Advanced Robot Learning

SS 2025

Task 3

Group 2

Name	Mat.Nummer
Dominic Khang	12012414
Chung-Hsuan Ryan Lee	52112880
Dominik Pusztai	12009186
Pascal Zöhrer	11941484

June 3, 2025

1 Introduction

The goal of this assignment is to become familiar working with Large Language Models (LLMs) for reasoning and task planning, with the aim of applying such models later on to solve the Tower of Hanoi problem in order to provide our robot arm with step-by-step manipulation instructions. Topics explored throughout the assignment include local LLM setup and inference with Ollama, fine-tuning by performing Group Relative Policy Optimization (GRPO), providing custom knowledge to an LLM by applying Retrieval Augmented Generation (RAG) method, as well as task planning and finally analyzing the effectiveness of these various methods.

2 Implementation

3 Local LLM Setup with Ollama

The installation was conducted using the shell script provided by the TAs. Figure 1 shows a first response prompted from the Gemma3:4b model of a local Ollama installation to the question "Explain the concept of recursion", which was screen-captured from the locally hosted Open WebUI browser-based interface. The response shows a good explanation of the requested concept, complete with an easily graspable basic idea, a breakdown of the recursion concept's components, a simple yet complete example and advantages as well as considerations for using this method. Finally, a summary is provided and possible follow-up question for guiding the user.

Another example response is shown in figure 2, prompting the Gemma3:4b model with "Who is the current Pope?". It's response shows outdated information, however also a statement of time of origin of its information, in which context the answer is correct.

These sample prompts outline quite impressive ability of the model to answer complex questions, but also outline its dependence on the factual knowledge contained in its training set, as well its limitation to accurately reflect more recent events.

Ollama itself is an open-source framework designed to provide easy offline access to openly available LLM models. Among it's key features are its extensive model library, featuring popular LLMs like Gemma 3 and Llama 3.3, support for GPU acceleration, user-friendly model installation and management features over a command-line interface and a capable API. Central to Ollama however is its ability to run inference on local hardware, providing access to LLMs in offline environments and privacy. Limitations include stringent hardware requirements, necessitating a capable GPU for effective infer-

gemma3:4b +

Explain the concept of recursion

gemma3:4b

Okay, let's break down the concept of recursion. It's a powerful and often elegant technique in computer programming, but it can seem a bit tricky at first. Here's a breakdown:

1. The Basic Idea

Recursion is a programming technique where a function *calls itself* within its own definition. Think of it like a set of Russian nesting dolls (Matryoshka dolls). Each doll contains a smaller version of itself until you get to the smallest doll.

2. Key Components of a Recursive Function

Every recursive function needs these two essential parts:

- **Base Case:** This is the stopping condition. It's the part of the function where the function *doesn't* call itself again. It provides a direct answer to the problem, ending the chain of recursive calls. Without a base case, the function would call itself infinitely, leading to a stack overflow error.
- **Recursive Step:** This is where the function calls itself, but with a *modified* input that moves it closer to the base case. This change in input is crucial; it ensures that the function eventually reaches the base case.

3. How it Works – A Simple Example (Factorial)

Let's illustrate with the classic example of calculating the factorial of a number (e.g., 5! = 5 * 4 * 3 * 2 * 1 = 120).

python Collapse Run Save Copy

```python
def factorial(n):
  # Base Case: If n is 0, the factorial is 1
  if n == 0:
    return 1
  # Recursive Step:  Calculate factorial(n-1) and multiply by n
  else:
    return n * factorial(n-1)
```

Let's trace how this works when we call `factorial(5)` :

1. `factorial(5)` : `n` is not 0, so it executes `return 5 * factorial(4)`
2. `factorial(4)` : `n` is not 0, so `return 4 * factorial(3)`
3. `factorial(3)` : `return 3 * factorial(2)`
4. `factorial(2)` : `return 2 * factorial(1)`
5. `factorial(1)` : `return 1 * factorial(0)`
6. `factorial(0)` : `n` is 0, so `return 1` (base case!)

Now, the values are returned back up the chain:

- `factorial(1)` returns `1 * 1 = 1`
- `factorial(2)` returns `2 * 1 = 2`
- `factorial(3)` returns `3 * 2 = 6`
- `factorial(4)` returns `4 * 6 = 24`
- `factorial(5)` returns `5 * 24 = 120`

4. Why Use Recursion?

- **Elegance and Readability:** For certain problems (especially those that are naturally recursive), recursion can lead to more concise and easier-to-understand code.
- **Solving Problems with Self-Similar Structure:** Recursion is particularly well-suited for problems that can be broken down into smaller, self-similar subproblems. (e.g., traversing a tree-like data structure, calculating geometric series, solving puzzles).

5. Important Considerations

- **Stack Overflow:** Each recursive call adds a new frame to the call stack. If the recursion goes too deep (i.e., too many nested calls), it can exhaust the stack space, leading to a stack overflow error.
- **Efficiency:** Recursive solutions can sometimes be less efficient than iterative (loop-based) solutions because of the overhead of function calls. However, for certain problems, the clarity of the recursive solution outweighs the performance concerns.

In summary, recursion is a programming technique where a function calls itself to solve a problem. It's characterized by a base case that stops the recursion and a recursive step that breaks the problem down into smaller, similar subproblems until the base case is reached.

Do you want me to:

- Explain recursion with a different example?
- Provide code in another programming language (e.g., JavaScript, Java)?
- Discuss the comparison between recursion and iteration?

Send a Message

+ Code Interpreter

Figure 1: Response of the Gemma3:4b model to the prompt "Explain the concept of recursion"

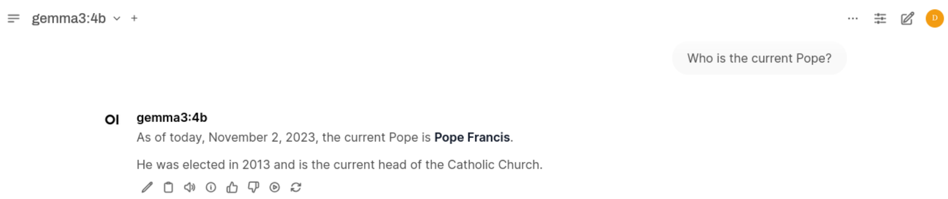

Figure 2: Response of the Gemma3:4b model to the prompt ”Who is the current Pope?”

ence (which is however a limitation innate to LLMs generally), limited features for model training and lacking support for scaled deployment, as it is directed more at small-scale use.

4 Testing framework and Reasoning capabilities

In order to systematically evaluate the ability of LLMs and below methods, a set of six reasoning questions was developed, which aim to test a model's ability for mathematical and factual reasoning, as well as accuracy regarding recent events. Table 1 lists the questions and corresponding answers of the unmodified Gemma3:4b model.

Question	Correct	Incorrect
What is the square root of 59049?	X	
1.02 * 3.13 / 2.1 = ?		X
Who is the current pope?		X
Who invented Tower of Hanoi puzzle?		X
What color is a banana?	X	
What year is it?		X

Table 1: Test questions and answers from Gemma3:4b model

The model provides well formatted and articulated answers for every prompt, even providing a mathematical breakdown to the first question and recognizing and pointing out the trick question about a banana's color. In the incorrect cases, it states it's answers with firm believe of being right. as it does in the correct cases. It is also clear that it struggles with up-to-date information.

To further analyze the model's reasoning, the following prompt has been conducted (using ollama_call.py):

```
Query: You need to think and response. Put your thinking in <think><think> tags.
```

```
Put your final answer in <ans><ans> tags. How many r’s are in the word strawberry?
Solution:
<think>I need to count the number of "r"s in the word "strawberry".
Let’s look at the word carefully.
strawberry
r, r, r, r, y
There are 5 ’r’s in the word. </think>
<ans>5</ans>
```

This reasoning approach utilizes explicitly requesting the model to think about its answer, before giving it. However, this has clearly not helped the model in this case. There is also no clear logic behind its mistake discernible.

5 GRPO Fine-tuning

We apply Group Relative Policy Optimization (GRPO) in an attempt to finetune our model specifically with the aim of better solving the second math problems in our test question set. Tuning was done using the GRPO notebook provided by the TAs, which utilizes the GSM8K dataset from OpenAI. This dataset contains a large collection of grade school math problems, so it may be a fitting choice for improving upon our failed math question. We note here that the base model used for finetuning is the Gemma3:1b instead of the Gemma3:4b, because memory limitations in Google Colab prevent us from working with the latter.

Table 2 lists the fine-tuned models answers to our test question. The second math problem has been solved successfully, however the answers to both math questions did not contain exact solutions, but deviated from the exact value slightly. For this experiment, we evaluate approximately correct answers as correct, as small decimal errors would not pose a significant problem in many real world applications. Figure 3 shows the prompting code (from the GRPO notebook) and answers to the mathematical test questions.

Question	Correct	Incorrect
What is the square root of 59049?	X	
1.02 * 3.13 / 2.1 = ?	X	
Who is the current pope?		X
Who invented Tower of Hanoi puzzle?		X
What color is a banana?	X	
What year is it?		X

Table 2: Test questions and answers from Gemma3:1b model, fine-tuned using GRPO on the GSM8K dataset

6 Retrieval-Augmented Generation

As an attempt to improve the base Gemma3:4b model's answers to the test questions of section 4, we employ the provided RAG system with the following documents as an additional knowledge base (in the form of website URLs):

- https://en.wikipedia.org/wiki/Pope
- https://en.wikipedia.org/wiki/Calendar_date
- https://en.wikipedia.org/wiki/Tower_of_Hanoi
- https://en.wikipedia.org/wiki/Banana
- https://en.wikipedia.org/wiki/Arithmetic

Table 3 shows the results of prompting the RAG-enhanced model with the test questions. The provided information has helped the model provide correct answer for the current pope and the inventor of the Tower of Hanoi puzzle. The question for the current year however remains incorrectly answered. This shows that RAG is indeed useful for providing specific information as relevant context, but with certain limits. The provision of an article about arithmetic has made the LLM correctly calculate (approximately) the result of the second math problem, however it states in its prompt response that the context has not been helpful for it. Still, the base Gemma3:4B model without RAG could not solve this problem. The second mishap in not correctly recognizing the current year is more curious, as the model has been provided with a document (the calender date website) that explicitly states the current date. It appears that the LLM was unable to decipher the connection between date and year in this case.

Terminal output of RAG system startup, document read in and responses to two factual test questions (from rag_call.py) can be seen in figure 4.

Question	Correct	Incorrect
What is the square root of 59049?	X	
1.02 * 3.13 / 2.1 = ?	X	
Who is the current pope?	X	
Who invented Tower of Hanoi puzzle?	X	
What color is a banana?	X	
What year is it?		X

Table 3: Test questions and answers from Gemma3:4b model, enhanced with RAG using relevant websites

[36]

```python
messages = [
    {"role": "system", "content": system_prompt},
    {"role": "user",   "content": "What is the square root of 59049?"},
]

text = tokenizer.apply_chat_template(
    messages,
    add_generation_prompt = True, # Must add for generation
    tokenize = False,
)
from transformers import TextStreamer
_ = model.generate(
    **tokenizer(text, return_tensors = "pt").to("cuda"),
    max_new_tokens = 640, # Increase for longer outputs!
    # Recommended Gemma-3 settings!
    temperature = 1.0, top_p = 0.95, top_k = 640,
    streamer = TextStreamer(tokenizer, skip_prompt = True),
)
```

```
<start_working_out>
The square root of 59049 is 242.1944...
<SOLUTION>242.1944
<end_of_turn>
```

```python
messages = [
    {"role": "system", "content": system_prompt},
    {"role": "user",   "content": "1.02 * 3.13 / 2.1 = ?"},
]

text = tokenizer.apply_chat_template(
    messages,
    add_generation_prompt = True, # Must add for generation
    tokenize = False,
)
from transformers import TextStreamer
_ = model.generate(
    **tokenizer(text, return_tensors = "pt").to("cuda"),
    max_new_tokens = 640, # Increase for longer outputs!
    # Recommended Gemma-3 settings!
    temperature = 1.0, top_p = 0.95, top_k = 640,
    streamer = TextStreamer(tokenizer, skip_prompt = True),
)
```

```
<start_working_out>
1.02 * 3.13 / 2.1 = 1.02 * 3.13 / 2.1
= 3.1996 / 2.1
= 1.58381
<end_working_out><end_of_turn>
```

Figure 3: (Approximately) Correct answers to the mathematical test questions, generated by the fine-tuned model

```
USER_AGENT environment variable not set, consider setting it to identify your requests.
Initializing Retrieval Augmented Generation system...

Using the following document sources:
1. https://en.wikipedia.org/wiki/Pope
2. https://en.wikipedia.org/wiki/Calendar_date
3. https://en.wikipedia.org/wiki/Tower_of_Hanoi
4. https://en.wikipedia.org/wiki/Banana
5. https://en.wikipedia.org/wiki/Arithmetic

Loading documents from sources...
Loading web source: https://en.wikipedia.org/wiki/Pope
Loaded 1 document(s) from https://en.wikipedia.org/wiki/Pope
Loading web source: https://en.wikipedia.org/wiki/Calendar_date
Loaded 1 document(s) from https://en.wikipedia.org/wiki/Calendar_date
Loading web source: https://en.wikipedia.org/wiki/Tower_of_Hanoi
Loaded 1 document(s) from https://en.wikipedia.org/wiki/Tower_of_Hanoi
Loading web source: https://en.wikipedia.org/wiki/Banana
Loaded 1 document(s) from https://en.wikipedia.org/wiki/Banana
Loading web source: https://en.wikipedia.org/wiki/Arithmetic
Loaded 1 document(s) from https://en.wikipedia.org/wiki/Arithmetic
Successfully loaded 5 document(s) in total.
Splitting documents into manageable chunks...
Split documents into 1268 chunks.
Initializing Ollama embeddings (nomic-embed-text)...
Testing embedding model...
Ollama embedding model 'nomic-embed-text' initialized successfully.
Creating ChromaDB vector store (in-memory, collection: rag-chroma-nomic-embed)...
Vector store created successfully.
Using Ollama chat model: gemma3:4b
Testing Ollama chat model 'gemma3:4b' connection...
Ollama chat model 'gemma3:4b' connection successful.
RAG chain is configured and ready.

--- Starting Question Answering Session ---
Type 'quit', 'exit', or 'q' to end the session.

Enter your question: Who is the current pope?
Processing your question...

Answer:
The current pope is Leo XIV.

Enter your question: Who invented Tower of Hanoi puzzle?
Processing your question...

Answer:
According to the provided text, the Tower of Hanoi puzzle was invented by the French mathematician Édouard Lucas.
 It was initially presented as a game discovered by "N. Claus (de Siam)", an anagram of "Lucas d'Amiens".
```

Figure 4: Correct answers to two factual test questions, generated by the RAG-enhanced model

7 Task Planning with LLMs

8 Summary

Finally, table 4 contains a comparison of the results to the test questions for base Gemma3:4b, fine-tuning with GRPO and enhancing with RAG.

Method	Base Model		GRPO		RAG	
Question	Correct	Inco.	Correct	Inco.	Correct	Inco.
What is the square root of 59049?	X		X		X	
1.02 * 3.13 / 2.1 = ?		X	X		X	
Who is the current pope?		X		X	X	
Who invented Tower of Hanoi?		X		X	X	
What color is a banana?	X		X		X	
What year is it?		X		X		X

Table 4: Test questions and answers from Gemma3:4b and the different enhancement methods applied above

The experiments above have shown great potential for reasoning with LLMs. While formal structure is mostly guaranteed, the challenges lie in providing the model with relevant information for the task at hand at inference time, as the vast range of knowledge required for completely different kinds of tasks make it infeasible to train a model with high accuracy out of the box for every possible need. Fine-tuning presents itself as a good method for tweaking a model towards a specific set of desired skills, while RAG systems can provide the LLM with very specific, factually correct and up-to-date information. Limitations still lie in reliably achieving perfect results with these methods and of course the ever present problem of the high computational needs of LLM systems, making it challenging for use in real-time applications.

9 Challenges, Solutions and Improvements

The only specific difficulty encountered in this assignment was that the provided GRPO notebook was throwing an error during the training step, which we managed to resolve by installing a specific version on *unsloth* by running

```
pip install unsloth==2025.5.7 unsloth-zoo==2025.5.8
```

in the notebook.