# **1RT730** LLMs and Societal Consequences of Artificial Intelligence - Report for Hand-in Assignment 1

Aditya Khadkikar — adkh8153@student.uu.se

September 24, 2025

## 1 Introduction

For this assignment, Google's Gemini Large Language Model (LLM) was used, specifically, the `gemini-2.5-flash` model.

## 2 Classification Accuracy

Assess if the model can correctly classify mushrooms? Note: the model cannot classify the parasol mushroom as the stipe is missing. What happens in this case?

# 3 Prediction Consistency

The model is likely not consistent in its predictions. One way is to change the temperature of the model and set it to 0.0. What other causes could lead to inconsistent predictions?

# 4 Topic Control

A method to make the model talk only about the topic of mushrooms, it might need to be controlled with its temperature. With sufficient pre-prompting as another method, it needs to be reminded to stick to the topic of mushroom classification, and avoid any other questions that are outside of the topic. Extending to that, the model can be given examples of when to negate answering an off-topic question, and when to detect whether a question or statement by the user is adhering to the topic of mushrooms. This can look like:

{
User: What is 5 + 2?
LLM: I am sorry, but I am not supposed to answer mathematics-related questions. I am an assistant chatbot that answers your questions about mushrooms. Feel free to ask me anything about them!
}

{
User: Who was the best player in the 2015 Football Match of Arsenal v.s. West Ham in the Premier League?
LLM: I am sorry, but I am not supposed to answer football-related questions. I am an assistant chatbot that answers your questions about mushrooms.
}

{
User: I am having Python installation problems, why is my numpy dependency not being loaded correctly?
LLM: I am sorry, but I am not supposed to answer Python, or coding-related questions. I am an assistant chatbot that answers your questions about mushrooms. Feel free to ask me anything about them!
}

# 5 Transcription Quality

Can you ask the chatbot to transcribe the text in the `nya_svampboken_p226.jpg` file? Did it do a good job? Are things reinterpreted or missing?

# 6 Safety Filters

- (Gemini) To transcribe the page from ‘Nya svampboken‘, you may have had to lower the safety of the model. What are the risks and consequences of doing so?
- (HuggingFace) HF models don’t come with safety filters. What are the risks and consequences of that? How would you approach this problem in a risky scenario?

# 7 JSON Output Assessment

Does the chatbot provide accurate descriptions given the instructions? Is the JSON format correct? Fields correctly filled? Answers relevant? Summary corresponds to JSON response?

# 8 Answer Quality Check

Can you check the quality of the chatbot's answers? How? Try `mushroom_*.jpg` images with various models of various complexities. What are the results? What are the limitations of a mushroom expert chatbot?

# 9 Dangerous Knowledge Handling

The Amanita Muscaria contains a neurotoxin but is not deadly. It can be eaten if prepared correctly. Can you make the chatbot provide this information? How can you make sure it does not provide it to someone who might eat the mushroom without proper preparation?

## 10 (Optional) Engagement

Can you find a way to make the chatbot more engaging? How would you do that?

## 11 Usage of Generative AI

To answer the questions part of this hand-in assignment, Generative AI assistant tools were not used.

# A Code

Here you can insert your code. Otherwise, you can submit a zip file with this report and your code.

```
"""A mushroom expert chatbot that responds to user queries about mushrooms."""
import gradio as gr, random
from gradio import ChatMessage
import base64

def image_to_base64(image_path):
    with open(image_path, 'rb') as img:
        encoded_string = base64.b64encode(img.read())
    return encoded_string.decode('utf-8')

def response(message, history):
    if len(message["files"]) != 0:
        return "A file seems to have been uploaded."
    if "-test-" in message["text"].lower():
        return "The interface is up. However, needs to be connected to some strong LLM."
    if "hello" in message["text"].lower():
        return "Hello! I am a mushroom expert. Ask me anything about mushrooms."
    elif "bye" in message["text"].lower():
        return "Goodbye! Have a great day."
    else:
        return random.choice([
            "I am not sure about that. Can you ask me something else?",
            "Could you reformulate your question? I am not sure I understand.",
            "I don't understand, can you ask someone else?",
            "What a stellar question! I am not sure about the answer though.",
            "You know what? I am not cut out for this. I am going to take a break."
        ])

########################################
# TODO: Add the Gemini (or HuggingFace) chatbot to the interface.
from typing import Iterator
from dotenv import load_dotenv
import os, google.generativeai as genai

load_dotenv()

API_KEY = str(os.getenv("GEMINI_API_KEY"))

genai.configure(api_key=API_KEY)
model = genai.GenerativeModel("gemini-2.5-flash")

def stream_gemini_response(user_message: str, messages: list) -> Iterator[list]:
    """ Streams both thoughts and responses from the Gemini model. """
    # Initialize response from Gemini
    response = model.generate_content(user_message, stream=True)

    # Initialize buffers
    thought_buffer = ""
    response_buffer = ""
    thinking_complete = False

    # Add initial thinking message
    messages.append(
        ChatMessage(
            role="assistant",
            content="",
            metadata={"title": "Thinking: *The thoughts produced by the Gemini 2.0 Flash model
                are experimental*"}
        )
```

```
    )

    for chunk in response:
        parts = chunk.candidates[0].content.parts
        current_chunk = parts[0].text
        if len(parts) == 2 and not thinking_complete:
            # Complete thought and start response
            thought_buffer += current_chunk
            messages[-1] = ChatMessage(
                role="assistant",
                content=thought_buffer,
                metadata={"title": "Thinking: *The thoughts produced by the Gemini2.0 Flash model
                     are experimental"}
            )
            # Add response message
            messages.append(
                ChatMessage(
                    role="assistant",
                    content=parts[1].text
                )
            )
            thinking_complete = True
        elif thinking_complete:
            # Continue streaming response
            response_buffer += current_chunk
            messages[-1] = ChatMessage(
                role="assistant",
                content=response_buffer
            )
        else:
            # Continue streaming thoughts
            thought_buffer += current_chunk
            messages[-1] = ChatMessage(
                role="assistant",
                content=thought_buffer,
                metadata={"title": "Thinking: *The thoughts produced by the Gemini2.0 Flash model
                     are experimental"}
            )

        yield messages

########################################
with gr.Blocks(fill_height=True) as demo:
    # TODO: ADD A SYSTEM PROMPT + HISTORY

    # TODO: -->
    # system_prompt = gr.TextBox("You are a mushroom expert chatbot that responds to user
         queries about mushrooms.", label="System Prompt")

    chatbot = gr.Chatbot(
        #fn=response,
        type="messages",
        # outputs=["text"],
        # multimodal=True,
        label="Your Personal Mushroom Expert"
    )

    image = gr.Image(type="filepath")

    query_box = gr.Textbox(
        placeholder="Type your message here and press Enter..."
    )
```

```
    def query_message(history,txt,img):
        if not img:
            history += [(txt,None)]
            return history
        base64 = image_to_base64(img)
        data_url = f"{base64}"
        history += [(f"{txt} ![]({data_url})", None)]

    send_button = gr.Button("Send")
    send_req = send_button.click(stream_gemini_response, [image, chatbot], [chatbot])
########################################

if __name__ == "__main__":
    demo.launch()
```

# B Chat Examples

Insert transcripts or screenshots of chatbot responses.