Exercise Sheet 3

Exercise 1

Exercise 1c)

BPMN model "Example" serialised in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions…
>
  <bpmn:process id=… >
    <bpmn:startEvent id="StartEvent_1">
      <bpmn:outgoing>Flow_1</bpmn:outgoing>
    </bpmn:startEvent>
      …
    <bpmn:sequenceFlow id="Flow_1" sourceRef="StartEvent_1" targetRef="Activity_1" />
  …
  </bpmn:process>
```

Exercise 1c)

Corresponding textual process description for the BPMN model "Example":

The process “…” begins with…

Exercise 1c)

BPMN model "Example" serialised in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions…
>
  <bpmn:process id=… >
  …
  </bpmn:process>
```

Corresponding textual process description for the BPMN model "Example":

The process “…” begins with…

Exercise 1d)

```
from haystack.document_stores.in_memory import InMemoryDocumentStore
document_store = InMemoryDocumentStore()

from haystack.components.converters import TextFileToDocument
converter = TextFileToDocument()
docs = converter.run(sources=[filepath])["documents"]
```

Exercise 2

2a-1. ChatPromptBuilder – Example Prompt

```
system_message = ChatMessage.from_system("You are an expert Q&A system with expert
knowledge on the business process modeling language BPMN. Consider the standard BPMN
specification. Assume you have created a BPMN model. Now you want to explain the complete
control flow with all interactions between participants and lanes represented by the BPMN
notations used in your created BPMN model to users without knowledge on BPMN notation.")



user_message = ChatMessage.from_user("""

Please create a textual process description for the given BPMN model serialised in XML.
Add to each type of BPMN element used in the BPMN model a short explanation of this
element type's semantics.

Exemplary textual process description for BPMN model "Example": {{ documents[0].content }}

BPMN model "Explain" serialised in XML: {{ query }}

""")


messages = [system_message, user_message]
```

2a-2. PromptBuilder – Example Prompt

```
template ="""
Please create a textual process description for the given BPMN model
serialised in XML. Add to each type of BPMN element used in the BPMN model
a short explanation of this element type's semantics.


Exemplary textual process description for BPMN model "Example":
{{ documents[0].content }}
BPMN model "Explain" serialised in XML: {{ query }}
"""
```

2a-3. PromptBuilder – Example Prompt

```
template ="""

Please create a textual process description for the given BPMN model
serialised in XML. Add to each type of BPMN element used in the BPMN model
a short explanation of this element type's semantics.

BPMN model "Explain" serialised in XML: {{ query }}

"""
```

2b-1. Chat-Generator

```
from haystack_integrations.components.generators.ollama import
OllamaChatGenerator

generator = OllamaChatGenerator(model="llama3.1:8b",
                            url = "http://localhost:11434",
                            timeout = 30*60,
                            generation_kwargs={
                              "num_ctx": 4096,
                              "temperature": 0.9,
                              })
```

2b-2. Generator

```
from haystack_integrations.components.generators.ollama import
OllamaGenerator

generator = OllamaGenerator(model="llama3.1:8b",
                            url = "http://localhost:11434",
                            timeout = 30*60,
                            generation_kwargs={
                              "num_ctx": 4096,
                              "temperature": 0.9,
                              })
```

2b-3. Generator

```
from haystack_integrations.components.generators.ollama import
OllamaGenerator

generator = OllamaGenerator(model="llama3.1:8b",
                            url = "http://localhost:11434",
                            timeout = 30*60,
                            generation_kwargs={
                              "num_ctx": 4096,
                              "temperature": 0.9,
                              })
```

2c – Custom Component

```
from haystack import component

from typing import Any, Dict, List, Optional, Set

@component

class Collector:

    @component.output_types(chat_rag_out=str, rag_out=str, no_rag_out=str)

    def run(self, chat_rag_out: List[ChatMessage], rag_out: List[str], 
no_rag_out: List[str]):

        return {"chat_rag_out": chat_rag_out[0].text,
               “rag_out": rag_out[0],
               "no_rag_out": no_rag_out[0]}


collector = Collector()
```

Exercise 3

3a)

```
import re

def clean_bpmn(BPMN_string):
    """Removes the content between `<bpmndi:BPMNDiagram>` and
`</bpmndi:BPMNDiagram>` tags from a string.
        """

    pattern = r'<bpmndi:BPMNDiagram.*?</bpmndi:BPMNDiagram>'
    new_string = re.sub(pattern, ‚‘, BPMN_string, flags=re.DOTALL)
    return new_string
```

3b) - Pipeline

```
from haystack import Pipeline
rag_pipeline = Pipeline()

rag_pipeline.add_component("text_embedder", text_embedder)
rag_pipeline.add_component("retriever", retriever)
rag_pipeline.add_component("chat_prompt_builder", chat_prompt_builder)
rag_pipeline.add_component("prompt_builder", prompt_builder)
rag_pipeline.add_component("prompt_builder_no_rag", prompt_builder_no_rag)
rag_pipeline.add_component("chat_llm", chat_generator)
rag_pipeline.add_component("llm", generator)
rag_pipeline.add_component("no_rag", generator_no_rag)
rag_pipeline.add_component("collector", collector)
```

3b) - Pipeline

```
rag_pipeline.connect("text_embedder.embedding", "retriever.query_embedding")
rag_pipeline.connect("retriever", "chat_prompt_builder.documents")
rag_pipeline.connect("retriever", "prompt_builder.documents")
rag_pipeline.connect("chat_prompt_builder.prompt", "chat_llm.messages")
rag_pipeline.connect("prompt_builder.prompt", "llm.prompt")
rag_pipeline.connect("prompt_builder_no_rag.prompt", "no_rag.prompt")
rag_pipeline.connect("llm.replies", "collector.rag_out")
rag_pipeline.connect("chat_llm.replies", "collector.chat_rag_out")
rag_pipeline.connect("no_rag.replies", "collector.no_rag_out")
```

New Content

Influence of Model Size

- Parameters are internal variables
- Number of parameters is influenced by many factors
  - Model structure
  - Number of layers
  - ...
- More Parameters does not mean higher accuracy
- More parameters can help with understanding more complex data
- For visualization:
  - https://bbycroft.net/llm
  - https://poloclub.github.io/transformer-explainer/

Other sources:
https://arxiv.org/pdf/2407.14088
https://ourworldindata.org/grapher/artificial-intelligence-parameter-count

Structured Output

- Allows LLMs to return data in a specific, predictable format
- Instead of natural language, you get structured data in the form of JSON objects, Pydantic models, or other dataclasses
- In Haystack, we can use an Output Validator to Reprompt, if the output is not in our desired datastructure.
- In my tests, Llama3.2 performed better than Llama3.1 when outputting JSON objects.