Assignment 3 - LLM Deployment Report

Aline UWIMANA

2025-06-11

Table of Contents

Assignment 3: Deployment and Integration of LLMs

1. Introduction

This assignment demonstrates the deployment and integration of both online and local Large Language Models (LLMs) to support programming tasks. I utilized the Hugging Face Transformers library for online (pipeline-based) text generation and Ollama for local inference with open LLMs. The goal was to evaluate the installation, usage, integration, and performance of both solutions in a real development environment.

---

2. Methodology

2.1. Online LLM Deployment (Hugging Face)

Step-by-step Setup:

1. **Create a Hugging Face Account**
   - Go to https://huggingface.co
   - Click on **Sign Up** and fill in your email, username, and password.
   - Verify your email address if prompted.
2. **Generate an Access Token**
   - After logging in, click on your profile icon (top-right corner) and go to **Settings**.
   - Navigate to the **Access Tokens** tab.
   - Click **New Token**, give it a name, select permission scope (I used `read`) because is safe and help for downloading and running models, then click **Generate**.
   - Copy and save the token securely. You will need this to authenticate from your development environment.
3. **Install Python and Required Packages**
   - After generating token from huggingface I first Ensured Python is installed. Didn't find, then I downloaded it from https://www.python.org/downloads/
   - Then install the required libraries:
     `pip install transformers huggingface_hub torch`
4. **Use the Hugging Face Transformers Pipeline in VSCode**
   - Here I tested the first code in VSCode by creating this Python script: "'python from transformers import pipeline from huggingface_hub import login

Login using generated token

login("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")

Load a small model (gpt2 is free and works easily)

generator = pipeline("text-generation", model="gpt2")

Run inference

result = generator("The future of AI is", max_length=50, num_return_sequences=1) print(result[0]["generated_text"])

5. **Run the Code**
   - Execute the script in the terminal. The output was this.

Output
***
`![alt text](assignment3_images/huggingface-test.png)`

***

2.2. Local LLM Deployment (Ollama)

1. **Install Ollama**
   - Visit https://ollama.com and download the version for your op
   - Follow the installation instructions. Make sure Ollama runs in the background or as

2. **Pull a Local Model**
   - In the terminal, run:
     ```bash
     ollama run mistral
     ```
   - This downloads the Mistral model for local inference.
3. **Run the Script**
   - Execute it in the terminal and observe the output generated locally this is first c

Output
***

`![alt text](assignment3_images/Ollama_test.png)`

***

2.3. Integration in Development Environment

- Both Hugging Face and Ollama models were tested in **VSCode on Windows**.

````
- They were **run in separate Python scripts**, not simultaneously.
- I attempted to compare speed and response time for both models. However, **my laptop f
- Despite this, **each model worked correctly when run independently**, and successfully

---

## 3. Results

### 3.1. Hugging Face Output

- The Hugging Face model (`gpt2`) correctly generated Python code for a factorial functi

- It was effective for programming prompts, although occasionally verbose.

```python
from transformers import pipeline
from huggingface_hub import login

# Login using your token
login("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")

# Load a small model (gpt2 is free and works easily)
generator = pipeline("text-generation", model="gpt2")

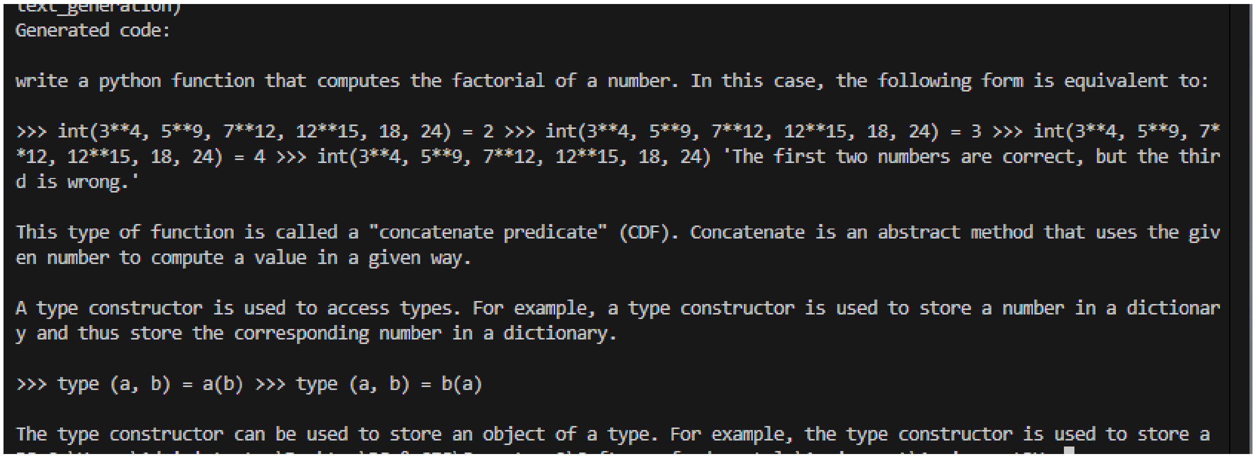
# Prompt for code generation
# prompt = "write a python function that computes the factorial of a number."
prompt = "provide the python code only for a factorial function."

# Generate output
result = generator(prompt, max_length=60, num_return_sequences=1)
print("Generated code:\n")
print(result[0]["generated_text"])
````

output

3.2. Ollama Output

- The locally-run Mistral model via Ollama also provided accurate factorial code.
- It gave concise output and functioned reliably when not running alongside other memory-intensive processes.
````



```
text_generation)
Generated code:

write a python function that computes the factorial of a number. In this case, the following form is equivalent to:

>>> int(3**4, 5**9, 7**12, 12**15, 18, 24) = 2 >>> int(3**4, 5**9, 7**12, 12**15, 18, 24) = 3 >>> int(3**4, 5**9, 7*
*12, 12**15, 18, 24) = 4 >>> int(3**4, 5**9, 7**12, 12**15, 18, 24) 'The first two numbers are correct, but the thir
d is wrong.'

This type of function is called a "concatenate predicate" (CDF). Concatenate is an abstract method that uses the giv
en number to compute a value in a given way.

A type constructor is used to access types. For example, a type constructor is used to store a number in a dictionar
y and thus store the corresponding number in a dictionary.

>>> type (a, b) = a(b) >>> type (a, b) = b(a)

The type constructor can be used to store an object of a type. For example, the type constructor is used to store a
```

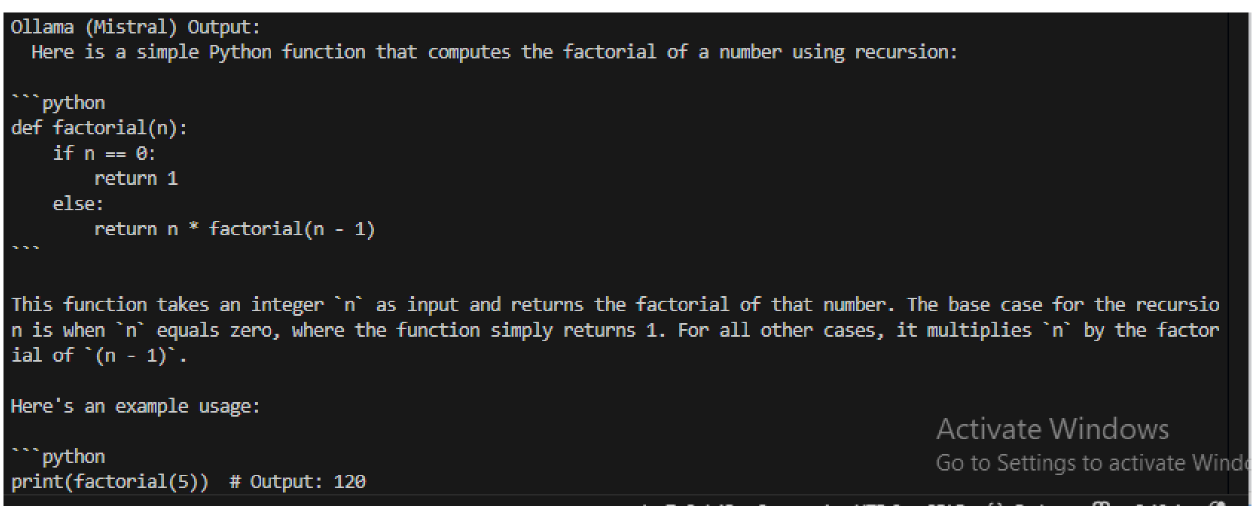
Figure 1: alt text

3. **Install the Ollama Python Package**
    - If using Python, install the client package:
      `pip install ollama`
4. **Run Local Inference**
    - Create a Python script in VSCode:

```python
import ollama

response = ollama.chat(model='mistral', messages=[
    {'role': 'user', 'content': 'Write a Python function that computes the fact
])
print(response['message']['content'])
```

Output



```
Ollama (Mistral) Output:
  Here is a simple Python function that computes the factorial of a number using recursion:

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

This function takes an integer `n` as input and returns the factorial of that number. The base case for the recursio
n is when `n` equals zero, where the function simply returns 1. For all other cases, it multiplies `n` by the factor
ial of `(n - 1)`.

Here's an example usage:

```python
print(factorial(5))  # Output: 120
```
```

```
Here's an example usage:

```python
print(factorial(5))  # Output: 120
```

In this case, we get the factorial of 5 (which is the product of all positive integers less than or equal to 5), res
ulting in 120.
```

---

4. Comparison

Aspect	Hugging Face (gpt2 pipeline)	Ollama (Mistral)
Prompt	Provide Python code for factorial	Write Python function for factorial
Output	Correct, sometimes verbose	Correct, concise
Setup	Hugging Face account, token, pip	Ollama install, model pull, pip
Privacy	Requires cloud access	Fully local inference
Hardware needed	Internet connection sufficient	High RAM usage
Usability	Easy and beginner-friendly	More setup, but no internet needed
Challenges	Needs access token and internet	System froze under load

---

5. Reflections & Challenges

- **Hugging Face** was simple to set up, produced quality outputs, and worked well in VSCode.
- **Ollama** was effective for local inference and offered better privacy, but **its RAM demands** made simultaneous comparisons difficult on my laptop.
- The biggest challenge was **system stability** when attempting to run both models at once.
- Overall, both tools are valuable and practical for enhancing programming and research workflows.

---

6. Conclusion

Deploying both online (Hugging Face) and local (Ollama) LLMs in my development workflow allowed me to generate and explain code efficiently. Hugging Face is lightweight, cloud-based, and user-friendly, ideal for quick integration. Ollama offers privacy and offline functionality, making it useful in restricted environments, although it comes with a heavier hardware requirement. Both tools can significantly enhance productivity in programming and research support.
