Multimodal Programming in Computer Science with Interactive Assistance Powered by a Large Language Model

Rajan Das Gupta[1], Md. Tanzib Hosain[1], M. F. Mridha[1], and Salah Uddin Ahmed[2]

[1]American International University-Bangladesh
[2]University of South-Eastern Norway

Introduction

Problem: Large introductory computer science courses struggle to provide students with timely and effective homework assistance.
Opportunity: Large Language Models (LLMs) like DeepSeek R1 can offer instant, interactive support.
Challenge: How can we use LLMs to facilitate learning without simply giving students the answers?
Our Solution: An interactive homework help system integrated into the students' programming environment that provides pedagogical guidance rather than direct solutions.

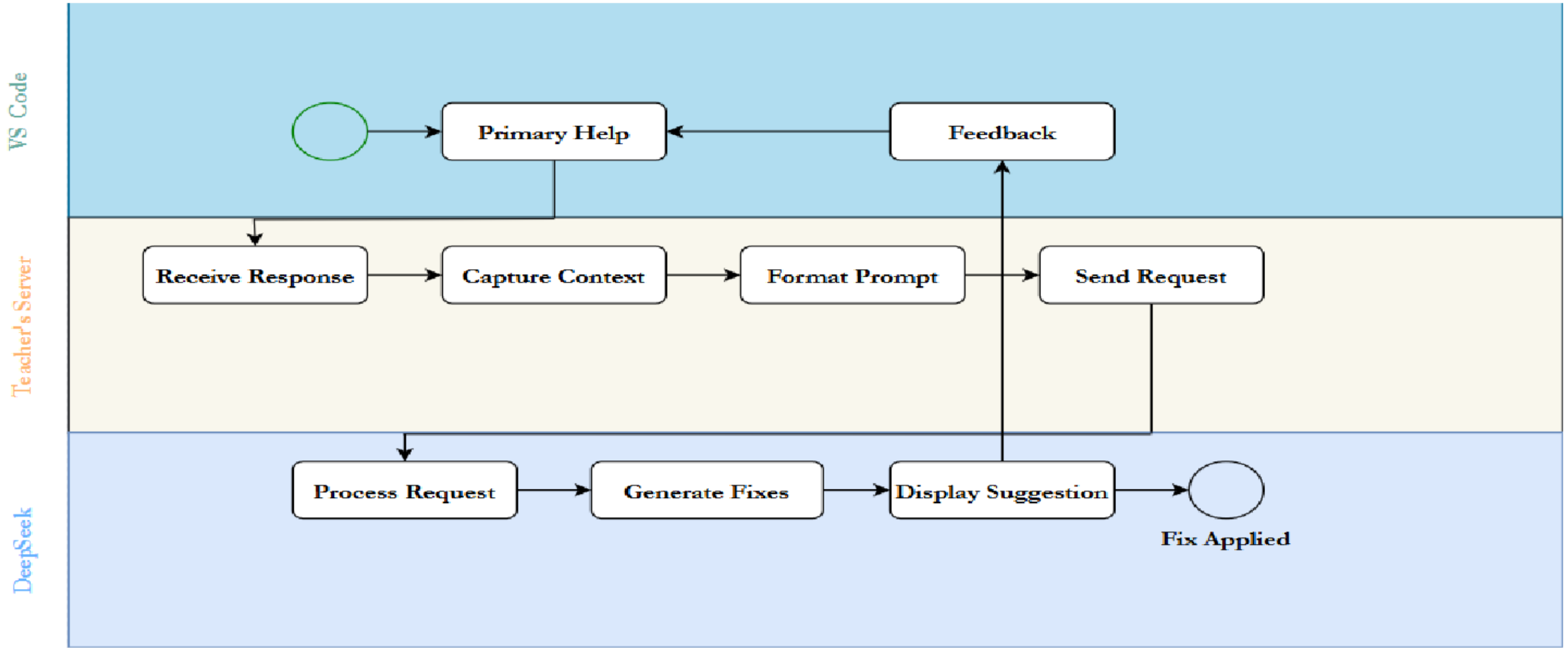
System Overview

Our system provides help through two main channels, both powered by the same backend assistant.

VS Code Extension: A "Primary Help" button directly within the code editor.

Command-Line Auto evaluator: A "Feedback" option integrated into the tool students use to check their code.

System Overview



Methodology

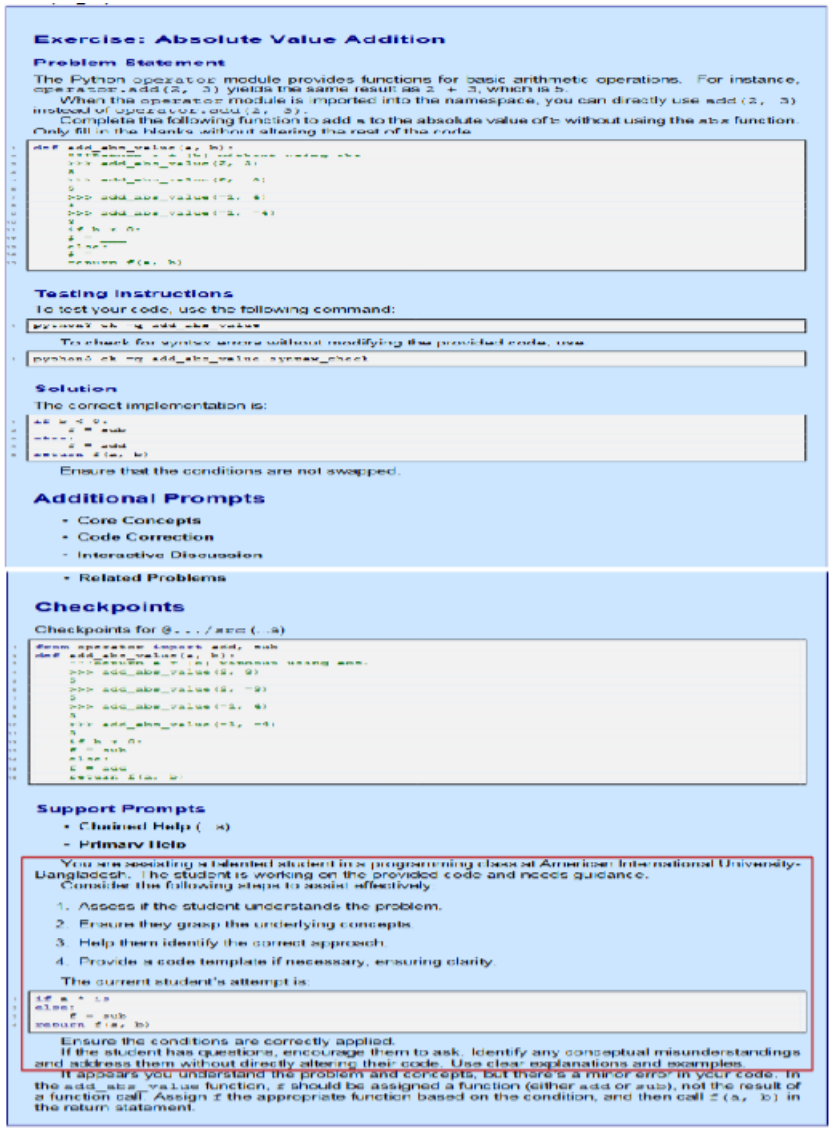
We decided against fine-tuning a model and instead focused on **prompt engineering** to guide the DeepSeek R1 model.

Goal: To provide feedback that mirrors how a human tutor would help.

Our Prompt's Strategy:

- Assess the student's understanding of the problem.
- Identify conceptual gaps.
- Help the student formulate a plan.
- Provide hints or code templates as a last resort, without giving away the solution.

Methodology



Exercise: Absolute Value Addition

Problem Statement

The Python `operator` module provides functions for basic arithmetic operations. For instance, `operator.add(2, 3)` yields the same result as `2 + 3`, which is 5.

When the `operator` module is imported into the namespace, you can directly use `add(2, 3)` instead of `operator.add(2, 3)`.

Complete the following function to add `a` to the absolute value of `b` without using the `abs` function. Only fill in the blanks without altering the rest of the code.

```
def add_abs_value(a, b):
    """Return a + abs(b) without using abs.
    >>> add_abs_value(2, 3)
    5
    >>> add_abs_value(2, -3)
    5
    >>> add_abs_value(-2, 3)
    -5
    >>> add_abs_value(-2, -3)
    -5
    """
    if b < 0:
        f = ____
    else:
        f = ____
    return f(a, b)
```

Testing Instructions

To test your code, use the following command:

```
pytest ch_9_add_abs_value
```

To check for syntax errors without modifying the provided code, use:

```
python3 ch_9_add_abs_value_syntax_check
```

Solution

The correct implementation is:

```
if b < 0:
    f = sub
else:
    f = add
return f(a, b)
```

Ensure that the conditions are not swapped.

Additional Prompts

- **Core Concepts**
- **Code Correction**
- **Interactive Discussion**
- **Related Problems**

Checkpoints

Checkpoints for `@.../add_abs_value(...3)`

```
from operator import add, sub
def add_abs_value(a, b):
    """Return a + abs(b) without using abs.
    >>> add_abs_value(2, 2)
    4
    >>> add_abs_value(2, -2)
    4
    >>> add_abs_value(-3, 4)
    -7
    >>> add_abs_value(-5, -4)
    -9
    """
    if b < 0:
        f = sub
    else:
        f = add
    return f(a, b)
```

Support Prompts

- **Chained Help (x)**
- **Primary Help**

You are assisting a talented student in a programming class at American International University-Bangladesh. The student is working on the provided code and needs guidance.

Consider the following steps to assist effectively:

1. Assess if the student understands the problem.
2. Ensure they grasp the underlying concepts.
3. Help them identify the correct approach.
4. Provide a code template if necessary, ensuring clarity.

The current student's attempt is:

```
if a < b
else:
    f = sub
return f(a, b)
```

Ensure the conditions are correctly applied.

If the student has questions, encourage them to ask. Identify any conceptual misunderstandings and address them without directly altering their code. Use clear explanations and examples.

It appears you understand the problem and concepts, but there's a minor error in your code. In the `add_abs_value` function, `f` should be assigned a function (either `add` or `sub`), not the result of a function call. Assign `f` the appropriate function based on the condition, and then call `f(a, b)` in the return statement.

The "Secret Sauce": Our Prompt Design

We developed a detailed prompt that instructs the LLM on how to behave like an effective tutor.

Key Instructions to the LLM:

"Your role is to guide students in learning programming concepts effectively."

"Avoid providing direct answers or complete code."

"If the code is incomplete or incorrect, consider the following steps..." (e.g., "Does the student lack any fundamental understanding?", "Is the existing code moving in the right direction?")

"Use a Socratic approach to encourage critical thinking."

"Keep your response brief - one or two sentences at most."

Challenges & Solutions

Challenge: The LLM often gave direct answers, especially in longer conversations.
Solution: We restricted the interaction to a "one-shot" request to limit this behavior.
Challenge: The LLM would sometimes "correct" perfectly good code (a false negative).
Solution: We experimented with having the LLM generate a correct solution first for comparison, but this added too much latency. The issue remains a key area for improvement.
Challenge: The LLM would suggest solutions that were technically correct but violated the specific constraints of the assignment.
Solution: We added assignment-specific context to the prompt.

Deployment & Findings

Rollout: Deployed in two phases to over 1100 students in an introductory programming course.

Positive Findings:

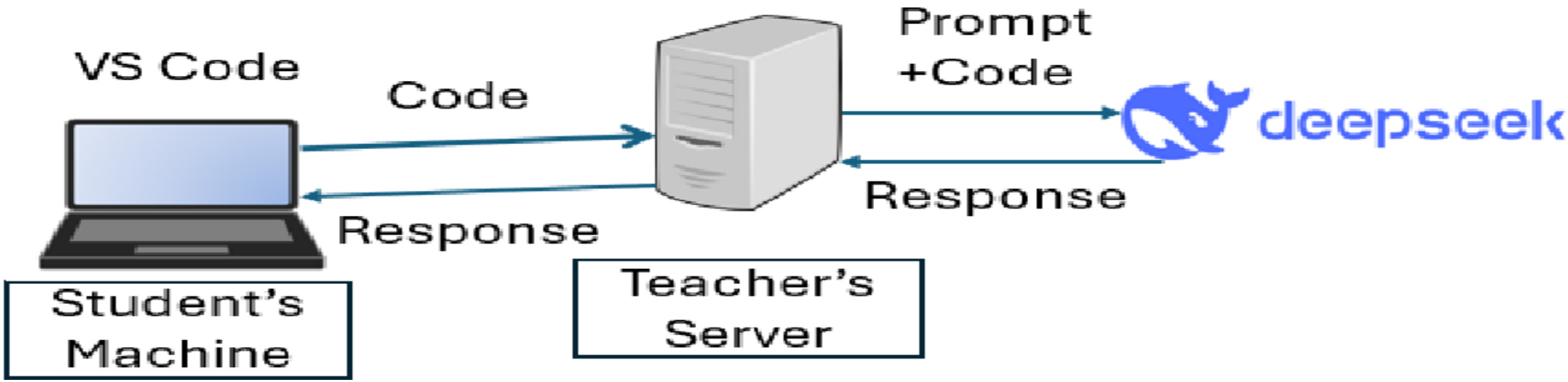
- Students used the bot frequently, especially near deadlines, suggesting they found it helpful.
- The bot was effective at helping students resolve syntax errors and function misuse.
- A noticeable decrease in the number of support inquiries during office hours was observed.

Negative Findings:

False Negatives: The bot would sometimes claim correct code was wrong.

False Positives: The bot would approve incorrect code or provide misleading suggestions, leading to student frustration.

Deployment & Findings



Limitations: The "Failure" Patterns

The most significant issue was the bot's unreliability, which manifested in two main ways:

False Negatives:

- The bot tells a student their correct code is incorrect.
- This is less harmful in the auto evaluator mode, as passing all test cases overrides the bot's feedback.

False Positives (More Damaging):

- The bot approves incorrect code, leading students down a rabbit hole of debugging.
- The bot suggests a fix that violates assignment constraints, causing conflicts with the autoevaluator. This was the most frustrating experience for students.

Conclusions & Future Directions

Conclusion: An LLM-powered assistant can be a valuable tool in programming education, but its effectiveness is highly dependent on careful prompt engineering and managing its limitations.

Future Work:

Improve Accuracy: Continue to refine prompts and potentially explore fine-tuning to reduce errors.

Enhance Interaction: Allow students to ask for clarification (e.g., "Explain that differently") and rate the helpfulness of the advice.

Deeper Analysis:

- How does bot usage correlate with student performance?
- When and why do students seek help from the bot?
- How do students feel about interacting with a bot versus a human TA?

Thank You

Any Questions?