

13 | 让AI帮你写测试，体验多步提示语

2023-04-10 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

上一讲，我们一起通过 ChatGPT 做了一个小应用。不过，这个过程并不是一个“自动档”的。我们尝试一步一步输入我们的需求，给到 ChatGPT，并根据拿到的指示尝试运行代码。通过和 ChatGPT 不断地交互，我们最终完成了一个小应用。

虽然这在我们探索性地开发一些功能的时候，已经极大地提高了我们的效率。但是这个过程并不能做成一个产品。我们理想中的产品应该是“自动档”的，我们只要用自然语言输入自己的需求，对应的代码就自动写出来了。如果中间出现了错误，AI 可以自己拿到反馈来更正，而不需要我们人工去介入调试，或者复制粘贴。

先让 GPT-4 写个代码

这个思路听起来似乎有些科幻，但是随着 GPT-4 的发布，以及未来模型能力的进一步增长，这其实并不是遥不可及的。不过，这个时候你应该还只有 GPT-3.5 的 API 权限。所以这一讲，我们还是先把目标放低一点，先来**通过大语言模型，帮我们自动写单元测试代码**。整个过程仍然是一个自动档的体验，只是能够提供的能力还相对比较简单，仅限于为现有代码提供单元测试而已。


这个想法，源自 OpenAI Cookbook 提供的 [🔗 AI 写单元测试的示例](#)。但是那个例子里面的代码，已经不能使用了，因为对应的 code-davinci-002 模型已经被 OpenAI 下线了。但是例子里，分步骤分析问题，通过多个 Prompts 来完成单元测试的想法，还是非常有借鉴意义的。

我相信学完这一讲之后，随着你拿到 GPT-4 的 API 乃至未来可能会出现 GPT-5，你都完全可以用同样的方法完成更复杂的“自动写代码”的程序。

设计一个有些挑战的小题目

要写测试，我们要先有一个程序。为了避免这个题目本身就在 AI 的训练数据集里面，它直接知道答案，我们就不选用像 Leetcode 这样的题库了。我用了这样一个我觉得很有意思的小题目，也就是让 Python 根据我们输入的一个整数代表的秒数，格式化成一段自然语言描述的时间。比如，输入 1 就返回 1s，输入 61 就返回 1min1s。

需求：

 复制代码

- 1 用Python写一个函数，进行时间格式化输出，比如：
- 2 输入 输出
- 3 1 1s
- 4 61 1min1s
- 5 要求仅需要格式化到小时(?h?min?s)，即可

题目有了，我们还需要一个正确的解题程序。我们今天的重点不是怎么用 AI 刷题，所以我们不如直接让 ChatGPT 帮我们把程序写好。



用Python写一个函数，进行时间格式化输出，比如：



输入 输出

1 1s

61 1min1s

要求仅需要格式化到 小时(?h?min?s)，即可



可以使用以下Python代码实现时间格式化输出：



python

Copy code

```
def format_time(seconds):
    minutes, seconds = divmod(seconds, 60)
    hours, minutes = divmod(minutes, 60)

    if hours > 0:
        return f"{hours}h{minutes}min{seconds}s"
    elif minutes > 0:
        return f"{minutes}min{seconds}s"
    else:
        return f"{seconds}s"
```

这个函数使用 `divmod` 函数将输入的秒数转换为小时、分钟和秒钟，并根据不同的时间段进行格式化输出。如果时间大于1小时，则输出为小时、分钟和秒钟；如果时间在1分钟到1小时之间，则输出为分钟和秒钟；如果时间小于1分钟，则仅输出为秒钟。

这是对应的 Notebook 里面的代码。

复制代码

```
1 def format_time(seconds):
```

```
2     minutes, seconds = divmod(seconds, 60)
3     hours, minutes = divmod(minutes, 60)
4     if hours > 0:
5         return f"{hours}h{minutes}min{seconds}s"
6     elif minutes > 0:
7         return f"{minutes}min{seconds}s"
8     else:
9         return f"{seconds}s"
```

可以看到，AI 快速给出了一个小程序，看上去没啥问题，能够完成我们想要的基本功能。

既然 ChatGPT 可以写代码，我们自然也可以让它帮我们写单元测试也写好。



可以使用以下Python代码实现时间格式化输出：

python

Copy code

```
def format_time(seconds):
    minutes, seconds = divmod(seconds, 60)
    hours, minutes = divmod(minutes, 60)

    if hours > 0:
        return f"{hours}h{minutes}min{seconds}s"
    elif minutes > 0:
        return f"{minutes}min{seconds}s"
    else:
        return f"{seconds}s"
```

请为这个程序用pytest写一个单元测试

Save & Submit

Cancel



好的，以下是使用 `pytest` 为上述代码编写的单元测试：




python

Copy code

```
import pytest
from format_time import format_time

def test_format_time():
    assert format_time(1) == "1s"
    assert format_time(59) == "59s"
    assert format_time(60) == "1min0s"
    assert format_time(61) == "1min1s"
    assert format_time(3600) == "1h0min0s"
    assert format_time(3661) == "1h1min1s"
```

这个测试文件导入了 `pytest` 和 `format_time` 函数，然后定义了一个 `test_format_time` 函数来测试 `format_time` 函数


 复制代码

```
1 conda install pytest
```

误信息。

 Regenerate response

这是对应在 Jupyter Notebook 上面写的代码。

 复制代码

```
1 import pytest
2
3 def test_format_time():
4     assert format_time(1) == "1s"
5     assert format_time(59) == "59s"
6     assert format_time(60) == "1min0s"
7     assert format_time(61) == "1min1s"
8     assert format_time(3600) == "1h0min0s"
9     assert format_time(3661) == "1h1min1s"
```

乍一看，我们的单元测试已经写完了，那这一讲就结束了吗？当然不是了。如果你是一个比较有经验的程序员，你就会发现这个单元测试其实还是有好几个问题的。

1. 这个测试没有考虑负数。如果我们输入的是负数会怎么样？
2. 没有考虑非整数类型的输入，如果我们输入浮点数 1.0 会怎么样？字符串 “abc” 会怎么样？nil 这样的空值会怎么样？
3. 即使是整数，我们还没有考虑过，超过 24 小时的话，格式化后的结果是怎么样的。

分解步骤撰写 Prompts

所以，很多事情不是我们直接把问题一塞，给到 ChatGPT 就能解决的。**我们需要反过来自己思考一下，如果我们自己来为一段代码写单元测试，我们会怎么做？**

OpenAI 的示例里给出了一个很好的思路，那就是把问题拆分成三个步骤。


1. 把代码提交给大语言模型，让大语言模型解释一下，这个代码是在干什么。

2. 把代码以及代码的解释一起交给大语言模型，让大语言模型规划一下，针对这个代码逻辑，我们到底要写哪几个 TestCase。如果在这个过程中，大语言模型规划的 TestCase 数量太少，那么我们就重复第二步，让 AI 多生成几个 TestCase。
3. 针对上面的 TestCase 的详细描述，再提交给大语言模型，让它根据这些描述生成具体的测试代码。在这个过程中，我们还会对生成的代码，进行一次语法检查，如果语法检查没法通过，我们就要让 AI 重新生成一下。这个可以避免因为大语言模型的概率采样不稳定，导致生成的代码无法运行的问题。

最后，我们来实际运行一下这些代码，看看我们的代码能不能通过这些自动化测试。

请 AI 解释要测试的代码

那最后，我们就根据这个步骤一步步拆解，通过 Python 程序来把整个过程“自动化”掉。

 复制代码

```
1 def gpt35(prompt, model="text-davinci-002", temperature=0.4, max_tokens=1000,
2         top_p=1, stop=["\n\n", "\n\t\n", "\n    \n"]):
3     response = openai.Completion.create(
4         model=model,
5         prompt = prompt,
6         temperature = temperature,
7         max_tokens = max_tokens,
8         top_p = top_p,
9         stop = stop
10    )
11    message = response["choices"][0]["text"]
12    return message
13
14 code = """
15 def format_time(seconds):
16     minutes, seconds = divmod(seconds, 60)
17     hours, minutes = divmod(minutes, 60)
18
19     if hours > 0:
20         return f"{hours}h{minutes}min{seconds}s"
21     elif minutes > 0:
22         return f"{minutes}min{seconds}s"
23     else:
24         return f"{seconds}s"
25 """
26
```



```

27 def explain_code(function_to_test, unit_test_package="pytest"):
28     prompt = f"""# How to write great unit tests with {unit_test_package}
29
30 In this advanced tutorial for experts, we'll use Python 3.10 and `{unit_test_pack
31 ```python
32 {function_to_test}
33
34
35 Before writing any unit tests, let's review what each element of the function is
36 - First, """
37     response = gpt35(prompt)
38     return response, prompt
39
40 code_explanation, prompt_to_explain_code = explain_code(code)
41 print(code_explanation)

```

在第一步里，我们的代码做了这样几件事情。

首先是定义了一个 gpt35 的函数，对调用 GPT3.5 的模型做了简单的封装。其中有 2 点需要注意。

1. 我们默认使用 text-davinci-002 模型，这是一个通过监督学习微调的生成文本的模型。因为这里我们希望生成目标明确的文本的代码解释，所以选用了这个模型。
2. 我们对 stop 做了特殊的设置，只要连续两个换行或者类似连续两个换行的情况出现，就中止数据的生成。这是避免模型一口气连测试代码也生成出来。那样的话，我们没法对测试代码的生成提出具体的要求。通过 stop，我们可以确保在第一步，只解释现在的功能代码有什么用。

然后，我们通过一组精心设置过的提示语，让 GPT 模型为我们来解释代码。我们在提示语里做了 4 件事情。


指定了使用 pytest 这个测试包。

把对应要测试的代码，也提供给了 GPT 模型。

告诉 AI，要精确描述代码做了什么。

在最后一行用 “- First” 开头，引导 GPT 模型，逐步分行描述要测试的代码干了什么。

输出结果：


 复制代码

```
1 we use the `divmod` built-in function to get the quotient and remainder of `secc
2 - Next, we do the same thing with `minutes` and 60, assigning the results to `hou
3 - Finally, we use string interpolation to return a string formatted according to
```

运行第一步的代码，我们可以看到，AI 回复了几个步骤，详细地描述了我们格式化时间的代码是怎么做的。

请 AI 根据代码解释制定测试计划

接下来，我们就根据生成的这个详细描述，请 AI 为我们制定一下具体的测试计划。

 复制代码

```
1 def generate_a_test_plan(full_code_explanation, unit_test_package="pytest"):
2     prompt_to_explain_a_plan = f"""
3
4 A good unit test suite should aim to:
5 - Test the function's behavior for a wide range of possible inputs
6 - Test edge cases that the author may not have foreseen
7 - Take advantage of the features of `{unit_test_package}` to make the tests easy
8 - Be easy to read and understand, with clean code and descriptive names
9 - Be deterministic, so that the tests always pass or fail in the same way
10
11 `{unit_test_package}` has many convenient features that make it easy to write and
12
13 For this particular function, we'll want our unit tests to handle the following d
14 _"""
15     prompt = full_code_explanation + prompt_to_explain_a_plan
16     response = gpt35(prompt)
17     return response, prompt
18
19 test_plan, prompt_to_get_test_plan = generate_a_test_plan(prompt_to_explain_code
20 print(test_plan)
```


我们整个测试计划的提示语，同样经过了精心设计。我们先是对 AI 做了几个要求。

1. 我们要求测试用例，尽量考虑输入的范围广一些。
2. 我们要求 AI 想一些连代码作者没有想到过的边界条件。
3. 我们希望 AI 能够利用好 pytest 这个测试包的特性。
4. 希望测试用例清晰易读，测试的代码要干净。
5. 我们要求测试代码的输出结果是确定的，要么通过，要么失败，不要有随机性。

然后，我们的提示语并没有立刻让 AI 去写测试代码，而是说我们要举几个例子。这样，AI 就会生成一系列的示例。我们对测试用例的提示是非常详尽的，这也是我们前面第一步没有直接让 AI 生成测试用例的原因。因为那样的话，我们没法在提示语中间插入这些详尽的要求。对具体的测试用例，只能寄希望于 AI 想得多一些。

最后，我们发给 AI 的提示语，则是既包括了第一步要求解释代码的内容，也包括 AI 生成的对代码的解释，以及这里我们新增的对测试用例的要求，提供了非常详细的上下文，这样 AI 的表现也会更好、更有逻辑性。

输出结果：

 复制代码

```
1 Normal behavior:
2     - `format_time(0)` should return `"0s"`
3     - `format_time(59)` should return `"59s"`
4     - `format_time(60)` should return `"1min0s"`
5     - `format_time(119)` should return `"1min59s"`
6     - `format_time(3600)` should return `"1h0min0s"`
7     - `format_time(3601)` should return `"1h0min1s"`
8     - `format_time(3660)` should return `"1h1min0s"`
9     - `format_time(7200)` should return `"2h0min0s"`
10 - Invalid inputs:
11     - `format_time(None)` should raise a `TypeError`
12     - `format_time("abc")` should raise a `TypeError`
13     - `format_time(-1)` should raise a `ValueError`
```


我运行了一下这个代码，可以看到，AI 提供了很多测试用例。并且，里面考虑了好几种情况，包括我们前面提到的负数这样的特殊条件，也包括输入字符串，以及 None 这样的内

容。

不过，生成哪些用例其实是有一定的随机性的。这个也是大语言模型的一个缺点，就是可控性差。有时候，AI 可能就只生成了 3 个用例，那样的话就会有很多情况我们的用例覆盖不到。


所以，我们可以在生成用例之后，加一个步骤，检查一下到底生成了多少个用例。如果太少的话，我们就让 AI 再生成一些。我在下面给了一段示例代码，通过 “\n-” 这样一个换行加横杆的标记来判断之前生成的测试用例数量，如果比我们设定的下限少，我们就再添加一段提示语，让 AI 再生成一些。

这里的提示语，我们要特别提醒 AI 考虑一下测试罕见情况和边界条件。

 复制代码

```
1 not_enough_test_plan = """The function is called with a valid number of seconds
2     - `format_time(1)` should return `1s`
3     - `format_time(59)` should return `59s`
4     - `format_time(60)` should return `1min`
5 """
6
7 approx_min_cases_to_cover = 7
8 elaboration_needed = test_plan.count("\n-") + 1 < approx_min_cases_to_cover
9 if elaboration_needed:
10     prompt_to_elaborate_on_the_plan = f"""
11
12 In addition to the scenarios above, we'll also want to make sure we don't forget
13 -"""
14     more_test_plan, prompt_to_get_test_plan = generate_a_test_plan(prompt_to_
15     print(more_test_plan)
```

输出结果：

 复制代码


```
1 The function is called with a valid number of seconds
2     - `format_time(1)` should return `1s`
3     - `format_time(59)` should return `59s`
4     - `format_time(60)` should return `1min`
5 - The function is called with an invalid number of seconds
6     - `format_time(-1)` should raise a `ValueError`
7     - `format_time("60")` should raise a `TypeError`
```

```
8 - The function is called with a `None` value
9 - `format_time(None)` should raise a `TypeError`
```

根据测试计划生成测试代码

当然，大部分情况下，生成的测试用例数都和我们前面的实际情况是一样的。那我们就可以直接用原来的代码、代码的解释以及测试用例，作为提示语，让 AI 帮我们写具体的测试了。

这里的提示语也没有什么稀奇的，其实就是把前面已经生成的所有内容拼接在一起，然后要求 AI 根据前面的内容来写具体的测试代码。唯一有一点值得注意的是，为了避免 AI 忘记一开头功能代码的内容，我们特地再在提示语的最后，再次给 AI 看了一下我们要测试的功能代码。

 复制代码

```
1 def generate_test_cases(function_to_test, unit_test_package="pytest"):
2     starter_comment = "Below, each test case is represented by a tuple passed to
3     prompt_to_generate_the_unit_test = f"""
4
5     Before going into the individual tests, let's first look at the complete suite of
6     ```python
7     import {unit_test_package} # used for our unit tests
8
9     {function_to_test}
10
11     #{starter_comment}"""
12     full_unit_test_prompt = prompt_to_explain_code + code_explanation + test_pla
13     return gpt35(model="text-davinci-003", prompt=full_unit_test_prompt, stop="``
14
15 unit_test_response, prompt_to_generate_the_unit_test = generate_test_cases(code)
16 print(unit_test_response)
```

输出结果：

 复制代码

```
1 .
2 #The first element of the tuple is the name of the test case, and the second eler
3 @pytest.mark.parametrize('test_input,expected', [
4     ('0', '0s'),
5     ('59', '59s'),
6     ('60', '1min0s'),
```

```


7     ('119', '1min59s'),
8     ('3600', '1h0min0s'),
9     ('3601', '1h0min1s'),
10    ('3660', '1h1min0s'),
11    ('7200', '2h0min0s'),
12 ])
13 def test_format_time(test_input, expected):
14     #For each test case, we call the format_time() function and compare the retur
15     assert format_time(int(test_input)) == expected
16
17 #We use the @pytest.mark.parametrize decorator again to test the invalid inputs.
18 @pytest.mark.parametrize('test_input', [
19     None,
20     'abc',
21     -1
22 ])
23 def test_format_time_invalid_inputs(test_input):
24     #For each invalid input, we expect a TypeError or ValueError to be raised.
25     with pytest.raises((TypeError, ValueError)):
26         format_time(test_input)

```

运行一下这段提示语，我们就拿到了最终输出的测试代码。可以看到，这个测试代码不仅有正常情况下的测试，也包含了异常输入的测试。

通过 AST 库进行语法检查

不过这还没有完，我们最好还是再检查一下生成的测试代码的语法，这个可以通过 Python 的 AST 库来完成。不过需要注意，检查语法的时候，我们不仅需要生成的测试代码，也需要原来的功能代码，不然是没办法通过语法检查的。


 复制代码

```

1 import ast
2
3 code_start_index = prompt_to_generate_the_unit_test.find("`python\n") + len("`
4 code_output = prompt_to_generate_the_unit_test[code_start_index:] + unit_test_res
5 try:
6     ast.parse(code_output)
7 except SyntaxError as e:
8     print(f"Syntax error in generated code: {e}")


```

很幸运，我们一次就通过了语法检查。那么接下来，我们就可以把对应的整个测试代码打印出来，执行试一试。

 复制代码

```
1 print(code_output)
```

输出结果：

 复制代码

```
1 import pytest # used for our unit tests
2
3
4
5 def format_time(seconds):
6     minutes, seconds = divmod(seconds, 60)
7     hours, minutes = divmod(minutes, 60)
8
9     if hours > 0:
10         return f"{hours}h{minutes}min{seconds}s"
11     elif minutes > 0:
12         return f"{minutes}min{seconds}s"
13     else:
14         return f"{seconds}s"
15
16
17
18 #Below, each test case is represented by a tuple passed to the @pytest.mark.param
19 #The first element of the tuple is the name of the test case, and the second eler
20 @pytest.mark.parametrize('test_input,expected', [
21     ('0', '0s'),
22     ('59', '59s'),
23     ('60', '1min0s'),
24     ('119', '1min59s'),
25     ('3600', '1h0min0s'),
26     ('3601', '1h0min1s'),
27     ('3660', '1h1min0s'),
28     ('7200', '2h0min0s'),
29 ])
30 def test_format_time(test_input, expected):
31     #For each test case, we call the format_time() function and compare the retur
32     assert format_time(int(test_input)) == expected
33
34
```

```

35 #We use the @pytest.mark.parametrize decorator again to test the invalid inputs.
36 @pytest.mark.parametrize('test_input', [
37     None,
38     'abc',
39     -1
40 ])
41 def test_format_time_invalid_inputs(test_input):
42     #For each invalid input, we expect a TypeError or ValueError to be raised.
43     with pytest.raises((TypeError, ValueError)):
44         format_time(test_input)
45

```

看看自动生成的测试帮我们抓了什么 Bug

我们可以把对应生成的代码，单独复制到一个 `auto_unit_test.py` 文件里面。然后去命令行里执行一下 `pytest` 这个命令，看看结果是怎样的。我这里，对应的会有一个测试用例失败，就是当输入是 `-1` 的时候，测试用例预期会遇到一个 `TypeError` 或者 `ValueError` 的报错，但是实际并没有。

```

auto_unit_test.py .....F [100%]

===== FAILURES =====
test_format_time_invalid_inputs[-1]

test_input = -1

@pytest.mark.parametrize('test_input', [
    None,
    'abc',
    -1
])
def test_format_time_invalid_inputs(test_input):
    #For each invalid input, we expect a TypeError or ValueError to be raised.
>   with pytest.raises((TypeError, ValueError)):
E   Failed: DID NOT RAISE (<class 'TypeError'>, <class 'ValueError'>)

auto_unit_test.py:41: Failed

===== short test summary info =====
FAILED auto_unit_test.py::test_format_time_invalid_inputs[-1] - Failed: DID NOT RAISE (<class 'TypeError'>, <class 'ValueError'>)
===== 1 failed, 10 passed in 0.16s =====
(py310)> openai-course git:(main) x

```

我们可以试着在 Notebook 里面调用一下 `format_time(-1)`，看看自动化测试跑得对不对。

 复制代码

```
1 format_time(-1)
```

输出结果：


```
1 '59min59s'
```

可以看到，输入 -1 的时候，输出变成了 59min59s，看来 AI 生成的测试代码的确帮我们捕捉到了一个 Bug。

小结

好了，到这里这一讲也就结束了。我们也有了一段完整的，可以针对一个 Python 函数生成一整套自动化测试的功能了，而且它还真的帮我们抓到了一个 Bug。**生成整套测试代码的过程中，我们不需要人工地复制粘帖任何内容，全都是代码自动完成的，是一个“自动档”的过程。**

之所以能做到这一点，是因为我们巧妙地利用了一个方法，就是将一个问题，拆分成多个提示语的步骤，循序渐进地让 AI 通过解释代码，构造测试用例，最后再根据代码的解释和设计的测试用例，生成最终的自动化测试。

多步提示语带来的一个好处，就是我们的内容是更加有条理、有逻辑的，也更符合我们平时写文字的方式，而不是一股脑地把各种要求都放在提示语的开头，这在解决复杂问题时往往效果不好。

此外，我们这里使用的提示语也非常重要。正是因为我们能够分步骤提示 AI，所以我们能在拿到代码的解释之后，让 AI 考虑各种边界条件。而 AI 也很给力，给出 -1、None 这样的特殊输入，让我们的测试代码最终真的抓住了程序里的 Bug。

回过头来看，如果我们只是直接把代码往 ChatGPT 里一贴，虽然也能生成测试用例，但是那些测试用例就比较欠考虑，不会涵盖各种边角的情况。

思考题

这一讲的代码有点长，思考题部分需要你做的事情也多一些。

1. 你可以试着减少我们的提示语或者提示步骤，看看生成的测试用例有什么样的变化。
2. 目前我们的代码，是过程式地一步步给你演示整个测试代码是如何生成的。如果语法检查出错了，其实我们应该要从头开始重试一遍，再生成测试代码。你可以试着把整个代码封装修改，变成一个会自动重试 3 次的函数。让我们可以直接调用，来为 Python 代码生成自动化测试。
3. 我们这一讲里的提示语是借鉴的 OpenAI Cookbook 里的样例，你能不能自己尝试总结一下，这些提示语有哪些值得借鉴的常用方法？

欢迎你把思考后的结果分享在评论区，也欢迎你把这一讲分享给感兴趣的朋友，我们下一讲再见。

推荐阅读

之所以我们要循序渐进地提示 AI，让 AI 生成例子再生成代码，是因为现在的大语言模型，有一种叫做“思维链（CoT）”的能力。当我们给出更详细的推理步骤的时候，AI 的表现会更好。在 OpenAI Cookbook 里，专门有一章讲解 [思维链能力](#)，你可以去好好研读一下。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (9)



peter

2023-04-10 来自北京

请教老师一个问题：

Q1: AI唱歌，有能够使用的软件吗？（开源、付费的都可以）。AI唱歌，是指用一个人的声音把一首歌完整的唱出来。比如有特朗普的一段音频（比如30s声音片段），然后软件根据这个声音片段，就可以把《好汉歌》唱出来。效果就是听众认为是特朗普唱的《好汉歌》。

作者回复: AI可以训练一个属于你自己声音的模型，这个无论是云平台比如Azure，还是开源的比如PaddleSpeech都有方案。不过一般都是用在合成说话，但是我想换唱歌应该也不难，现成的不一定有。

共 2 条评论 >

👍 3



Toni

2023-04-10 来自瑞士

Part 1

怎么"玩" ChatGPT，好题目。依据老师提供的思维链能力一文中'Let's think step by step'，和 ChatGPT 玩个小游戏。用ChatGPT 最擅长的英语进行测试，以排除语言干扰因素。

受2000字符的限制，只能将结果分集了。通过这个小小的测试，对ChatGPT 有了进一步的了解。下面是第一个测试问题："杂技演员有 16 个球。一半的球是高尔夫球，一半的高尔夫球是蓝色的。有多少个蓝色高尔夫球？" 正确的答案是有4个蓝色的高尔夫球。看看ChatGPT 表现如何：

问: A juggler has 16 balls. Half of the balls are golf balls and half of the golf balls are blue. How many blue golf balls are there?

答: If a juggler has 16 balls and half of the balls are golf balls, then there are $16 / 2 = 8$ golf balls in total. Since half of the golf balls are blue, there are $8 / 2 = 4$ blue golf balls.

答案是正确的，可见ChatGPT3.5 已比 3.0 版本进化了。但对它的处理方法 ' $8 / 2 = 4$ ' 有疑问。

接下来给ChatGPT 挖了个坑，看它能否跳过去。设置的问题是"一个变戏法者有 16 个球。一半的球是高尔夫球，一半的高尔夫球是蓝色的。少了一个白球，有多少个蓝色高尔夫球？" 这里使用了一个干扰项，不相关的白球。

问: A juggler has 16 balls. Half of the balls are golf balls and half of the golf balls are blue. A white ball is missing, how many blue golf balls are there?

答: If a juggler has 16 balls and half of the balls are golf balls, then there are $16 / 2 = 8$ golf balls in total.

However, since a white ball is missing, the total number of golf balls would be reduced by 1, which means there are $8 - 1 = 7$ golf balls remaining.

Since half of the golf balls are blue, there would be $7 / 2 = 3.5$ blue golf balls. However, since it's not possible to have half a ball, we can conclude that there are no blue golf balls, as the number of golf balls is not evenly divisible by 2, after accounting for the missing white ball.

ChatGPT 没有给出正确答案。也认识到3.5个蓝高尔夫球的答案不合理，但试图改这个不合理

却带出了更大的错误没有蓝高尔夫球(we can conclude that there are no blue golf balls)。

这个错误暗示ChatGPT 没能进行红蓝分类。进一步的测试在part 2.

作者回复: 🍌 推理能力才是chatgpt 特别是 gpt-4的最厉害的地方

共 2 条评论 >

👍 1



Toni

2023-06-01 来自瑞士

OpenAI 5月底官宣，在数学推理过程中加入监督奖励模型可极大地提高了AI的推理能力。现在ChatGPT处理比4月10日展示中复杂得多的问题都已不在话下，短短一个半月，发展日新月异。有志在模型改造方面有建树的同学可参考原文，获取思路灵感。

<https://openai.com/research/improving-mathematical-reasoning-with-process-supervision>

We've trained a model to achieve a new state-of-the-art in mathematical problem solving by rewarding each correct step of reasoning ("process supervision") instead of simply rewarding the correct final answer ("outcome supervision"). In addition to boosting performance relative to outcome supervision, process supervision also has an important alignment benefit: it directly trains the model to produce a chain-of-thought that is endorsed by humans.

中文翻译:

我们训练了一个模型，通过奖励每一个正确的推理步骤（“过程监督”）而不是简单地奖励正确的最终答案（“结果监督”）来实现数学问题解决的最新技术水平。除了提高与结果监督相关的性能外，过程监督还有一个重要的校准对标好处：它直接训练模型以产生人类认可的思维链。



金hb.Ryan 冷空氣...

2023-05-22 来自上海

让GPT总结了一下 [openai-cookbook /techniques_to_improve_reliability.md](#)

>>

本文介绍了如何提高大型语言模型在复杂任务上的可靠性，包括寻找更好的提示、将任务分解为子任务、给出更清晰的指令等技巧。同时，文章还介绍了逐步推理、少样本示例和微调等技术，以及链式思考提示、选择-推理提示、忠实推理架构和从最少到最多提示等技术，都旨在将复杂任务分解为更小、更可靠的子任务，并给模型更多的时间和空间来解决问题。此外，文章还提到了如何将这些技术应用于概率图模型中，以便更好地选择、组合和发现新技术。



Ethan New

2023-04-27 来自浙江

学习打卡



金口

2023-04-11 来自湖南

CoT确实得利用起来



Toni

2023-04-10 来自瑞士

part 2

为帮助ChatGPT 应对 part 1 中的白球干扰，提示球的颜色有两类。问题是这样的"变戏法者有 16 个球，白色或蓝色，高尔夫球和非高尔夫球。问题是：如果已知一半的球是高尔夫球，并且一半的高尔夫球是蓝色的。如果少了一个白球，他有多少个蓝色高尔夫球？如果您理解问题，请回答。如果不清楚，请询问。"

问: The juggler has 16 balls, white or blue, golf and non-golf. The question is: if half of the balls are known to be golf balls, and half of the golf balls are blue. If one white ball is missing, how many blue golf balls does he have? Please answer if you understand the question. If something is unclear, please ask.

答: I apologize for the confusion in my previous responses. Based on the information provided, we can deduce the following:

There are 16 balls in total.

Half of the balls are golf balls, which means there are $16 / 2 = 8$ golf balls.

Half of the golf balls are blue, which means there are $8 / 2 = 4$ blue golf balls.

Now, if one white ball is missing, it does not affect the total number of golf balls or the proportion of blue golf balls, as the missing ball is white and not a golf ball. So, the number of blue golf balls would still be 4, based on the original information provided.

ChatGPT 给出的'4个蓝色的高尔夫球'的结论是对的，但它的依据 'as the missing ball is white and not a golf ball.' 并不完全正确。在提醒注意区别白色和蓝色后，ChatGPT 基本能理解白球

的丢失不影响蓝球的数量这一'逻辑'。

上面的对话测试并不是单单在找 ChatGPT 的错，而是在对提问进行不断的调优。能否清楚地给出提示词是挖掘 ChatGPT 潜力的关键，诚如老师在本课所讲的。

不知 ChatGPT 4 在应对上面挑战时的表现如何。

最后一段对话放在下面：

问: Considering the difference between white balls and blue balls, will losing white balls affect the number of blue balls?

答: Based on the information provided, which states that half of the balls are golf balls and half of the golf balls are blue, losing white balls would not affect the number of blue balls.

...

作者回复: GPT-4的回复

I understand the question. The juggler has 16 balls in total, with half of them being golf balls. This means there are 8 golf balls. Half of the golf balls are blue, so there are 4 blue golf balls. Since one white ball is missing, this does not affect the count of blue golf balls. Therefore, the juggler has 4 blue golf balls.

共 2 条评论 >



Oli张帆

2023-04-10 来自北京

既然老师讲到了prompt engineering的应用，我也问一个相关的问题。我有一个很复杂的任务，使用Turbo Chat API(GPT 3.5)来完成，目的是喂给OpenAI一篇英文文章，然后根据用户的语言，输出一个JSON格式的数据，供前端来生成Quiz。大概的prompt是这样的：

...

Create a quiz with n questions (translated to Chinese) for the given context in JSON. Return a JSON string with an array of n quiz questions, each represented as a JSON object with a 'question', an 'answer', a 'options' and an 'explanation' property.

Example output:

```
[{"question":"What is the largest organ in the human body?","answer":"Skin","options":["Liver","Heart","Lungs"],"explanation":"The skin is the largest organ in the human body, and it serves as a protective barrier against external threats such as pathogens, ultraviolet radiation, and dehydration. It also plays a crucial role in regulating body temperature and maintaining a healthy immune system."},{ "question":"What is the smallest country in the world?","answer":"Vatican City","options":["Monaco","Liechtenstein","Nauru"],"explanation":"Vatican City is the smallest country in the world, both in terms of area and population. It is an independent city-state enclaved within Rome, Italy, and it is the spiritual and administrative center of the Roman Catholic Church."}]
```

###

The context:

一篇文章，很长。

...

我发现了三个问题，一是如果文章的token超过1024个，OpenAI就不会生成正确的JSON格式，因此我只能把文章按照1024的token限制先切分，再分别请求；二是如果想要让OpenAI始终能按照我的要求将数据翻译成相应的语言，就需要在prompt中把example也先翻译为相应的语言；三是这个请求总是无法在30秒内返回，因为我使用的Heroku有30秒的硬限制，所以这种请求必须放在background worker进行，然后前端通过poll的方式来获取结果。

虽然知道对应的解决方法，但是还是希望老师帮我从LLM的角度理解一下这些问题的原因，帮助我更好地理解。

作者回复: 对于文章或者Context比较长，我有时会选择用 >>> 和 <<< 隔开，并在Prompt中提示。类似于Langchain中Example的一些

```
""Between >>> and <<< are the raw search result text from google.
```

```
Extract the answer to the question '{query}' or say "not found" if the information is not contained.
```

```
Use the format
```

```
Extracted:<answer or "not found">
```

```
>>> {requests_result} <<<
```

```
Extracted:""
```

对于语言，目前我测试下来，建议用英语，然后结果再翻译。目前英语无论是推理能力还是知识都是效果最好的。

另外在你这个例子里面，完全可以先只生成Question和Options，然后再要求根据Questions和Options，生成Answer和Explanation。不然我看现在Answer都不是Options中的一个。这个逻辑看起来拆分步骤推理更合适，而不是一次让LLM生成完。



geek_time

2023-04-10 来自广东

中间有把一部分格式乱了

编辑回复：感谢提醒，已更新🌹

