOM>
```

Inference Functionality

- Can be used as a traditional left to right code generation
- Additionally can attributability infill at by inserting <Mask:k> tokens
- Generates a span to insert into Left and Right context sequences by sampling tokens autoregressively from the distribution
- Continues until <EOM> token is generated or a task-dependent stopping criterion is achieved

$$P(\cdot \mid [\text{Left}; \text{<Mask:0>}; \text{Right}; \text{<Mask:0>}])$$

Zero-shot Inference

Type Inference

```python
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Variable Name Prediction

```python
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_count = {}
        for line in f:
            for word in line.split():
                if word in word_count:
                    word_count[word] += 1
                else:
                    word_count[word] = 1
    return word_count
```

Docstring Generation

```python
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.

    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
            word_counts = {}
            for line in f:
                for word in line.split():
                    if word in word_counts:
                        word_counts[word] += 1
                    else:
                        word_counts[word] = 1
        return word_counts
```

Multi-Region Infilling

```python
from collections import Counter

def word_count(file_name):
    """Count the number of occurrences of each word in the file."""
    words = []
    with open(file_name) as file:
        for line in file:
            words.append(line.strip())
    return Counter(words)
```

Models

- Primary model 6.7B Transformer
  - Fairseq Architecture
  - InCoder-6.7B
- Trained on
  - Public code with permissive, non-copyleft, open-source licenses from GitHub and GitLab
  - StackOverflow questions, answers, and comments.
  - 159 GB of code Primarily Python, but contains 28 other languages

Infilling Experimentation

- 3 inference methods for testing
  - Causal masking
  - Left-to-right single
  - Left-to-right reranking

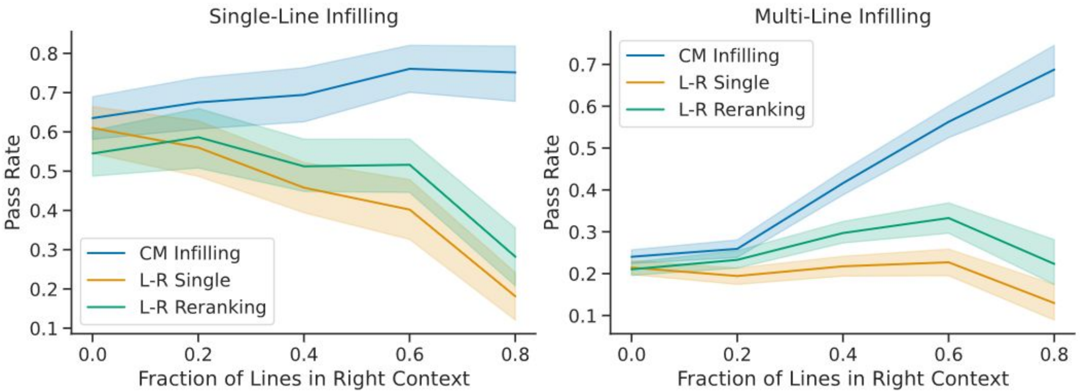
Method	Pass Rate	Exact Match
L-R single	48.2	38.7
L-R reranking	54.9	44.1
CM infilling	69.0	56.3
PLBART	41.6	—
code-cushman-001	53.1	42.0
code-davinci-001	63.0	56.0

(a) Single-line infilling.

Method	Pass Rate	Exact Match
L-R single	24.9	15.8
L-R reranking	28.2	17.6
CM infilling	38.6	20.6
PLBART	13.1	—
code-cushman-001	30.8	17.4
code-davinci-001	37.8	19.8

(b) Multi-line infilling.

Infilling Code



- Single - generate a single-line completion for a blank from a description of the function and the code lines before and after the blank
- Multi - multiple masked lines

Docstring Generation

- ➢ CodeXGLUE
  - Consists of code and corresponding documentation pairs
- ➢ Generate documentation and compare performance
- ➢ Significant because it trails of the performance of fine tuned
  - Zero shot setting no tuning vs fine tuned with 250k examples

Method	BLEU
Ours: L-R single	16.05
Ours: L-R reranking	17.14
Ours: Causal-masked infilling	18.27
RoBERTa (Finetuned)	18.14
CodeBERT (Finetuned)	19.06
PLBART (Finetuned)	19.30
CodeT5 (Finetuned)	20.36

Return Type Prediction

- ➢ Ability to see a function and predict the return type
  - ○ CodeXGLUE adapted benchmark
- ➢ Variable name prediction

Method	Accuracy
Left-to-right single	18.4
Left-to-right reranking	23.5
Causal-masked infilling	30.6

Method	Accuracy
Left-to-right single	12.0
Left-to-right reranking	12.4
Causal-masked infilling	**58.1**

Method	Precision	Recall	F1
Ours: Left-to-right single	30.8	30.8	30.8
Ours: Left-to-right reranking	33.3	33.3	33.3
Ours: Causal-masked infilling	**59.2**	**59.2**	**59.2**
TypeWriter (Supervised)	54.9	43.2	48.3

Ablation Experiments

- ➢ Study the effect on lower model size and modified training
- ➢ Casual mask vs left-right mask

#	Size (B)	Obj.	Training Data	Data Size	Train Tokens	Train Compute	HumanEval Pass@1	MBPP Pass@1
1)	6.7	CM	multi lang + SO	204 GB	52 B	3.0 Z	15	19.4
2)	1.3	CM	multi lang + SO	204 GB	52 B	0.6 Z	8	10.9
3)	1.3	LM	multi lang + SO	204 GB	52 B	0.6 Z	6	8.9
4)	1.3	LM	Python + SO	104 GB	25 B	0.3 Z	9	9.8
5)	1.3	LM	Python	49 GB	11 B	0.1 Z	5	6.1
6)	2.3	LM	multi lang + SO	204 GB	52 B	1.1 Z	9	12.7

Qualitative Examples

```
<| file ext=.py |>
# count the words in all files in the current directory
import os
import sys

def main():
  cwd = os.getcwd()

  words = 0

  for filename in os.listdir(cwd):
    if filename.endswith(".in"):
      fname = os.path.join(cwd, filename)
      with open(fname) as infile:
        for line in infile:
          words += len(line.split())

  print(words)

if __name__ == '__main__':
  main()
```

```
<| file ext=.sh |>
# count the words in all files in the current directory
find . -type f -name "*.txt" -exec wc -w {} \; | sort -nr | head -n 20
```

```
class Person:

    def __init__(self,name,age,gender):
        self.name=name
        self.age=age
        self.gender=gender

p = Person('Eren', 18, "Male")
```

Conclusion

- ➢ Demonstrated that casual masking objective allows for strong zero shot performance
- ➢ Allows for infilling without sacrificing left to right performance
- ➢ Performs comparable to similar models and trails fine tuned models

Limitations and Future work

- ➢ The InCoder model is only 6.7B
- ➢ Does not directly modify current code, only can mask and infill
  - ○ Harder to use for debugging
- ➢ Future work expects performance to increase with increased parameters, data, and training steps

SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering

John Yang* Carlos E. Jimenez* Alexander Wettig Kilian Lieret
Shunyu Yao Karthik Narasimhan Ofir Press

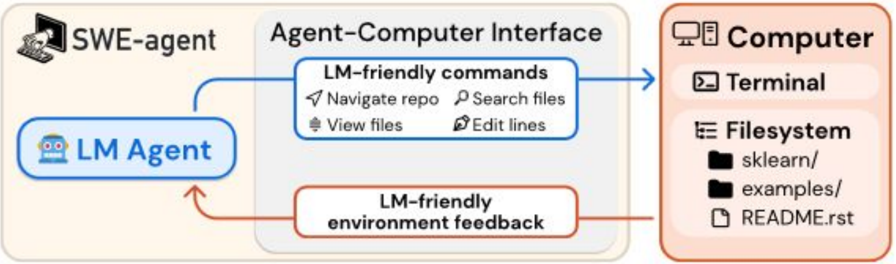
Princeton Language and Intelligence, Princeton University

Introduction

- ➢ Agentic language models are increasingly used to automate digital tasks
- ➢ Recent works prove code generation and execution feedback, but…
- ➢ Lack of complex and software engineering capabilities
- ➢ Current agents struggle to edit small segments and give feedback for invalid edits

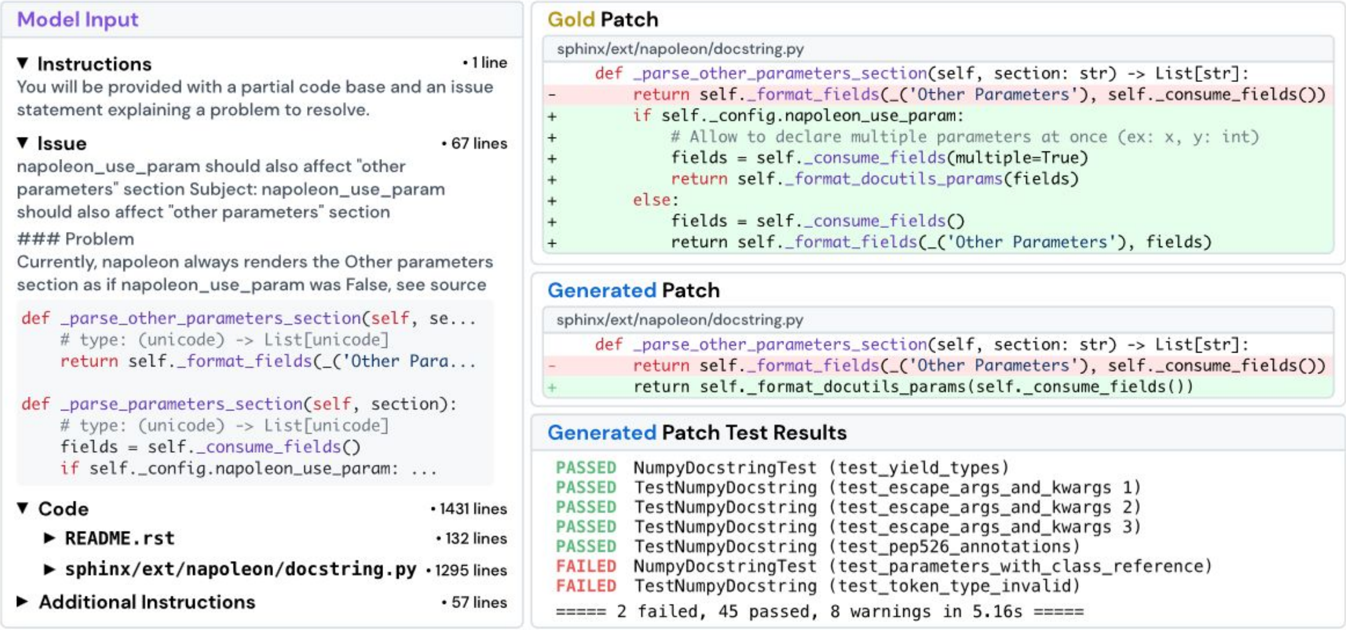
Introduction

- ➢ To address this SWE-agent attempts to better interface with software and computer systems
- ➢ Provides a set of actions with guardrails
  - ○ Agent receives and responds to system feedback
- ➢ SWE-agent solves 12.47% of SWE-benchmark
  - ○ Previous best was 3.8%



SWE-Benchmark

- ➢ Real world software engineering problems from github issues
  - Python



SWE-Benchmark

Metadata

Repo	sympy / sympy	**Issue #s**	[17006]
Instance ID	sympy__sympy-17022	**Pull Number**	17022
Created At	Jun 9, 2019	**Base Commit**	b6fbc76

Problem Statement

Using lambdify on an expression containing an identity matrix gives us an unexpected result:

```
>>> import numpy as np
>>> n = symbols('n', integer=True)
>>> A = MatrixSymbol("A", n, n)
>>> a = np.array([[1, 2], [3, 4]])
>>> f = lambdify(A, A + Identity(n))
>>> f(a)
array([[1.+1.j, 2.+1.j],
       [3.+1.j, 4.+1.j]])
```

Instead, the output should be `array([[2, 2], [3, 5]])` , since we're adding an identity matrix to the array. Inspecting the globals and source code of f shows us why we get the result:

```
>>> import inspect
>>> print(inspect.getsource(f))
def _lambdifygenerated(A):
    return (I + A)
>>> f.__globals__['I']
1j
```

The code printer prints `I`, which is currently being interpreted as a Python built-in complex number. Printer should support identity matrices ...

Test Patch

sympy/printing/tests/test_pycode.py [...]

```
 9    from sympy.logic import And, Or
10 -  from sympy.matrices import SparseMatrix, MatrixSymbol
11 +  from sympy.matrices import SparseMatrix, MatrixSymbol, Identity
12    from sympy.printing.pycode import (
...
46    def test_NumPyPrinter():
47        p = NumPyPrinter()
48        assert p.doprint(sign(x)) == 'numpy.sign(x)'
49        A = MatrixSymbol("A", 2, 2)
50        assert p.doprint(A**(-1)) == "numpy.linalg.inv(A)"
51        assert p.doprint(A**5) == "numpy.linalg.matrix_power(A, 5)"
52 +      assert p.doprint(Identity(3)) == "numpy.eye(3)"
```

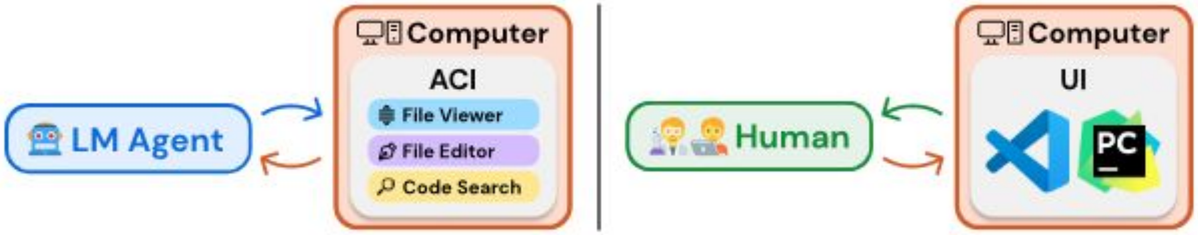
Gold Patch

sympy/printing/pycode.py

```
609              return "%s(%s)" % (func, self._print(expr.tolist()))
610
611 +    def _print_Identity(self, expr):
612 +        shape = expr.shape
613 +        if all([dim.is_Integer for dim in shape]):
614 +            return "%s(%s)" % (self._module_format('numpy.eye'),
     self._print(expr.shape[0]))
615 +        else:
616 +            raise NotImplementedError("Symbolic matrix dimensions are not
     yet supported for identity matrices")
617 +
618      def _print_BlockMatrix(self, expr):
```

Agent-Computer Interface

- ➢ The LLM is a new type of end user with it’s own needs
- ➢ An agent-computer interface (ACI) is analogous to code editors and similar software
- ➢ LLM agents currently do not have the visual understanding to use coding GUI applications
- ➢ Humans can ignore unnecessary information
    - ○ While any information has a cost for LLMs

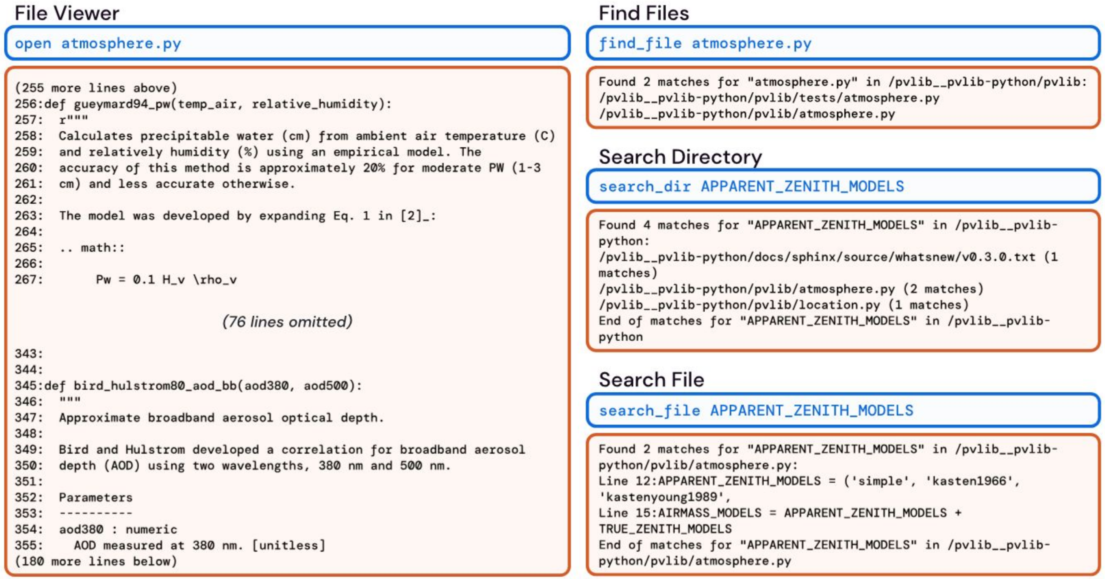


ACI Design

- A well designed ACI should have…
  - Simple and understandable commands
  - Commands should be compact and efficient
  - Environmental feedback, but kept concise
  - Guardrails such as a syntax checker

ACI for Software Engineering

- ➢ Search and Navigation commands
  - ○ Find_file, search_file, search_dir
- ➢ Suppresses verbose results
- ➢ Only shows 50 results, asks user to be more specific



ACI for Software Engineering

- ➢ File Viewer
  - Open, scroll, goto
- ➢ File editor
  - edit
- ➢ Context management
  - Generates both a thought and action
  - Repeats until valid

(a) Observation from the file viewer. (b) Action using the edit interface.

Error Message

```
Your output was not formatted correctly. You must always include one
discussion and one command as part of your response. Make sure you do
not have multiple discussion/command tags.
Please make sure your output precisely matches the following format:
DISCUSSION
Discuss here with yourself about what your planning and what you're
going to do in this step.

```
command(s) that you're going to run
```
```

Experimental Setup

- Baselines
  - RAG with no other tools (BM25 Retrieval System)
  - Agents with access to shell
- Dataset
  - SWE Bench
    - 2294 instances of real world github issues
    - All python repository
    - Superset of SWE Bench Lite (a subset, for detailed analysis)
  - HumanEvalFix
    - Debugging Benchmark Covering various PLs
- Metrics
  - Pass @ 1
  - Cost ($)
- Models used:
  - GPT-4 Turbo
  - Claude 3 Opus

Main Results

- SOTA Performance
  - Significance Performance gains on both dataset and models
  - Most costs than baseline models

Table 1: Main results for SWE-agent performance on the full and Lite splits of the SWE-bench test set. We benchmark models in the SWE-agent, Basic CLI, and Retrieval Augmented Generation (RAG) settings established in SWE-bench [20].

	SWE-bench		SWE-bench Lite	
Model	% Resolved	$ Avg. Cost	% Resolved	$ Avg. Cost
RAG				
w/ GPT-4 Turbo	1.31	0.13	2.67	0.13
w/ Claude 3 Opus	3.79	0.25	4.33	0.25
Shell-only agent				
w/ GPT-4 Turbo	-	-	11.00	1.46
w/o Demonstration	-	-	7.33	0.79
SWE-agent				
w/ GPT-4 Turbo	**12.47**	1.59	**18.00**	1.67
w/ Claude 3 Opus	10.46	2.59	13.00	2.18

Table 3: SWE-bench Lite performance under ablations to the SWE-agent interface, which is denoted by . We consider different approaches to searching and editing (see Figures 5 and 6, respectively). We also verify how varying the file viewer window size affects performance, and we ablate the effect of different context management approaches.

Editor		Search		File Viewer		Context	
`edit` action	15.0 ↓3.0	Summarized	18.0	30 lines	14.3 ↓3.7	Last 5 Obs.	18.0
w/ linting	18.0	Iterative	12.0 ↓6.0	100 lines	18.0	Full history	15.0 ↓3.0
No `edit`	10.3 ↓7.7	No search	15.7 ↓2.3	Full file	12.7 ↓5.3	w/o demo.	16.3 ↓1.7

Table 2: Pass@1 results on HumanEvalFix [32]. Except for SWE-agent, we use scores as reported in Yu et al. [65].

Model	Python	JS	Java
CodeLLaMa-instruct-13B	29.2	19.5	32.3
GPT-4	47.0	48.2	50.0
DeepseekCoder-CodeAlpaca-6.7B	49.4	51.8	45.1
WaveCoder-DS-6.7B	57.9	52.4	57.3
SWE-agent w/ GPT-4 Turbo	**87.7**	**89.7**	**87.9**

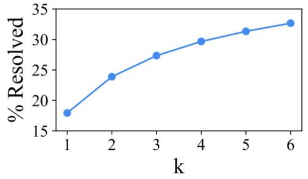


Figure 4: SWE-agent w/ GPT-4 Turbo Pass@k performance across 6 runs on SWE-bench Lite.

Human user interfaces are not always suitable as agent-computer interfaces.

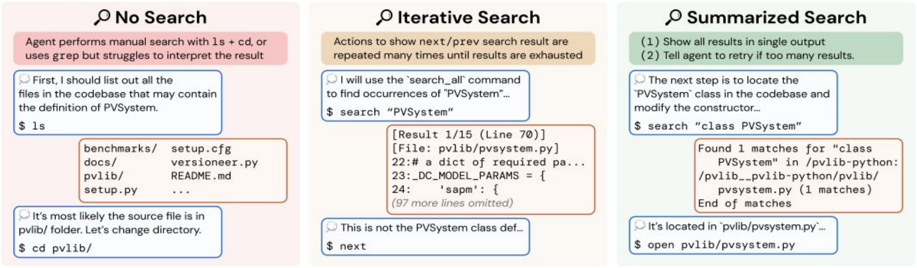


Figure 5: Three different Search interfaces for task instance pvlib__pvlib-python-1224. In Shell-only, an agent performs localization using only standard bash commands and utilities. Compared to *Iterative* search, *Summarized* search shows an exhaustive list of search results and provides guidance on refining under-specified queries.

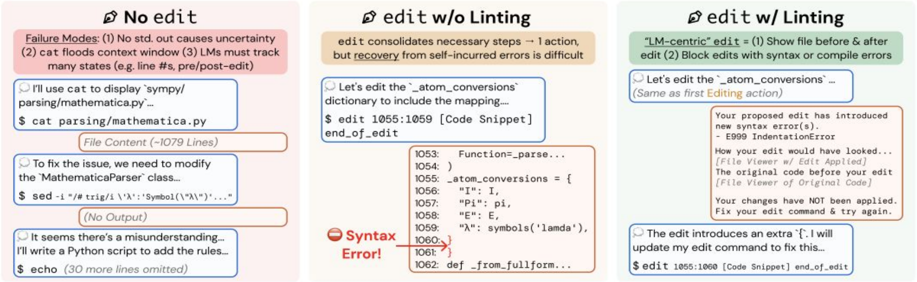


Figure 6: Three different Edit interfaces for task instance sympy__sympy-24102. Editing with bash commands requires several actions to successfully modify a file. The *Editing* component defines an edit command that leverages the File Viewer component to replace the bash style of editing workflow with a single command. *Linting* is beneficial for stymieing cascading errors that often start with an error-introducing edit by the agent.

Trace and Failure Analysis

- ➢ Reproduction and/or localization is the first step
- ➢ Remaining turns are mostly “edit, then execute" loops
- ➢ Most failures are incorrect implementations

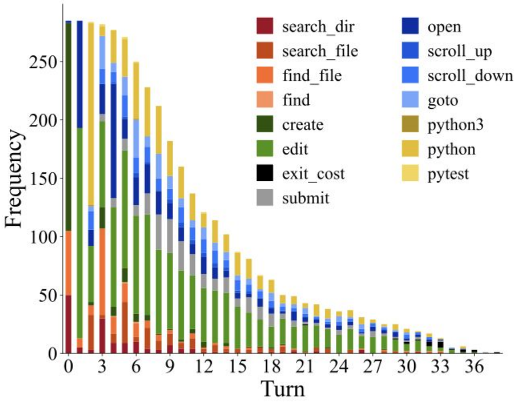


Figure 7: The frequency with which actions are invoked at each turn by SWE-agent w/ GPT-4 for task instances that it solved on the SWE-bench full test set (286 trajectories).

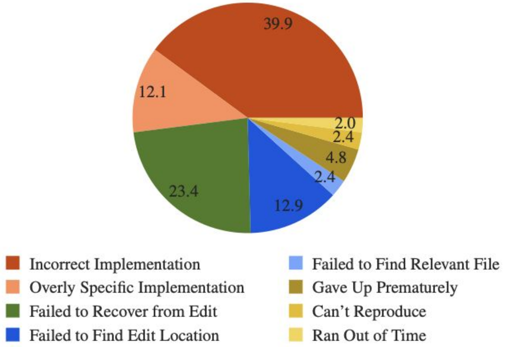


Figure 8: Failure mode distribution for SWE-agent w/ GPT-4 Turbo trajectories of unresolved instances. Each instance is labeled automatically using an LM with the categories from Table 9.

Summary of Findings

- **SWE-agent, utilizing long-context Models like GPT-4 Turbo and a tailored Agent-Computer Interface (ACI), significantly outperforms existing approaches on the challenging real-world software engineering tasks in SWE-bench and Human Eval Fix like coding editing and debugging**
- **The ACI designed specifically for LMs outperforms user interfaces (UIs) designed for human users, such as the Linux shell**
- **Analysis of agent behavior reveals recurring problem-solving patterns**
  - **Agents tend to succeed quickly and fail slowly.**
  - **The interactive setting of SWE-agent enables more effective file localization compared to non-interactive retrieval methods.**
  - **Reproduction and/or localization is the first step.**
  - **Guardrails can improve error recovery**
  - **More on appendix.**

Conclusion

- **Demonstrates the Power of Agent-Computer Interfaces (ACIs), which Highlights the Differences Between Human and LM Interface Needs**
- **Provides Design Principles for ACIs**
  - Actions should be simple and easy to understand for agents
  - Actions should be compact and efficient
  - Environment feedback should be informative but concise
- **Opens New Avenues for Research on Language Models and Agents** and **Sets a New State-of-the-Art for Automated Software Engineering**

Limitation and Future

- ➢ Limited number of Actions
- ➢ Cross-Domain or Automated ACI Design
- ➢ Error Recoveries
- ➢ Prompting Robustness and non-prompting Methods

Teaching Large Language Models to Self-Debug

Xinyun Chen[1] **Maxwell Lin**[2] **Nathanael Schärli**[1] **Denny Zhou**[1]
[1] Google DeepMind [2] UC Berkeley
{xinyunchen,schaerli,dennyzhou}@google.com, mxlin@berkeley.edu

Introduction and Motivation

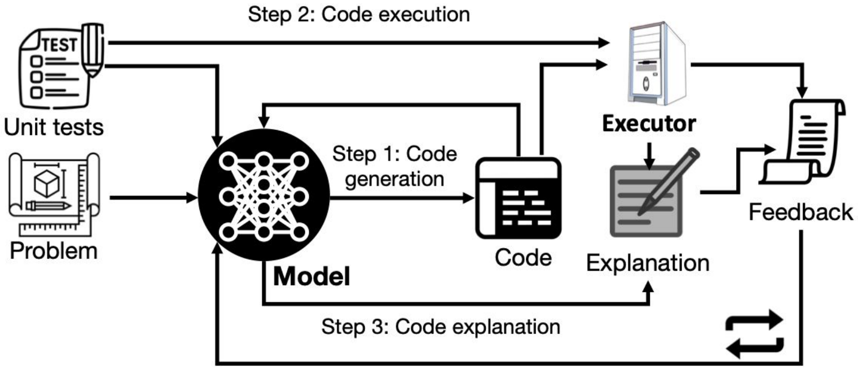
- ➢ Much progress has been made on Large Language Models (LLMs) in code generation.
- ➢ generating correct code for complex tasks in a single attempt remains difficult for LLMs due to its autoregress nature.
- ➢ Introduce SELF-DEBUGGING, a novel approach to teach LLMs to debug their own code using few-shot demonstrations with additional training.

Related Work

- ➢ Prompting techniques like chain-of-thought significantly improve programming
- ➢ language models have potential for self correction, generating feedback messages to critique and refine their outputs
- ➢ Trained special LLM for coding (code LLama), but requires too many computational resources.

Self-Debugging Framework

- ➢ Enables models to identify mistakes by analyzing execution results and explaining the generated code in natural language (rubber duck debugging)
- ➢ Generation: The model predicts candidate programs
- ➢ Explanation: The model processes predictions by explaining them in natural language or creating execution traces
- ➢ Feedback: A message about the code's correctness is generated, either by the model itself or from existing unit tests.
- ➢ Process can be iterative, continuing until the code is deemed correct or a maximum number of turns is reached



Feedback Mechanism

- ➢ Simple Feedback: Basic indication of code correctness.
- ➢ Unit Test Feedback: Utilizing unit test results for richer debugging information
- ➢ Code Explanation Feedback: The model explains its own code to identify errors
- ➢ Process can be iterative, continuing until the code is deemed correct or a maximum number of turns is reached

Feedback Mechanism

Simple Feedback	Unit Test (UT) Feedback	Unit Test + Explanation (+Expl.)	Unit Test + Trace (+Trace)
Below are C++ programs with incorrect Python translations. Correct the translations using the provided feedback. [C++] [Original Python] [Simple Feedback] [Revised Python #1] [Simple Feedback] [Revised Python #2] ...	Below are C++ programs with incorrect Python translations. Correct the translations using the provided feedback. [C++] [Original Python] [UT Feedback] [Revised Python #1] [UT Feedback] [Revised Python #2] ...	Below are C++ programs with incorrect Python translations. *Explain the original code, then explain the translations line by line* and correct them using the provided feedback. [C++] [C++ Explanation] [Original Python] [Python Explanation] [UT Feedback] [Revised Python #1] [Python Explanation] [UT Feedback] [Revised Python #2] [Python Explanation] ...	Below are C++ programs with incorrect Python translations. Using the provided feedback, *trace through the execution of the translations to determine what needs to be fixed*, and correct the translations. [C++] [Original Python] [UT Feedback] [Trace] [Revised Python #1] [UT Feedback] [Trace] [Revised Python #2] ...

Datasets and Examples

- Text to Code(Spider, MBPP)
- Code Translation (Transcoder)
- Some with and some without Unit Tests.

C++ Program

```
string caesar_cipher ( string text,
int s ) {
  string result = "";
  for ( int i = 0;
  i < text . length ( );
  i ++ ) {
    if ( isupper ( text [ i ] ) )
    result += char ( int ( text [ i ]
    + s - 65 ) % 26 + 65 );
    else result += char ( int ( text [
     i ] + s - 97 ) % 26 + 97 );
  }
  return result;
}
```

Python Program

```
def caesar_cipher(text, s):
    result = ''
    for i in range(len(text)):
        char = text[i]
        if char.isupper():
            result += chr((((ord(char
            ) + s) - 65) % 26) + 65))
        else:
            result += chr((((ord(char
            ) + s) - 97) % 26) + 97))
    return result
```

Unit Tests

```
assert caesar_cipher('35225904', 2) == 'ikhhkofj'
... (8 unit tests omitted)
assert caesar_cipher('11', 93) == 'tt'
```

Figure 4: An example from the TransCoder dataset. The problem description contains the C++ program and unit tests, and the model is required to predict the Python program.

Problem description

```
CREATE TABLE customers (
customer_id number ,
customer_name text ,
customer_details text ,
primary key ( customer_id )
)
insert into customers (customer_id, customer_name, customer_details) values (1,
 'Savannah', 'rerum') ;

CREATE TABLE orders (
order_id number ,
customer_id number ,
order_status text ,
date_order_placed time ,
order_details text ,
primary key ( order_id ),
foreign key ( customer_id ) references customers ( customer_id )
)
insert into orders (order_id, customer_id, order_status, date_order_placed,
order_details) values (1, 2, 'Shipped', '2009-02-21 15:26:19', NULL);
...

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

Output SQL

```
SQL: 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"
```

Prompting for Self-Debugging

- few shot prompting for initial code attempt
- decode multiple samples, using majority voting on execution results to select predicted code
- filter out programs that do not pass unit tests

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 uustomers.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!

Figure 3: An example of SELF-DEBUGGING prompting for text-to-SQL generation. The SQL query, explanation and feedback are all predicted by the model. When the returned table has more than 2 rows, only the first 2 rows are included in the prompt. Database information is omitted in the figure for clarity, and we present the full prompts in Appendix E.

Evaluations

- Baselines
  - Trained Code LLM
    - T5/LEVER
  - Prompting without self correction (By generating multiple candidates)
    - Coder Reviewer
      - Selects the program by utilizing both the likelihood of the predicted code given the problem description and and the likelihood of the problem description given the predicted code
    - MBR-Exec
      - Selects the program with the most frequent output
- Metrics
  - Accuracy
  - Pass @ K

(a) Results on the Spider development set.

	Spider (Dev)
w/ training	
T5-3B + N-best Reranking	80.6
LEVER (Ni et al., 2023)	81.9
Prompting only w/o debugging	
Coder-Reviewer	74.5
MBR-Exec	75.2
SELF-DEBUGGING (this work)	
Codex	81.3
+ Expl.	**84.1**

(b) Results on MBPP dataset.

	n samples
Prior work	
MBR-Exec	63.0 ($n = 25$)
Reviewer	66.9 ($n = 25$)
LEVER	68.9 ($n = 100$)
SELF-DEBUGGING (this work)	
Codex	72.2 ($n = 10$)
Simple	73.6
UT	75.2
UT + Expl.	**75.6**

Different Forms of Feedback

E.g Unit Test

From MBPP

(a) Results on the Spider development set.

Spider	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	81.3	71.1	73.2	64.7
Simple	81.3	**72.2**	73.4	**64.9**
+Expl.	**84.1**	**72.2**	**73.6**	**64.9**

(b) Results on TransCoder.

TransCoder	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	80.4	89.1	77.3	70.0
Simple	89.3	91.6	80.9	72.9
UT	91.6	**92.7**	88.8	76.4
+ Expl.	**92.5**	**92.7**	**90.4**	**76.6**
+ Trace.	87.9	92.3	89.5	73.6

(c) Results on MBPP.

MBPP	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	61.4	67.6	72.8	47.2
Simple	68.2	70.8	78.8	50.6
UT	69.4	72.2	**80.6**	52.2
+ Expl.	69.8	**74.2**	80.4	52.2
+ Trace.	**70.8**	72.8	80.2	**53.2**

Original Python code

```
assert encode_list
([1,1,2,3,4,4.3,5,1])==[[2, 1], [1,
2], [1, 3], [1, 4], [1, 4.3], [1, 5],
[1, 1]

Write a function to reflect the run-
length encoding from a list.
```

```python
def encode_list(nums):
    res = []
    for i in nums:
        if i not in res:
            res.append([nums.count(i),
            i])
    return res
```

Prediction after self-debugging

```python
def encode_list(nums):
  res = []
  count = 1
  for i in range(1, len(nums)):
    if nums[i] == nums[i-1]:
      count += 1
    else:
      res.append([count, nums[i-1]])
      count = 1
  res.append([count, nums[-1]])
  return res
```

Figure 11: An example on MBPP where the prediction after SELF-DEBUGGING is very different from the initial code.

Ablation Study

➢

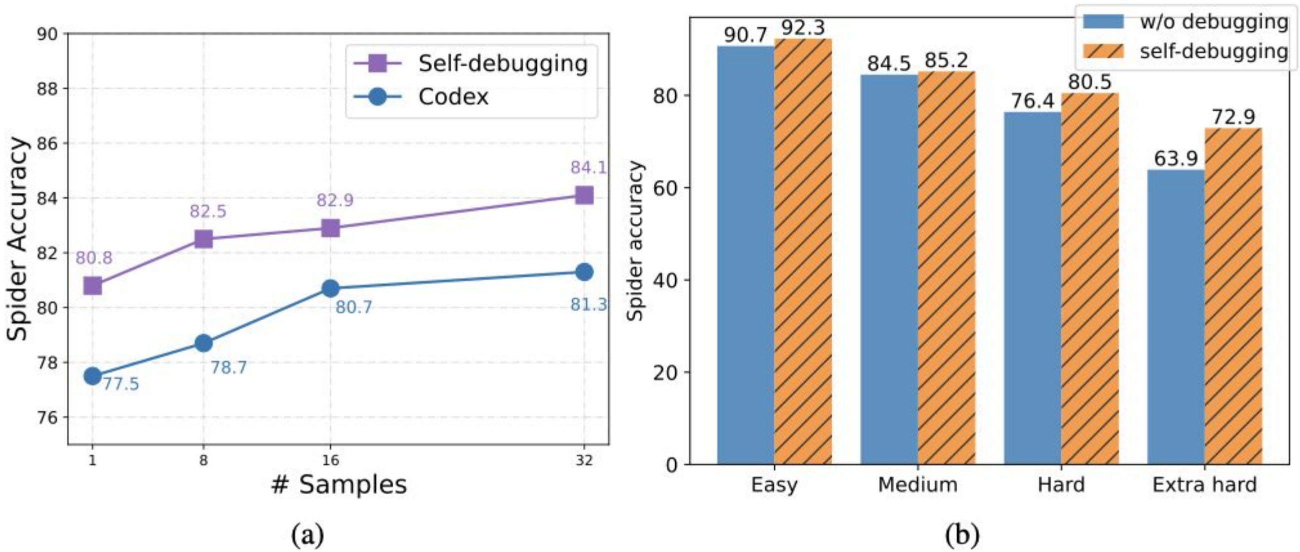


Figure 6: Ablation studies on the Spider development set with Codex. (a) Accuracies with different numbers of initial samples. (b) Breakdown accuracies on problems with different hardness levels.

Importance of Unit Tests

Table 3: Results of SELF-DEBUGGING without unit test execution.

(a) Results on Transcoder.

TransCoder	Codex	GPT-3.5	GPT-4
Baseline	80.4	**89.1**	77.3
Simple	83.4	**89.1**	78.2
+ Expl.	**83.9**	**89.1**	78.0
+ Trace.	**83.9**	**89.1**	**78.4**

(b) Results on MBPP

MBPP	Codex	GPT-3.5	GPT-4
Baseline	61.4	67.6	72.8
Simple	57.6	68.2	76.0
+ Expl.	64.4	68.2	76.0
+ Trace.	**66.2**	**69.2**	**76.4**

Examples

Original SQL

```
Question: Give the names of countries with English and French as official
languages.

SELECT country.name FROM country
JOIN countrylanguage ON country.code = countrylanguage.countrycode
WHERE countrylanguage.language = "English"
INTERSECT SELECT country.name FROM country
JOIN countrylanguage ON country.code = countrylanguage.countrycode
WHERE countrylanguage.language = "French"
```

Code explanation

```
The execution of the SQL query above would return a table with 1 column.
The first column, ‘‘country.name’’ would contain the country name.
With ‘‘country JOIN countrylanguage’’, the table would contain the data about
countries and languages.
With ‘‘WHERE countrylanguage.language = ’English’’’, the table filters the
records to only include countries with the language ‘‘English’’.
With ‘‘WHERE countrylanguage.language = ’French’’’, the table filters the
records to only include countries with the language ‘‘French’’.
With ‘‘INTERSECT’’, the table would be filtered to only include countries with
both ‘‘English’’ and ‘‘French’’ as languages.
So the SQL query returns a table with 1 column, the country name of countries
with both ‘‘English’’ and ‘‘French’’ as languages.
```

Self-debugging with explanation

```
As in your explanation, the SQL query returns a table with 1 column, the
country name of countries with both ‘‘English’’ and ‘‘French’’ as languages.
The question returns the names of countries with English and French as offi-
cial languages. So the SQL prediction above is wrong. Please fix the SQL.
```

```
SELECT country.name FROM country JOIN countrylanguage ON country.code =
countrylanguage.countrycode
WHERE countrylanguage.language = "English"
AND countrylanguage.isofficial = "T"
INTERSECT SELECT country.name FROM country JOIN countrylanguage ON country.code
 = countrylanguage.countrycode WHERE countrylanguage.language = "French"
AND countrylanguage.isofficial = "T"
```

Figure 7: An example on the Spider benchmark, where SELF-DEBUGGING fixes the prediction for an extra hard problem. Database information is omitted in the figure for clarity.

Results Summary

- SELF-DEBUGGING framework, particularly when combined with appropriate feedback mechanisms like code explanations and unit tests, significantly enhances the code generation capabilities of various large language models across different challenging benchmarks
  - SOTA on all three datasets with or without unit tests for verification
  - Combining unit test feedback with code explanation feedback resulted in further performance gains.

Conclusion

- ➢ SELF-DEBUGGING is an effective approach to improve the accuracy and sample efficiency of code generation by LLMs without additional training
- ➢ Simple as leveraging the model's ability to explain its own code and analyze different forms of feedbacks can lead to significant performance gains
- ➢ Highlights the potential of using simple yet effective techniques, inspired by human debugging practices (rubber ducker debugging)

Limitations and Future

- ➢ Depends on the Explanation capability of LLMs (no test on smaller LLMs)
- ➢ The robustness of the approach to different prompt variations and the potential for further optimization of the prompts could be explored.
- ➢ Computational cost of the iterative debugging process might be a limitation for real-world applications, especially for very complex code generation tasks.
- ➢ Other forms of feedback