Instruct, Not Assist: LLM-based Multi-Turn Planning and Hierarchical Questioning for Socratic Code Debugging

Priyanka Kargupta*, Ishika Agarwal*,
Dilek Hakkani-Tur, Jiawei Han

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CONVAI

Motivation

Student: Hey, I really need help. My Fibonacci function is not returning the correct number.

Single-Turn Assistant: I'll be happy to help you. You made an error in the recursive function. Here is the corrected code:

```
def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        sequence = fibonacci(n-1)
        sequence.append(sequence[-1] + sequence[-2])
        return sequence
```

Motivation

Student: Hey, I really need help. My Fibonacci function is not returning the correct number.

Multi-Turn Instructor: I'll be happy to help you. **How is the Fibonacci sequence defined and computed?**

Student: The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones.

Instructor: That is correct! Let's take a step further. **How are you utilizing recursion for computing the sequence?**

...

Student: Oh, I see. *I made an error in the recursive condition. I should change n-2 to n-1.*

Problem Formulation

Application: teaching code debugging

Input: (1) problem statement, (2) buggy code, (3) correct code, (4) bug fixes

Goal: enable the *Instructive* LLM to ask Socratic questions that is personalized to the Student's learning

Output: list of final bug fixes from the Student

“Sequential” Example

Given int x and list seq, return the index of the first occurrence of x in seq.

If x is not in seq, return the index where x should be inserted (assuming ascending order).

search(5, [-1, 5, 8]) → 1

Search (6, [-1, 5, 8]) → 2

```
def search(x, seq):
    for i in range(len(seq)):
        if x < seq[i]:
            return i
    return len(seq)
```

“Sequential” Example

Given int x and list seq, return the index of the first occurrence of x in seq.

If x is not in seq, return the index where x should be inserted (assuming ascending order).

search(5, [-1, 5, 8]) → 1

Search (6, [-1, 5, 8]) → 2

```
def search(x, seq):
    for i in range(len(seq)):
        if x < seq[i]:
            return i
    return len(seq)
```

“Sequential” Example

Given int x and list seq, return the index of the first occurrence of x in seq.

If x is not in seq, return the index where x should be inserted (assuming ascending order).

search(5, [-1, 5, 8]) → 1

Search (6, [-1, 5, 8]) → 2

```
def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
    return len(seq)
```

Methodology: agents

Teachers have multiple roles:

1. Asking Socratic questions
2. Verifying student answers
3. Updating student understanding

Methodology: agents

Teachers have multiple roles:

1. Asking Socratic questions → Instructor Agent
2. Verifying student answers } → Verifier Agent
3. Updating student understanding

Methodology: algorithm

1. δ←Verifier generates state
2. Instructor generates question
3. While bugs remain
    a. α ← **Student** responds
    b. Verifier validates α
    c. If α is wrong:
        i. Instructor generates sibling question
    d. Else:
        i. Verifier updates **δ**
        ii. If any updates:
            1. Instructor asks **Student** to generate bug fixes
        iii. Else:
            1. Instructor asks **Student** a child question

Methodology: algorithm

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Methodology: state space estimation

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

To help a student, human teachers:

1. Estimate what their student knows/doesn't
2. Understand what steps are required to go from buggy code to correct code
3. Adapt to a student depending on their answers mid-conversation

Methodology: state space estimation

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

How do we represent the Student's knowledge?

- State space: the set of possible steps to solve a problem from start to finish

Methodology: state space estimation

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

How do we represent the Student's knowledge?

1. Understand that the function should return the index of the first occurrence of x in seq
2. Recognize that the function should return the index where x should be inserted to keep seq sorted if x is not in seq
3. Correctly modify the comparison operator on line 3 to use <= instead of <

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Ground question generation on:

- Buggy code
- Ground truth bug fixes
- State Variable #1: Understand that the function should return the index of the first occurrence of x in seq

Methodology: question generation

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: What do you think the function should return when x is found in the sequence seq for the first time?

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier compares ground truth list of bug fixes the to **Student**'s bug fixes.

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier compares ground truth list of bug fixes the to **Student**'s bug fixes.

Ground truth BF:

- Replace `<` with `<=`

Student BF:

-

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: What do you think the function should return when x is found in the sequence seq for the first time?

Student response: The function should return the index of the position where x is found in seq.

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: What do you think the function should return when x is found in the sequence seq for the first time?

Student response: The function should return the index of the position where x is found in seq.

Verifier assessment:

answer_addresses_question: True

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: What do you think the function should return when x is found in the sequence seq for the first time?

Student response: The function should return the index of the position where x is found in seq.

Verifier assessment:

answer_addresses_question: True

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: What do you think the function should return when x is found in the sequence seq for the first time?

Student response: The function should return the index of the position where x is found in seq.

Verifier assessment:

answer_addresses_question: True

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier checks if each state variable is solved:

1. return the index of the first occurrence: True
2. return the index where x should be inserted: False
3. modify the comparison operator: False

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates δ
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Verifier checks if each state variable is solved:

1. return the index of the first occurrence: True
2. return the index where x should be inserted: False
3. modify the comparison operator: False

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Student generates fixes based on any modifications suggested in the conversation.

There are none, so no bug fixes.

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Yes!

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Ground question generation on:

- Buggy code
- Ground truth bug fixes
- State Variable #2: Recognize that the function should <u>return the index where x should be inserted</u> to keep seq sorted if x is not in seq

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: That is correct! Let's take a step further. What would happen when x is not found in seq?

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: That is correct! Let's take a step further. What would happen when x is not found in seq?

Student response: [...] an incorrect index being returned

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: That is correct! Let's take a step further. What would happen when x is not found in seq?

Student response: [...] an incorrect index being returned

Verifier assessment:

answer_addresses_question: False

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: That is correct! Let's take a step further. What would happen when x is not found in seq?

Student response: [...] an incorrect index being returned

Verifier assessment:

answer_addresses_question: False

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: That is correct! Let's take a step further. What would happen when x is not found in seq?

Student response: [...] an incorrect index being returned.

Verifier assessment:

answer_addresses_question: False

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier checks if each state variable is solved:

1. return the index of the first occurrence: True
2. return the index where x should be inserted: False
3. modify the comparison operator: False

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier checks if each state variable is solved:

1. return the index of the first occurrence: True
2. return the index where x should be inserted: False
3. modify the comparison operator: False

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Asks a question that digs deeper into the content of the previous question.

Instructor question: That's not entirely correct, let me rephrase the question. What would happen if the function returns the index of the occurrence of x in seq when x is not found in seq?

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: That's not entirely correct, let me rephrase the question. What would happen if the function returns the index of the occurrence of x in seq when x is not found in seq?

Student response: If x is not already in seq, then the correct answer would be the index immediately following the element bigger than x in seq.

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: That's not entirely correct, let me rephrase the question. What would happen if the function returns the index of the occurrence of x in seq when x is not found in seq?

Student response: If x is not already in seq, then the correct answer would be the index immediately following the element bigger than x in seq. [Change the condition to reflect that…]

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
   a. α ← **S** responds
   b. V validates α
   c. If α is wrong:
      i. I gens. sibling q
   d. Else:
      i. V updates **δ**
      ii. If any updates:
         1. I asks **S** to gen. BF
      iii. Else:
         1. I asks **S** a child q

Instructor question: That's not entirely correct, let me rephrase the question. What would happen if the function returns the index of the occurrence of x in seq when x is not found in seq?

Student response: If x is not already in seq, then the correct answer would be the index immediately following the element bigger than x in seq. [Change the condition to reflect that…]

Verifier assessment:

answer_addresses_question: True

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: That's not entirely correct, let me rephrase the question. What would happen if the function returns the index of the occurrence of x in seq when x is not found in seq?

Student response: If x is not already in seq, then the correct answer would be the index immediately following the element bigger than x in seq. [Change the condition to reflect that…]

Verifier assessment:

answer_addresses_question: True

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Instructor question: That's not entirely correct, let me rephrase the question. What would happen if the function returns the index of the occurrence of x in seq when x is not found in seq?

Student response: If x is not already in seq, then the correct answer would be the index immediately following the element bigger than x in seq. [Change the condition to reflect that…]

Verifier assessment:

answer_addresses_question: True

answer_has_no_mistakes: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier checks if each state variable is solved:

1. return the index of the first occurrence: True
2. return the index where x should be inserted: True
3. modify the comparison operator: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier checks if each state variable is solved:

1. return the index of the first occurrence: True
2. return the index where x should be inserted: True
3. modify the comparison operator: True

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates δ
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Student generates fixes based on any modifications suggested in the conversation.

Student bug fixes:

- Replace `<` with `<=`

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier compares ground truth list of bug fixes the to **Student**'s bug fixes.

Ground truth BF:

- Replace `<` with `<=`

Student BF:

- Replace `<` with `<=`

Methodology

1. δ←V gens. state
2. I gens. q
3. While bugs remain
    a. α ← **S** responds
    b. V validates α
    c. If α is wrong:
        i. I gens. sibling q
    d. Else:
        i. V updates **δ**
        ii. If any updates:
            1. I asks **S** to gen. BF
        iii. Else:
            1. I asks **S** a child q

Verifier compares ground truth list of bug fixes the to **Student**'s bug fixes.

Ground truth BF:

- Replace `<` with `<=`

Student BF:

- Replace `<` with `<=`

Termination!

Evaluation

Datasets:

- Socratic Debugging Benchmark (SDB)
  - Contains data with (1) problem statements, (2) buggy and (3) correct code, (4) corresponding bug fixes and (5) bug descriptions
  - 149 problems
- MULTI-DEBUG
  - Based on Leetcode problems (16 easy, 29 medium, 5 hard = 50)
  - Injected 1, 2, and 3 bugs to make a total of 150
  - Same data as SDB

Evaluation

Performed human evaluation for each conversation. Measured:

- **Average number of turns**
- **Success rate**: % of bugs solved
- **Relevant**: is the question relevant to the bugs?
- **Indirect**: does the question reveal the bug/solution?
- **Logic**: does the question flow logically from the previous question/previous **Student** response?

Evaluation

Table 1: Results on the Socratic Debugging Benchmark Dataset (Single Bug). **Bolded** and † values denote the top 2 methods respectively.

		Syntactical (42 samples)				Conceptual (107 samples)			
Methods	Avg. Turns	Success	Relevant	Indirect	Logic	Success	Relevant	Indirect	Logic
Vanilla	3.23	**80.95**	83.72†	76.19	78.70†	76.64†	87.35†	80.32†	78.79†
Bridge	6.00	78.57†	76.50	82.24†	41.72	62.14	78.12	79.86	34.38
TreeInstruct	5.41	77.27	**92.01**	**96.48**	**88.95**	**80.26**	**95.63**	**89.10**	**94.63**

Evaluation

Table 2: Results on the **MULTI-DEBUG** dataset. In total, 1-bug has 29 syntactical and 21 conceptual bugs, 2-bug has 50 syntactical and 50 conceptual bugs, and 3-bug has 78 syntactical and 72 conceptual bugs. **Bolded** and † values denote the top 2 methods respectively.

			Syntactical				Conceptual			
Bugs	Methods	Avg. Turns	Success	Relevant	Indirect	Logic	Success	Relevant	Indirect	Logic
1	Vanilla	2.36	71.43	92.16	55.12	84.15	**78.57**	94.58	59.17	84.17
	BRIDGE	16.60	50.00	93.93	**98.04**	24.23	68.00	97.27	**96.67**	35.38
	TreeInstruct	7.24	**76.19**	93.98†	94.08	85.28†	71.43	97.57†	93.02†	86.02†
	TreeInstruct	3.94	75.00†	**100.00**	95.59†	**96.63**	76.92†	**100.00**	88.01	**94.76**
2	Vanilla	8.32	53.26	83.45	74.41	60.82	62.50	86.96	74.13	59.90
	BRIDGE	15.28	34.88	89.47	89.33	52.40	42.71	89.67	88.06	46.64
	TreeInstruct	9.04	66.67†	93.00†	92.17†	84.59†	72.62†	94.15†	92.58†	81.46†
	TreeInstruct	6.14	**69.32**	**97.96**	**98.47**	**90.14**	**73.91**	**99.58**	**98.47**	**94.45**
3	Vanilla	17.48	44.00†	69.88	64.31	52.38	67.00	84.68	84.68	41.51
	BRIDGE	8.44	19.00	87.78	83.95	64.95	43.00	90.09	85.78	44.65
	TreeInstruct	10.46	43.00	95.68†	88.88†	80.94†	72.00†	96.76†	**97.95**	83.28†
	TreeInstruct	10.46	**73.00**	**100.00**	**99.27**	**95.57**	**92.00**	**98.40**	95.89†	**93.63**

Takeaways

1-bug and 3-bug have advantages that 2-bug doesn't

- 1-bug is an easier setting, as there is only 1 bug to tackle
- 3-bug benefits from inter-bug dependency

Still, TreeInstruct tackles challenging problems effectively

- Much less variation in TreeInstruct scores than baselines

Instruct, Not Assist: LLM-based Multi-Turn Planning and Hierarchical Questioning for Socratic Code Debugging

Priyanka Kargupta*, Ishika Agarwal*,
Dilek Hakkani-Tur, Jiawei Han

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CONVAI