LLM Should Ask Clarifying Questions to Increase Confidence in Generated Code
—— On the Communication Skills of LLM

Jie JW Wu
George Washington University
Incoming Postdoc at University of British Columbia
@jw_ _wu
https://jie-jw-wu.github.io/

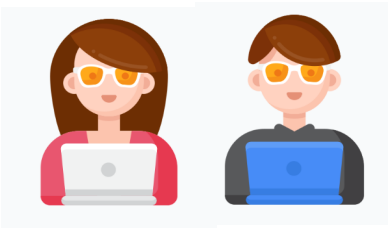
Background

ChatGPT

GitHub Copilot

Amazon CodeWhisperer

Gap



Software engineers

Job:
Programmer's assistant for generating code

Job:
writing code,
communications,
requirements,
design
…

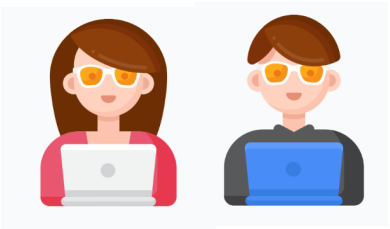
Motivation

Issues in programming with LLM:

- Intent specification, problem decomposition and computational thinking (Sarkar et al. 2022)
- Code quality and overconfidence (Johnson et al. 2023, Liu et al. 2023)
- Usability issues (Liang et al. 2024)

Motivation

- Apply the communication lens to inspect the gap
- Question: Does asking clarifying questions increase confidence in code generation?
- Compare communication skill of software engineer vs LLM:

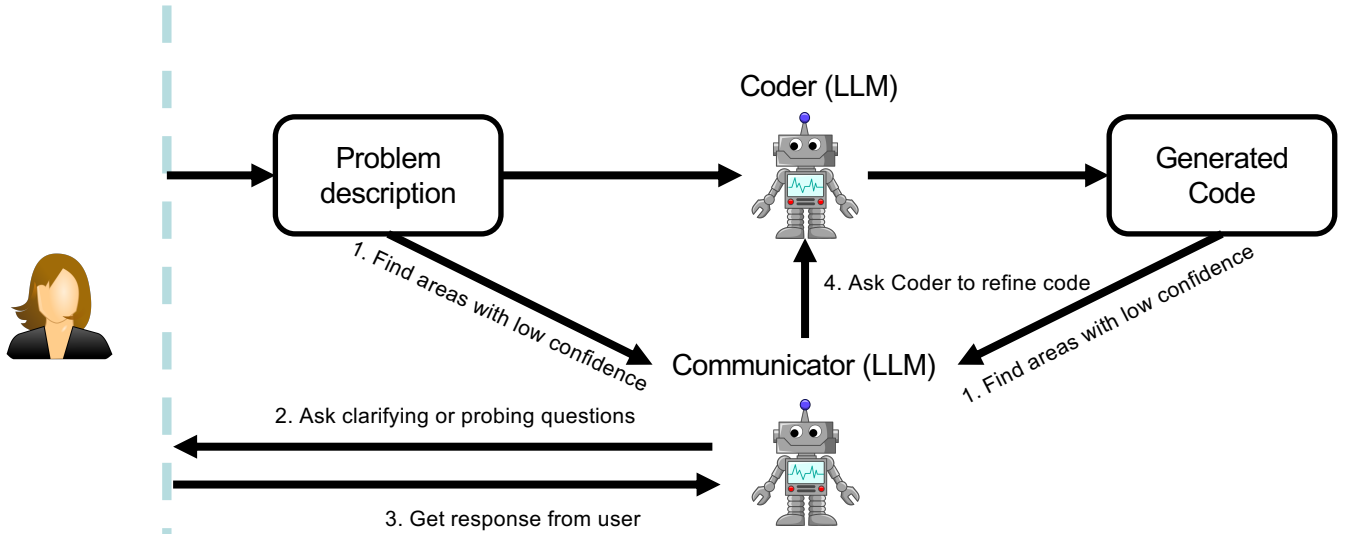


Effective communication is a critical skill to accomplish software engineering tasks reliably with high quality

The degree of communication skills is rarely evaluated in code generation

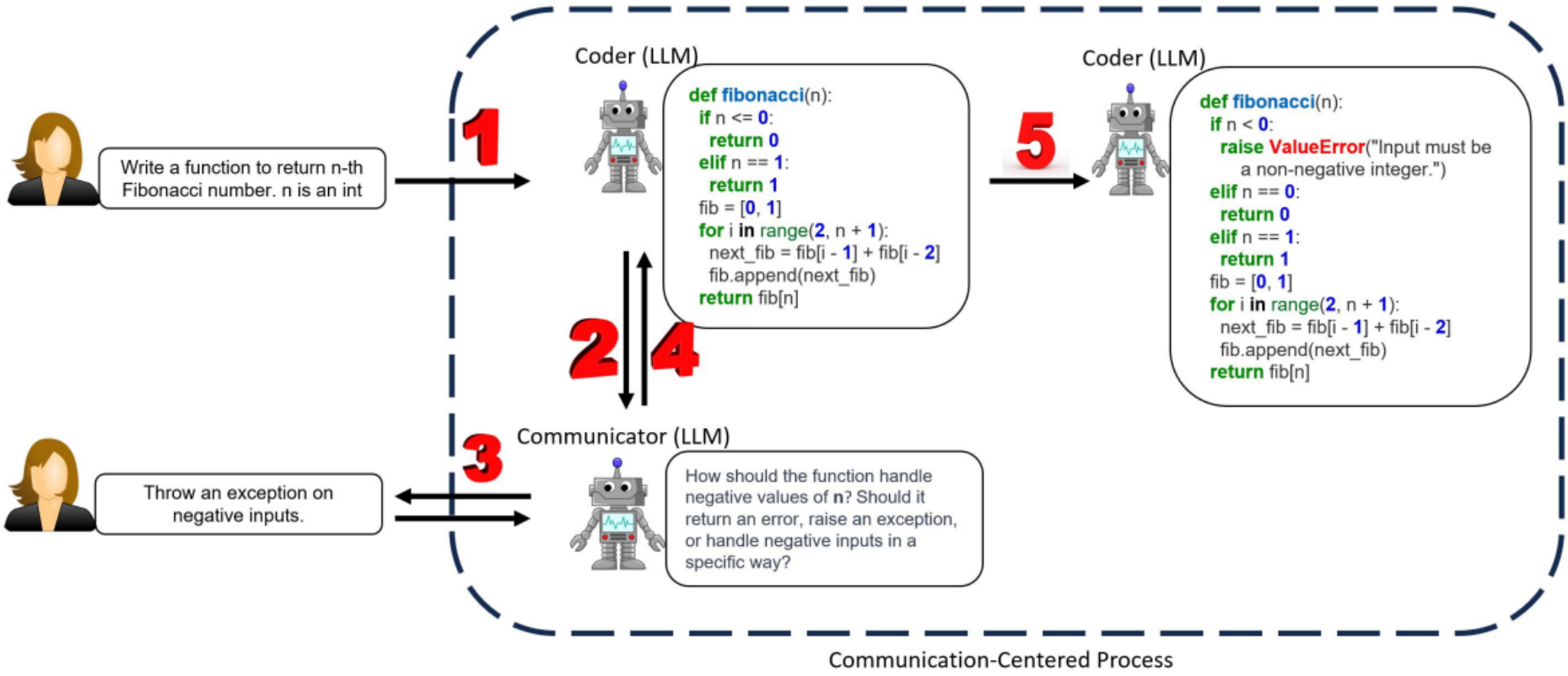
Approach: Communication-Centered Process

- Study the potential of LLMs from the dimension of effective communication skills.
- Explore a process with 2 LLMs (Coder + Communicator) to ask more clarifying questions to refine the generated code.



Communication-Centered Process - Example Usage

- Key part of the process: how the communicator interacts with the user and the coder.



Communication-Centered Process - Example Usage

- Prompt and Generated questions of Communicator LLM

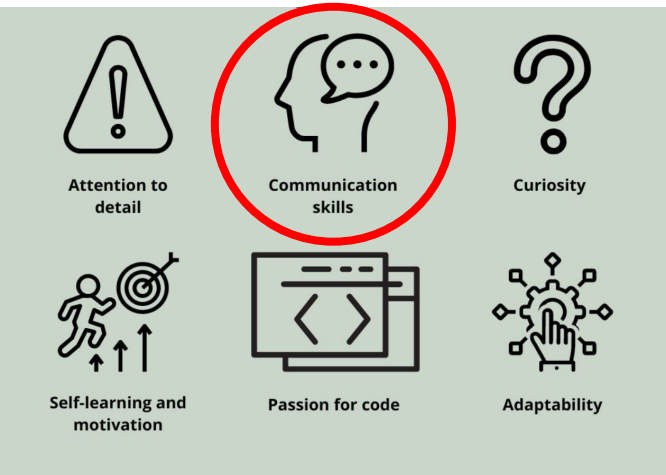
```
You are an expert in software engineering. You will be given the problem description and current code of a coding task. You will generate a list of clarifying questions that may result in refining the code.

### Problem Description
"write a function to return n-th Fibonacci number. n is an int "

### Generated Code From Previous Iteration
def fibonacci(n):
 if n <= 0:
  return 0
 elif n == 1:
  return 1
 fib = [0, 1]
 for i in range(2, n + 1):
  next_fib = fib[i - 1] + fib[i - 2]
  fib.append(next_fib)
 return fib[n]
```

Topic	Clarifying Questions Generated by the Communicator LLM
Input Validation	Should input validation be part of the function?
	If so, what should be done if n is not an integer?
Error Handling	How should errors be handled if n is not a valid integer or if it's negative?
	Should the function raise exceptions, return error codes, or follow a different error-handling strategy?
Performance Requirements	Are there any performance requirements or constraints for this function?
	Is there a need for optimizing the function for efficiency, especially for large values of n?
Function Signature	Are there any specific naming conventions or requirements for the function name and parameters?
	Should the function take any additional parameters, options, or configuration settings?
Output Type	Is there a preferred return type for the function, such as an integer, a string, or another data type?
Default Values	Should the function provide default values for n if it's not provided by the caller?
Documentation	What documentation standards should be followed for this function, such as docstrings, comments, or external documentation?
Testing	Are there specific test cases or test scenarios that should be considered when testing the function?
	Should the function be tested for edge cases, such as very large values of n?
Language and Environment	Is there a specific programming language or environment in which this function should be implemented?
	Are there any platform or compatibility requirements?
Algorithm Choice	Is there a preference for using a specific algorithm for calculating Fibonacci numbers, such as an iterative or recursive approach?

⋮

Benchmarking the communication skills of Code LLM

Question: How to effectively evaluate the degree of Code LLM's communication skills?



Top-Notch Software Engineer:

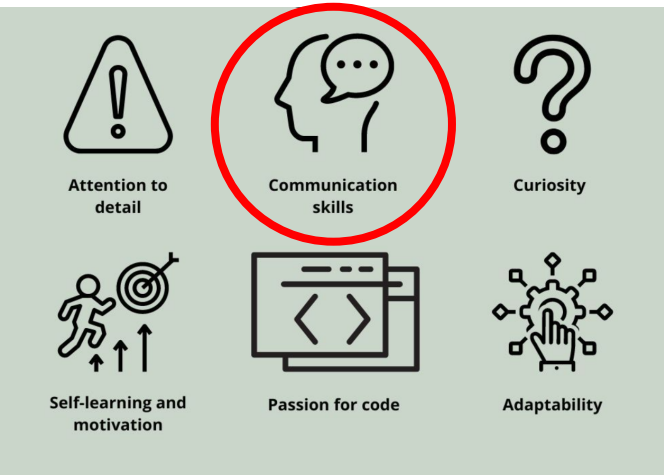
- Deliberately remove information from problem description in job interviews
- Check if candidate ask clarifying questions

LLM:

- ***Randomly removing*** certain parts of the problem descriptions of the existing dataset
- Check if LLM asks clarifying questions

Benchmarking the communication skills of Code LLM

Question: How to effectively evaluate the degree of Code LLM's communication skills?



New benchmark: HumanEval-C

Top-Notch Software Engineer:

- Deliberately remove information from problem description in job interviews
- Check if candidate ask clarifying questions

LLM:

- ***Randomly removing*** certain parts of the problem descriptions of the existing dataset
- Check if LLM asks clarifying questions

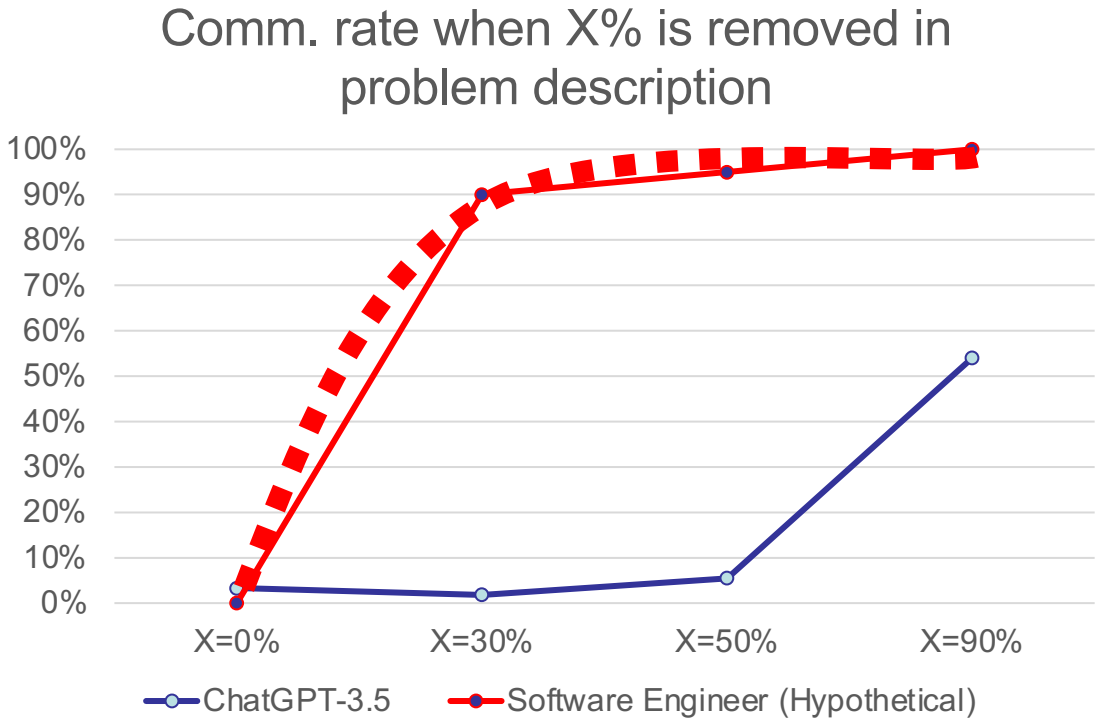
HumanEval-C: Benchmarking the communication skills of Code LLM

Dataset: Randomly remove **X% (30%,50%,90%)** of consecutive words from the original problem description in HumanEval dataset.

Evaluation: 1) test pass rate, 2) communication rate (comm. rate) = $\frac{\#\,\text{👍}}{\#Problems}$

% Removed in Description	Example Problem Description
0%	def encode_cyclic(s: str): returns encoded string by cycling groups of three characters. # split string to groups. Each of length 3. groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)] # cycle elements in each group. Unless group has fewer elements than 3. groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups] return "".join(groups) def decode_cyclic(s: str): takes as input string encoded with encode_cyclic function. Returns decoded string.
30%	def encode_cyclic(s: str): returns encoded string by cycling groups of three characters. split string to in each group. Unless group has fewer elements than 3. groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups] return "".join(groups) def decode_cyclic(s: str): takes as input string encoded with encode_cyclic function. Returns decoded string.
50%	def encode _cyclic(s: str): returns encoded string by cycling groups of three characters. split string to groups. Each of length 3. groups = [s[(3 * i):min((3 * i + takes as input string encoded with encode_cyclic function. Returns decoded string.
90%	def encode_cyclic(s: str): encode_cyclic function. Returns decoded string.

HumanEval-C: Findings

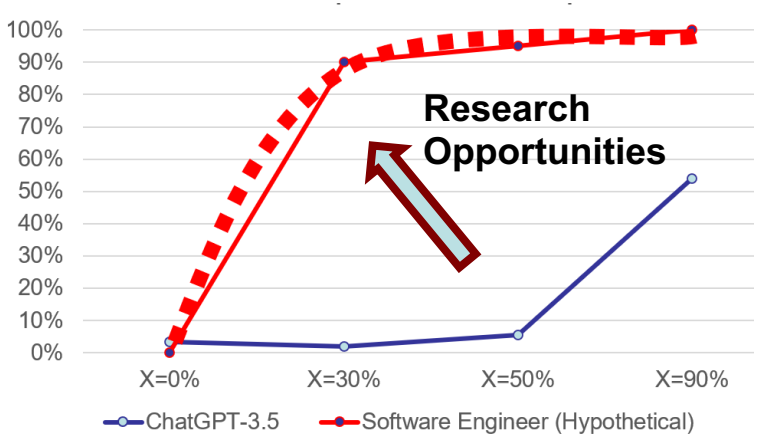


- ChatGPT-3.5 is currently very **weak** at asking clarifying questions when this ability is necessary for trustworthy code generation.
- **Lower temperature** or **using GPT-4** does not help much to increase the chance of LLM to ask questions, but the new **Communication-Centered Process** is effective. (Comm. Rate 5%→64% when X=50%)

Takeaway

- **LLM *should* ask clarifying questions to increase confidence in generated code**

"Asking a good question can be valuable in and of itself, irrespective of the answer. It communicates your respect for the other person."
- Iowa Peace Institute Message



New benchmark: HumanEval-C and preprint will be available @jw__wu

- ChatGPT is very **weak** at asking clarifying questions when this ability is necessary for trustworthy code generation
- Opportunities: improve **evaluation** and **model** of communication skills in LLM