Bugs in Large Language Models Generated Code

Our research aimed at better understanding the kind of mistakes and/or bugs Large Language Models (LLMs), such as Codex, make when generating code. We focused mainly on Python generated code, yet we believe similar issues would happen in other languages.

The purpose of this survey is to get feedback from practitioners on the identified bugs/mistakes categories. Please feel free to participate if you find yourself eligible. Moreover, you could kindly share this survey with students/colleagues/friends who you consider eligible to participate. The results of this survey will be publicly accessible through arXiv.org in anonymized form. We greatly appreciate your help in doing so. The survey should take no longer than 10 minutes.

At no point in the survey will we ask you for your name, and we will not be logging your IP address to allow anonymity. If you would like to know more about this study, feel free to contact us with your questions.

Florian Tambon (florian-2.tambon@polymtl.ca), Arghavan Moradidakhel (arghavan.moradi-dakhel@polymtl.ca)
Amin Nikanjam (amin.nikanjam@polymtl.ca), Foutse Khomh (foutse.khomh@polymtl.ca)
SWAT Lab., Polytechnique Montréal,
Montréal, Canada,
http://swat.polymtl.ca/

* Indicates required question

General LLM Usage

This part is about general questions on the LLM usage and the potential bugs found in LLM generated code.

1. What is your current job title? (Developer, Researcher, ... students may indicate degree: PhD, Master) *

____

2. Which LLMs do you mostly use? Select as many as you use. You can enter one manually if it's not on the list. *

Check all that apply.

- ☐ Codex (Copilot)
- ☐ ChatGPT
- ☐ LLama
- ☐ CodeGen
- ☐ Other: _______________________________

3. Aside from Python, which programming languages did you generate code for using LLMs? *

Check all that apply.

- ☐ C
- ☐ C++
- ☐ Java
- ☐ JavaScript
- ☐ C#
- ☐ Go
- ☐ Other: _______________________________

4. How often do you encounter mistakes in LLMs generated code you use? *

Those mistakes can be anything, from code that won't compile (variable not defined, function that does not exist) to more silly choice from the LLMs (multiples If condition checking the same thing or casting back and forth a variable for no reason).

Mark only one oval.

- Never
- Occasionally
- Sometimes
- Often
- Always

5. When you encounter mistakes in LLMs generated code, how complex is fixing those issues? *

Mark only one oval.

- Trivial
- Easy
- Medium
- Hard
- Very Hard
- Never Encountered a mistake

6. To fix those issues, how do you proceed? *

Mark only one oval.

- Manually modifying the code
- Asking the LLMs with a new prompt to correct the mistakes until it does so
- Combination of both
- Other: ______________________

Categories of bugs

We extracted LLM generated code using an existing benchmark (CoderEval: https://arxiv.org/pdf/2302.00288.pdf). We analyzed buggy code snippets and we came up with a taxonomy based on what type of mistakes occur. In this section, you will be asked to assess:

- If you ever encountered such a mistake

- How would you rate this type of bugs according to three factors:

-> Diagnosing: How hard would it be for you to diagnose such an error if it happens to be inside a LLM generated code? Could you distinguish it easily or would it require actually running the code, using a debugger... rendering the diagnostic hard?

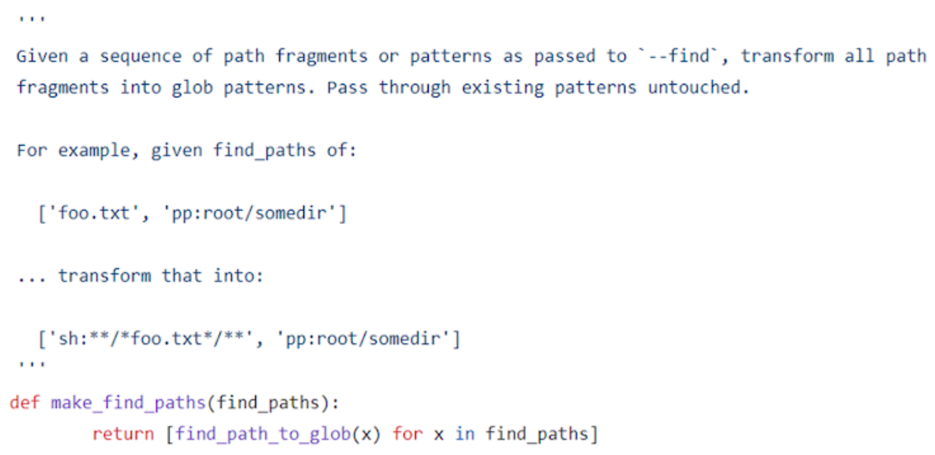
-> Complexity: Is the type of error rather trivial (i.e. not something somebody with decent programming knowledge would) or, on the contrary, does it denote some degree of complexity?

-> Fixing: If you had to fix such a mistake, would it be hard? Would it just require adjusting some part of the code (i.e. easy), or would it require an extensive refactoring (i.e. hard)?

Every time, you will be given the name of the category and a short description. We also provide a code snippet example. Said snippet is composed of the docstring and prototype of the function for which a LLM adds to generate a code for. You will also see an example of generated code by an LLM which made the particular mistake of the category we described.

We also provide a comment box (optional) at the end of each category would you feel like adding something.

7. **Hallucinated objects:** *
LLMs hallucination such as using a function or built-in type that does not exist and was not defined previously but that the LLMs consider present within the context.
In the example below, the LLM (besides not doing what the docstring asks), uses a function named "find_path_to_glob" which is not defined in the context.



```
'''
Given a sequence of path fragments or patterns as passed to `--find`, transform all path
fragments into glob patterns. Pass through existing patterns untouched.

For example, given find_paths of:

  ['foo.txt', 'pp:root/somedir']

... transform that into:

  ['sh:**/*foo.txt*/**', 'pp:root/somedir']
'''
def make_find_paths(find_paths):
        return [find_path_to_glob(x) for x in find_paths]
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	○	○	○	○	○	Always

8. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	⬭	⬭	⬭	⬭	⬭
Complexity	⬭	⬭	⬭	⬭	⬭
Fixing	⬭	⬭	⬭	⬭	⬭

9. Comments?

10. **Silly Mistakes** *

Mistakes such as redundant conditions or casting for no reason. The change won't introduce a bug on its own but it can easily lead to it.

In the example below, the LLM repeats the same functionality inside both branches of the 'if...else...' structure. This does not cause a bug, but has little usefulness in the code and can even lead easily to a mistake if the user tries to expand the code.

```
if not logfile.exists():
        logging.basicConfig(
                filename=str(logf),
                format='%(asctime)s %(levelname)s %(name)s %(message)s',
                datefmt='%Y-%m-%d %H:%M:%S',
                level=logging.DEBUG,
        )
else:
        logging.basicConfig(
                filename=str(logf),
                format='%(asctime)s %(levelname)s %(name)s %(message)s',
                datefmt='%Y-%m-%d %H:%M:%S',
                level=logging.DEBUG,
        )
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

11. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

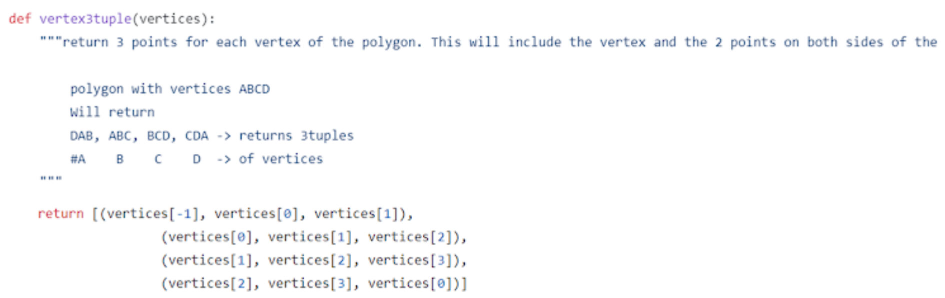
	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

12. Comments?

---

13. **Prompt-biased** *

When the LLM relies too much on an example or words given in the prompt when implementing the function which hampers the generated code generalization.
In the example below, the LLM relies too much on the example given in the docstring. By doing so, it generates a code that only works on a polygon with 4 vertices (just like in the example). Yet, the function should generalize to any number of vertices.



```
def vertex3tuple(vertices):
    """return 3 points for each vertex of the polygon. This will include the vertex and the 2 points on both sides of the

        polygon with vertices ABCD
        Will return
        DAB, ABC, BCD, CDA -> returns 3tuples
        #A    B    C    D  -> of vertices
    """
    return [(vertices[-1], vertices[0], vertices[1]),
                (vertices[0], vertices[1], vertices[2]),
                (vertices[1], vertices[2], vertices[3]),
                (vertices[2], vertices[3], vertices[0])]
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

14. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

15. Comments?

____

____

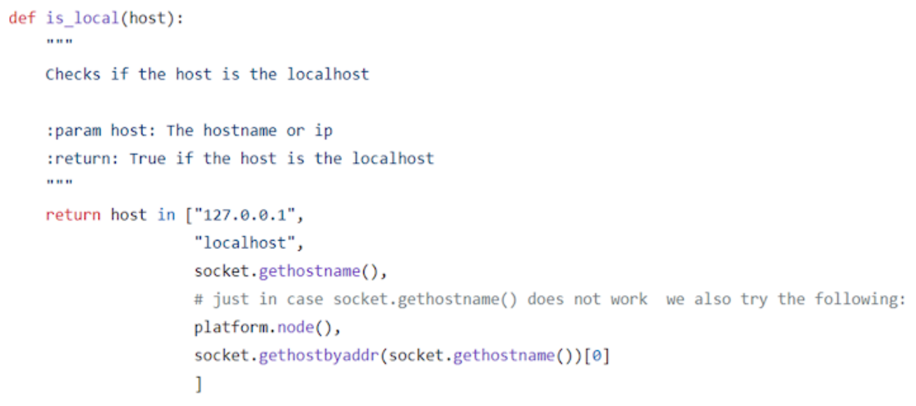
____

____

____

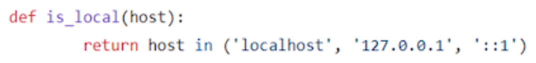
16. **Missing Corner Case** *

When the code is very similar to what was expected, yet the LLM might miss some corner case or neglect a part of the information given by the prompt.
In the example below, the expected code is given at the top and the LLM generated code is at the bottom. To generate the code, the LLM only had access to the function signature and the small docstring in between """. As one can see, the LLM code is very similar to the expected one, but it's missing some additional parts that would have been expected. Nonetheless, the code is still functional with regard to the information given.



```
def is_local(host):
    """
    Checks if the host is the localhost

    :param host: The hostname or ip
    :return: True if the host is the localhost
    """
    return host in ["127.0.0.1",
                    "localhost",
                    socket.gethostname(),
                    # just in case socket.gethostname() does not work  we also try the following:
                    platform.node(),
                    socket.gethostbyaddr(socket.gethostname())[0]
                    ]
```



```
def is_local(host):
        return host in ('localhost', '127.0.0.1', '::1')
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

17. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

18. Comments?

____

____

____

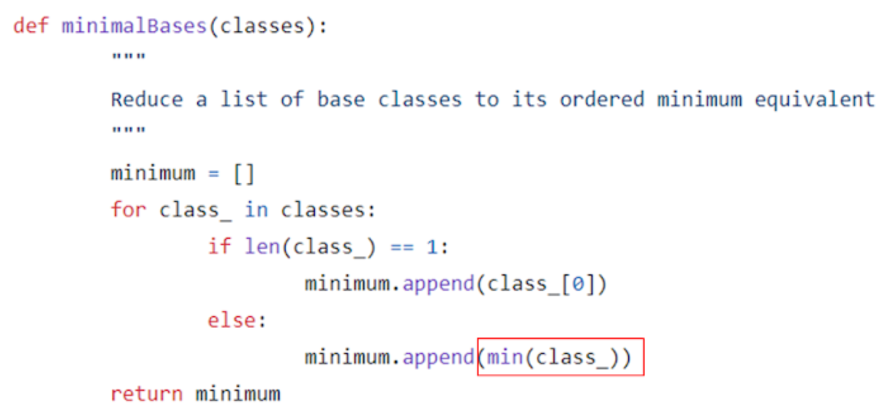
____

____

19. **Wrong Input Type** *

When the LLM uses an incorrect input type in some function.

In the example below, the LLM generated code uses the Python function 'min' on 'class_' which is a class object. The function 'min' can't handle class objects as input and so an error is raised.



```python
def minimalBases(classes):
        """
        Reduce a list of base classes to its ordered minimum equivalent
        """
        minimum = []
        for class_ in classes:
                if len(class_) == 1:
                        minimum.append(class_[0])
                else:
                        minimum.append(min(class_))
        return minimum
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	⬭	⬭	⬭	⬭	⬭	Always

20. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	⬭	⬭	⬭	⬭	⬭
Complexity	⬭	⬭	⬭	⬭	⬭
Fixing	⬭	⬭	⬭	⬭	⬭

21. Comments?

____

____

____

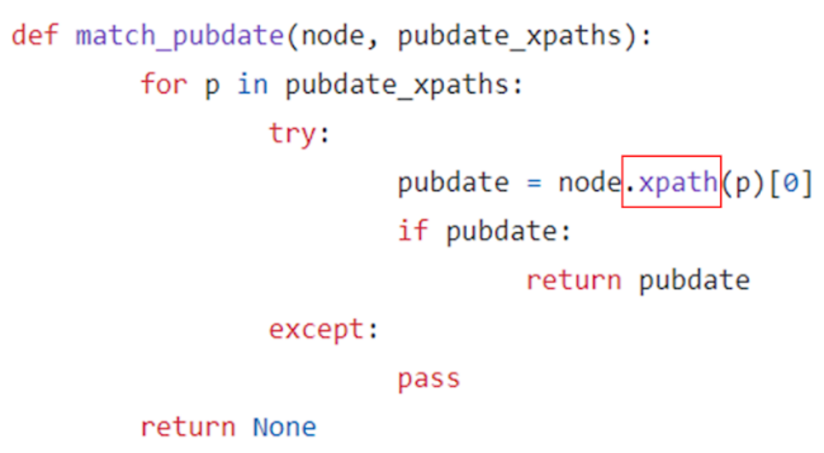
____

____

22. **Wrong Attribute** *

When the LLM uses a wrong attribute for an object or module.

In the example below, the LLM generated code for parsing an .xml file. The LLM generated a method attribute named 'xpath' which does not exist for an xml object, raising an error.



```
def match_pubdate(node, pubdate_xpaths):
        for p in pubdate_xpaths:
                try:
                        pubdate = node.xpath(p)[0]
                        if pubdate:
                                return pubdate
                except:
                        pass
        return None
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

23. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

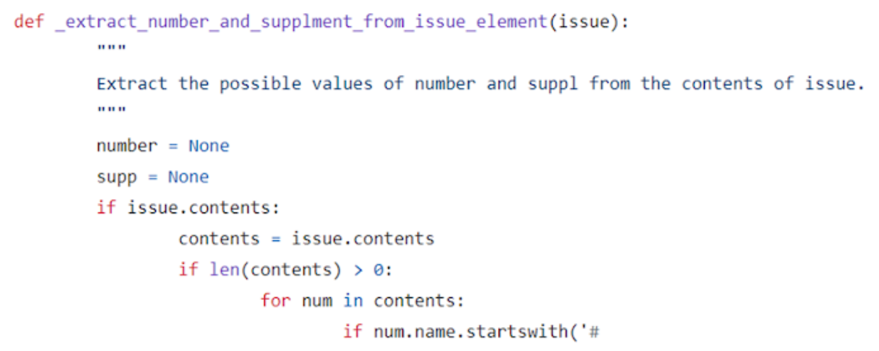
	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

24. Comments?

25. **Incomplete Generation** *

When there is no code generated, simply a "pass" or the code generation is not finished either because the LLM does not "know" what to put next or because of a token limit.

In the example below, the LLM stops the generation of the code while the function code is clearly not finished (symbolized with a the final '#').



```
def _extract_number_and_supplment_from_issue_element(issue):
        """
        Extract the possible values of number and suppl from the contents of issue.
        """
        number = None
        supp = None
        if issue.contents:
                contents = issue.contents
                if len(contents) > 0:
                        for num in contents:
                                if num.name.startswith('#
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

26. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

27. Comments?

____

____

____

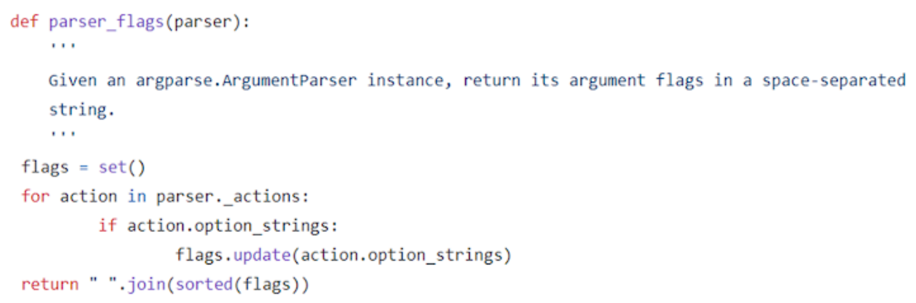
____

____

28. **Non-Prompted Consideration** *

When the LLM adds something to the code which is not asked in the prompt to the point it leads to an error directly or indirectly.

In the example below, the LLM processes the prompt currently, parsing the ArgumentParser object, until it suddenly decides to sort the result using the "sorted" function. This was not asked in the prompt and leads to an error since the different flags of ArgumentParser won't be in the initial order.



```
def parser_flags(parser):
    '''
    Given an argparse.ArgumentParser instance, return its argument flags in a space-separated
    string.
    '''
 flags = set()
 for action in parser._actions:
        if action.option_strings:
                flags.update(action.option_strings)
 return " ".join(sorted(flags))
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

29. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

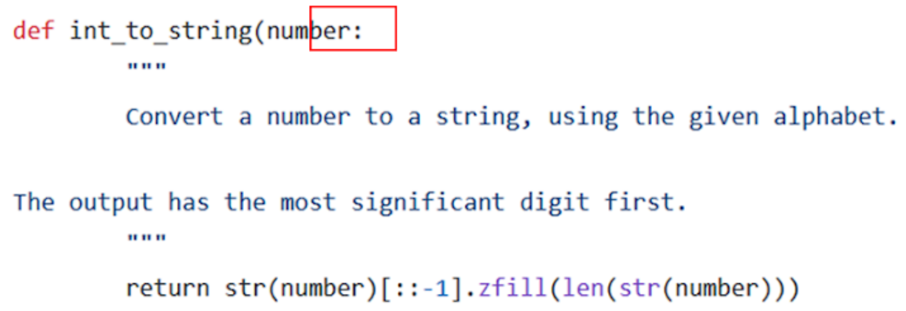
	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

30. Comments?

31. **Synthax Error** *

When the LLM code contains syntax errors such as missing parenthesis, indentation error, function of a library not properly called. ..

In the example below, the LLM generated a code where the parenthesis is missing in function signature. This will obviously raise a bug.



```
def int_to_string(number:
        """
        Convert a number to a string, using the given alphabet.

The output has the most significant digit first.
        """
        return str(number)[::-1].zfill(len(str(number)))
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev	◯	◯	◯	◯	◯	Always

32. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

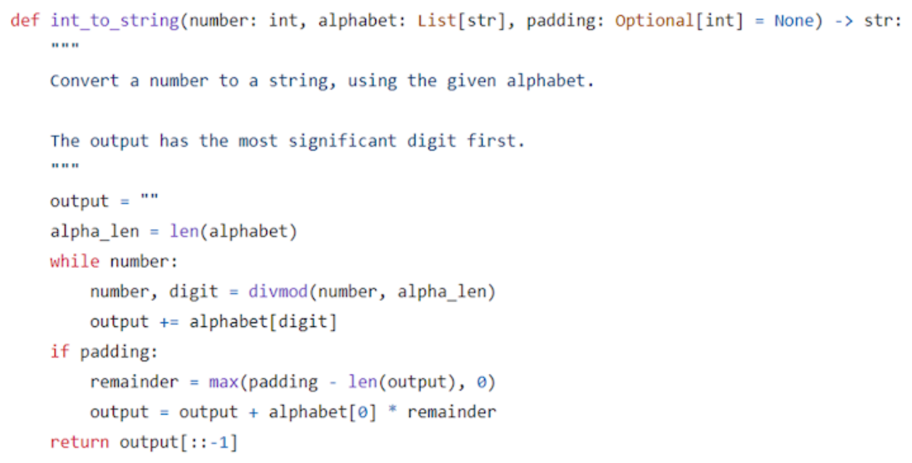
	1 (Easy/Complex)	2	3	4	5 (Hard/Complex)
Diagnosing	◯	◯	◯	◯	◯
Complexity	◯	◯	◯	◯	◯
Fixing	◯	◯	◯	◯	◯

33. Comments?

34. **Misinterpretation** *

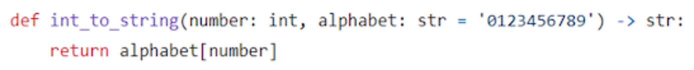
When the LLM generated code is false and is not in the direction of addressing the prompt (i.e. it's quite different from what was intended in the prompt).

In the example below, the expected code is given at the top and the LLM generated code is at the bottom. To generate the code, the LLM only had access to the function signature and the small docstring in between """. One can see that, on top of forgetting one of the function parameters and changing the default value of another one (alphabet), the generated code is way different compared to the given prompt.



```python
def int_to_string(number: int, alphabet: List[str], padding: Optional[int] = None) -> str:
    """
    Convert a number to a string, using the given alphabet.

    The output has the most significant digit first.
    """
    output = ""
    alpha_len = len(alphabet)
    while number:
        number, digit = divmod(number, alpha_len)
        output += alphabet[digit]
    if padding:
        remainder = max(padding - len(output), 0)
        output = output + alphabet[0] * remainder
    return output[::-1]
```



```python
def int_to_string(number: int, alphabet: str = '0123456789') -> str:
    return alphabet[number]
```

How often do you encounter such a type of mistake?

Mark only one oval.

	1	2	3	4	5	
Nev(	◯	◯	◯	◯	◯	Always

35. How do you assess diagnosing, severity, and fixing of such a bug? *

If you never experienced such bug, please answer based on your understanding using the previous snippet as an example for the bug type.

Mark only one oval per row.

	1 (Easy/Trivial)	2	3	4	5 (Hard/Complex)
Diagnosing	⬭	⬭	⬭	⬭	⬭
Complexity	⬭	⬭	⬭	⬭	⬭
Fixing	⬭	⬭	⬭	⬭	⬭

36. Comments?

37. Thank you very much for your answers, it is very much appreciated !

If you have any additional comments/remarks/suggestions please let us know below.