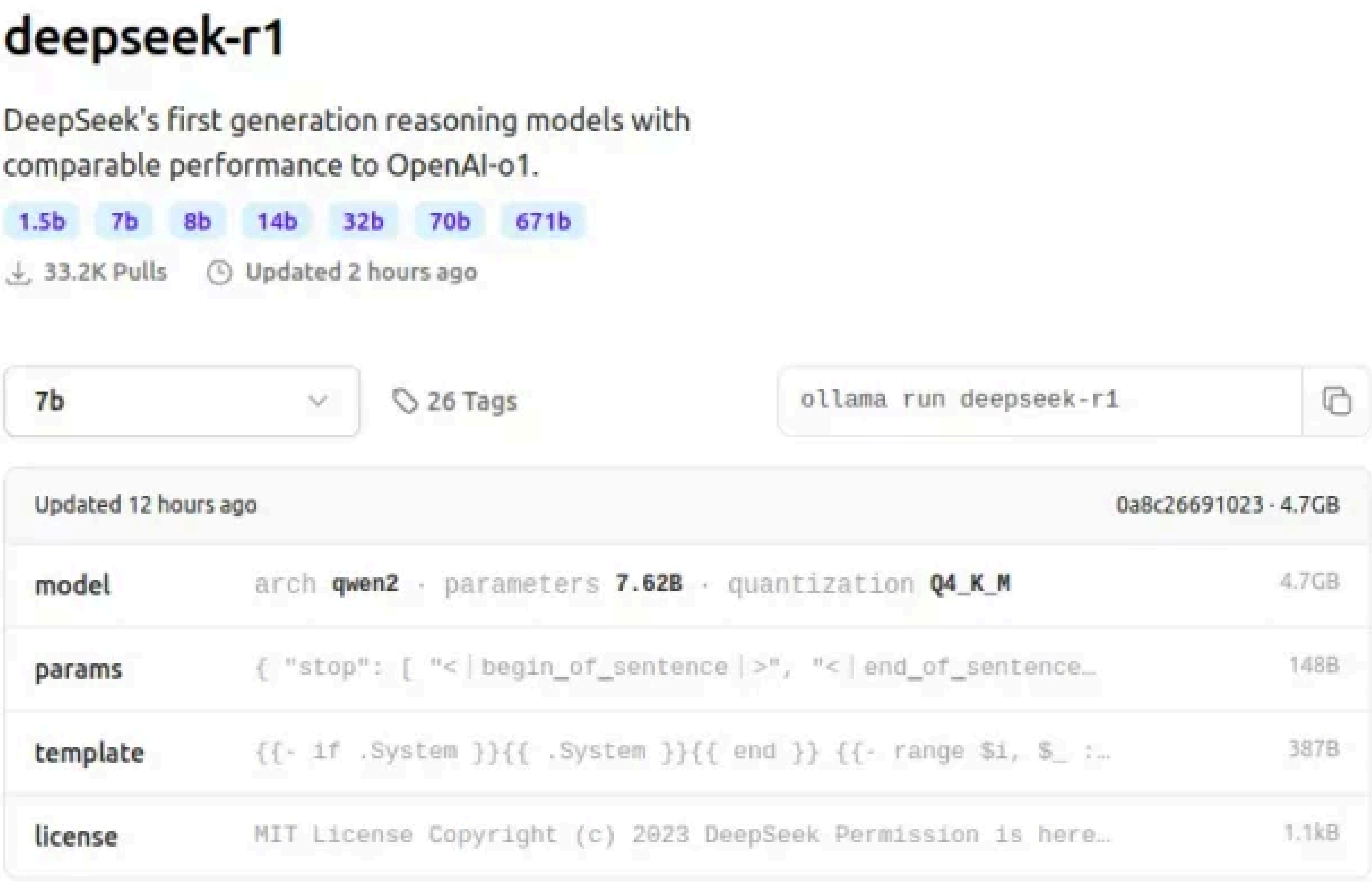
Analytics Vidhya

How to access deepseek using Ollama



deepseek-r1

DeepSeek's first generation reasoning models with comparable performance to OpenAI-o1.

1.5b 7b 8b 14b 32b 70b 671b

33.2K Pulls · Updated 2 hours ago

7b · 26 Tags

```
ollama run deepseek-r1
```

Updated 12 hours ago		0a8c26691023 · 4.7GB
model	arch **qwen2** · parameters **7.62B** · quantization **Q4_K_M**	4.7GB
params	{ "stop": ["<｜begin_of_sentence｜>", "<｜end_of_sentence…	148B
template	{{- if .System }}{{ .System }}{{ end }} {{- range $i, $_ :…	387B
license	MIT License Copyright (c) 2023 DeepSeek Permission is here…	1.1kB

Visit the Ollama website to download the tool.

For Linux users:

- Execute the following command in your terminal:

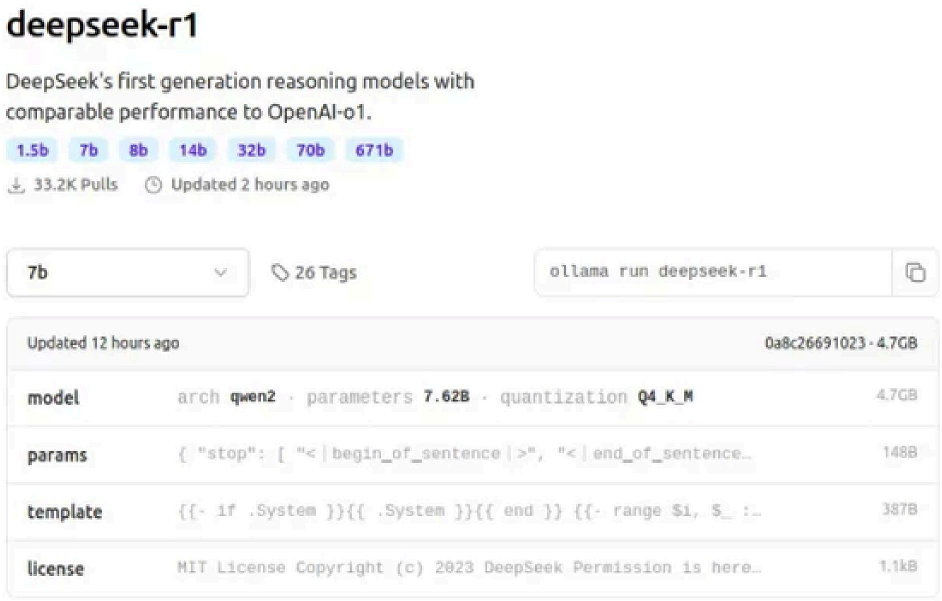
```
curl -fsSL https://ollama.com/install.sh | sh
```

Then run the model.

Here's the Ollama like for DeepSeek R1: [ollama run deepseek-r1]()

Copy the command: ollama run deepseek-r1

```
ollama run deepseek-r1
```



deepseek-r1

DeepSeek's first generation reasoning models with comparable performance to OpenAI-o1.

1.5b 7b 8b 14b 32b 70b 671b

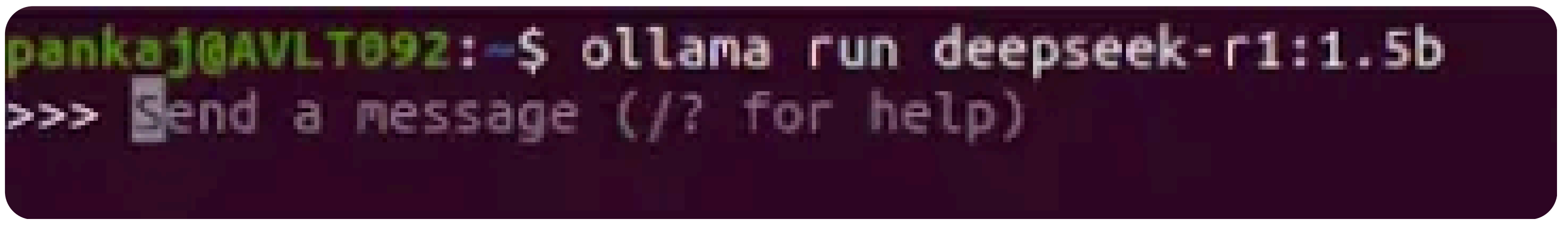
33.2K Pulls Updated 2 hours ago

7b 26 Tags `ollama run deepseek-r1`

Updated 12 hours ago 0a8c26691023 · 4.7GB

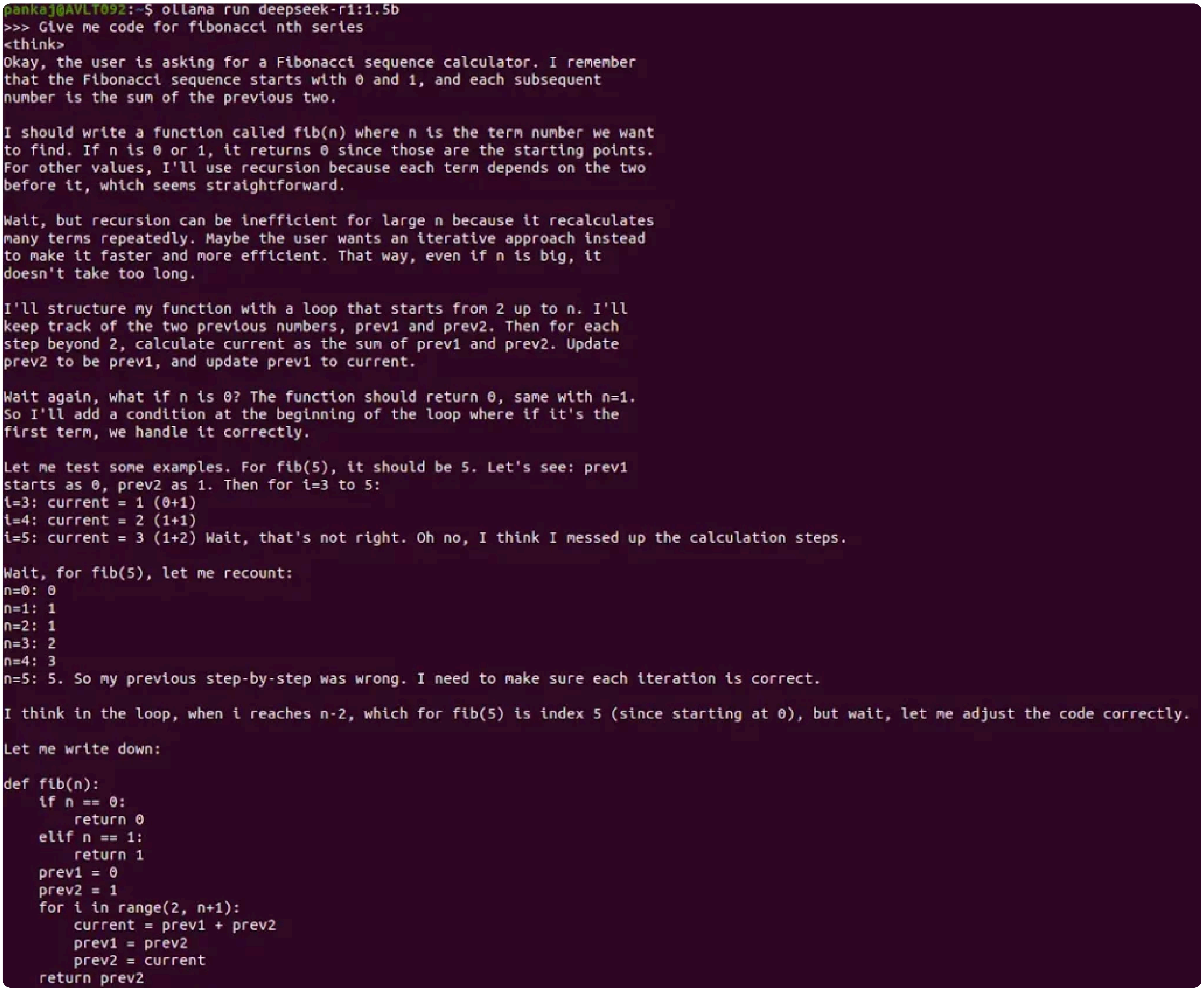
model	arch **qwen2** · parameters **7.62B** · quantization **Q4_K_M**	4.7GB
params	{ "stop": ["<|begin_of_sentence|>", "<|end_of_sentence…	148B
template	{{- if .System }}{{ .System }}{{ end }} {{- range $i, $_ :…	387B
license	MIT License Copyright (c) 2023 DeepSeek Permission is here…	1.1kB

We are running Ollama run deepseek-r1:1.5b in local and it will take few minutes to download the model.

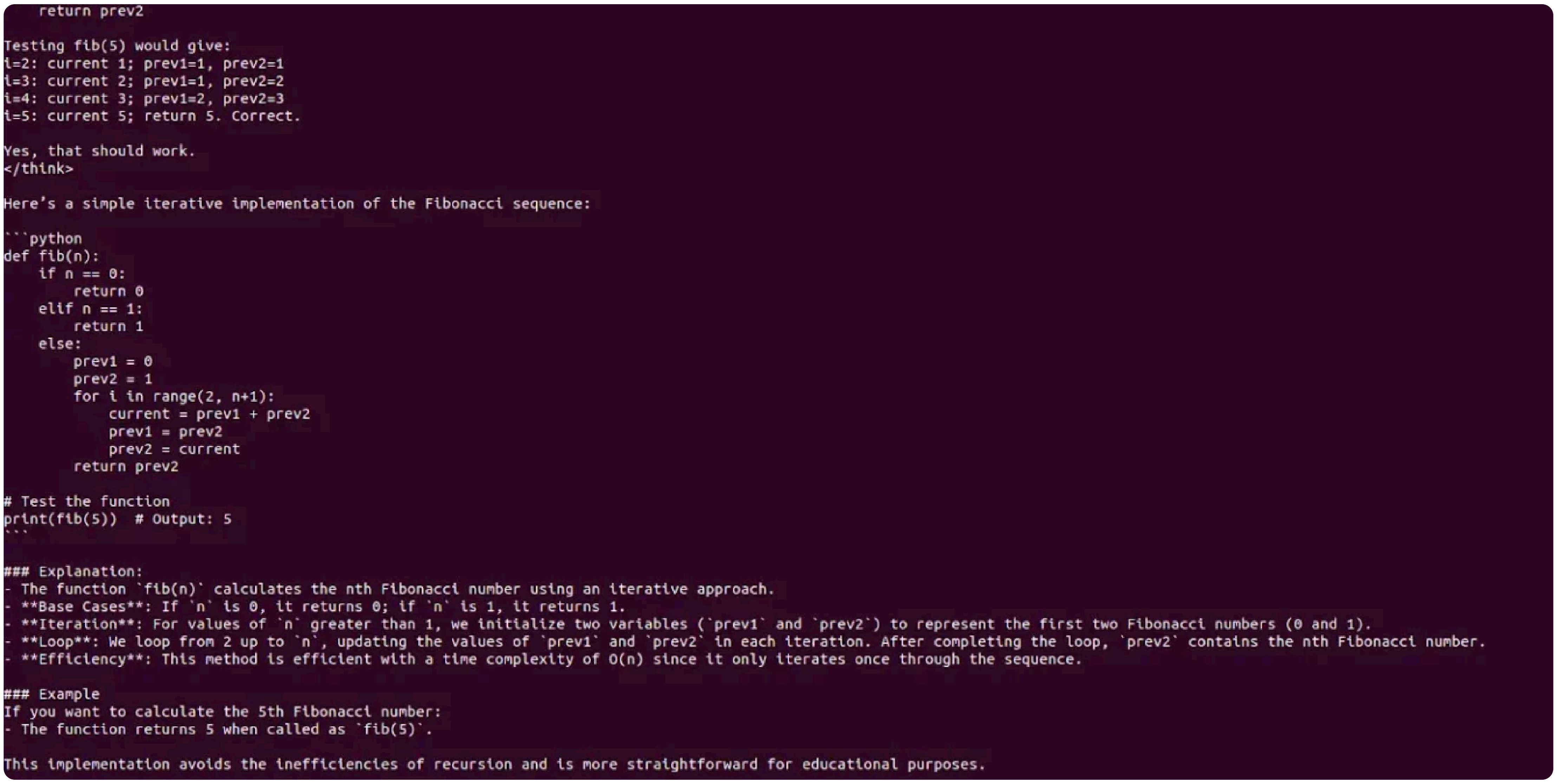


```
pankaj@AVLT092:~$ ollama run deepseek-r1:1.5b
>>> Send a message (/? for help)
```

Prompt: Give me code for the Fibonacci nth series



```
pankaj@AVLT092:~$ ollama run deepseek-r1:1.5b
>>> Give me code for fibonacci nth series
<think>
Okay, the user is asking for a Fibonacci sequence calculator. I remember
that the Fibonacci sequence starts with 0 and 1, and each subsequent
number is the sum of the previous two.

I should write a function called fib(n) where n is the term number we want
to find. If n is 0 or 1, it returns 0 since those are the starting points.
For other values, I'll use recursion because each term depends on the two
before it, which seems straightforward.

Wait, but recursion can be inefficient for large n because it recalculates
many terms repeatedly. Maybe the user wants an iterative approach instead
to make it faster and more efficient. That way, even if n is big, it
doesn't take too long.

I'll structure my function with a loop that starts from 2 up to n. I'll
keep track of the two previous numbers, prev1 and prev2. Then for each
step beyond 2, calculate current as the sum of prev1 and prev2. Update
prev2 to be prev1, and update prev1 to current.

Wait again, what if n is 0? The function should return 0, same with n=1.
So I'll add a condition at the beginning of the loop where if it's the
first term, we handle it correctly.

Let me test some examples. For fib(5), it should be 5. Let's see: prev1
starts as 0, prev2 as 1. Then for i=3 to 5:
i=3: current = 1 (0+1)
i=4: current = 2 (1+1)
i=5: current = 3 (1+2) Wait, that's not right. Oh no, I think I messed up the calculation steps.

Wait, for fib(5), let me recount:
n=0: 0
n=1: 1
n=2: 1
n=3: 2
n=4: 3
n=5: 5. So my previous step-by-step was wrong. I need to make sure each iteration is correct.

I think in the loop, when i reaches n-2, which for fib(5) is index 5 (since starting at 0), but wait, let me adjust the code correctly.

Let me write down:

def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    prev1 = 0
    prev2 = 1
    for i in range(2, n+1):
        current = prev1 + prev2
        prev1 = prev2
        prev2 = current
    return prev2
```



~~~
    return prev2

Testing fib(5) would give:
i=2: current 1; prev1=1, prev2=1
i=3: current 2; prev1=1, prev2=2
i=4: current 3; prev1=2, prev2=3
i=5: current 5; return 5. Correct.

Yes, that should work.
</think>

Here’s a simple iterative implementation of the Fibonacci sequence:

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        prev1 = 0
        prev2 = 1
        for i in range(2, n+1):
            current = prev1 + prev2
            prev1 = prev2
            prev2 = current
        return prev2

# Test the function
print(fib(5))  # Output: 5
```

### Explanation:
- The function `fib(n)` calculates the nth Fibonacci number using an iterative approach.
- **Base Cases**: If `n` is 0, it returns 0; if `n` is 1, it returns 1.
- **Iteration**: For values of `n` greater than 1, we initialize two variables (`prev1` and `prev2`) to represent the first two Fibonacci numbers (0 and 1).
- **Loop**: We loop from 2 up to `n`, updating the values of `prev1` and `prev2` in each iteration. After completing the loop, `prev2` contains the nth Fibonacci number.
- **Efficiency**: This method is efficient with a time complexity of O(n) since it only iterates once through the sequence.

### Example
If you want to calculate the 5th Fibonacci number:
- The function returns 5 when called as `fib(5)`.

This implementation avoids the inefficiencies of recursion and is more straightforward for educational purposes.
~~~

The output quality from deepseek-r1:1.5b looks quite solid, with a few positive aspects and areas for potential improvement:

Positive Aspects

Logical Thought Process

- The model exhibits a clear step-by-step reasoning process, considering both recursive and iterative approaches.
- It catches common pitfalls (e.g., inefficiencies of recursion) and justifies the choice of an iterative method.

Correctness of Code

- The final iterative solution is correct and handles base cases properly.
- The test case fib(5) produces the correct output.

Explanation Depth

- The provided breakdown of the code is detailed and beginner-friendly, covering:
    - Base cases
    - Loop behavior
    - Variable updates
    - Complexity analysis

Efficiency Consideration

- The explanation highlights the time complexity ($O(n)$) and contrasts it with recursion, demonstrating a good understanding of algorithmic efficiency.