**به نام خدا**

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

# درس پردازش زبان طبیعی

## پاسخ تمرین شماره شش

نام و نام خانوادگی: امیرحسین شاه قلی

شماره دانشجویی: ۸۱۰۱۹۹۴۴۱

**تیر ماه ۱۴۰۳**

# Table of Contents

## Answer of the question one

### part one: Receiving and Preparing Data

**RecursiveCharacterTextSplitter:** The RecursiveCharacterTextSplitter with a chunk_size of 1024 and a chunk_overlap of 64 is a hypothetical tool used to break down a long text string into smaller parts or chunks. Here's a breakdown of how it generally operates and why this is useful.

**How It Works:**

- **Chunk Size**: The main parameter here is the chunk_size which is set to 1024 characters. This means that the splitter will attempt to divide the text into segments, each containing up to 1024 characters.
- **Chunk Overlap:** The chunk_overlap parameter is set to 64 characters. This means that each chunk will share 64 characters with the next chunk. For instance, if the first chunk ends with the sentence "Hello world", the last 64 characters of this chunk will be repeated as the first 64 characters of the next chunk.
- **Recursive Process:** The term Recursive in RecursiveCharacterTextSplitter suggests that this process can handle situations where even the 1024-character chunks are too large and need further splitting. It will continuously split the chunks into smaller pieces until they meet another condition or until they can be managed.

**Why This Is Needed:**

- **Handling Large Texts:** Large texts are difficult to process in one go due to memory limitations or because the processing power required would be prohibitive. Splitting the text into manageable parts allows for more efficient processing.

- **Context Retention:** By overlapping chunks of text (64 characters in this case), the context or meaning around the border of each chunk is better preserved. This is especially important in applications like natural language processing where the meaning of text can dramatically alter if parts of sentences are treated in isolation.

- **Recursive Solutions:** For exceedingly large texts, even the initial split might not be enough. Recursive splitting ensures that all portions of the text reach a size that can be effectively processed, while still preserving the necessary context around breaks.

## Importance of chunk_size and overlap_chunk:

The selection of appropriate values for chunk_size and overlap_chunk is crucial when using text splitting tools. These values must be chosen to align with the purpose of the text processing and any limitations of the system.

**Importance of chunk_size:**

- **Memory and Resource Management:** If chunk_size is too large, it can lead to memory issues because larger chunks consume more resources.
- **Processing Speed:** Very large chunks might slow down the processing speed because more time is needed to process each chunk.
- **Processing Accuracy:** In cases like natural language processing, very small chunks might lead to a loss of sufficient context necessary for accurately understanding meanings.

**Importance of overlap_chunk:**

- **Context Retention:** Adequate overlap helps preserve context so important information at the edges of chunks isn't lost.

- **Model Accuracy:** In complex processing such as NLP, inadequate overlap can decrease model accuracy, as models might not fully comprehend the complete sense of sentences.

**Problems from Choosing Inappropriate Values:**

- **Memory Shortage or Runtime Errors:** A very large chunk_size can lead to excessive use of memory, resulting in runtime errors.
- **Loss of Data or Context:** If overlap_chunk is too small, critical information or context between chunks could be lost.
- **Increased Processing Time:** Setting the overlap_chunk too large unnecessarily can lead to excessive data duplication and slow down the processing.
- **Resource Exhaustion:** Very small chunks with high redundancy and dispersal can lead to resource exhaustion and delays in execution.

## PART TWO: EMBEDDING

The use of an appropriate embedder is critical for the successful representation and processing of textual data, especially in complex computational tasks such as natural language processing (NLP). An embedder typically refers to models that convert text into numerical vectors or embeddings that capture semantic meaning, context, and syntactic elements in a form that machines can understand and process.

### Importance of Using the Right Embedder:

- **Semantic Understanding:** Appropriate embeddings help in capturing the correct semantic meaning within the text. This is fundamental in applications like sentiment analysis, machine translation, or topic extraction.
- **Performance:** The accuracy and reliability of NLP tasks heavily depend on the quality and suitability of the embeddings. The better the embeddings align with the specific characteristics of the text, the higher the performance of subsequent tasks.

- **Language Specificity:** Each language has unique syntactical, grammatical, and usage nuances. Embedders trained in specific languages are more adept at understanding these nuances than those trained in different languages.

### Problems Using Inappropriate Embedder for Persian Sentences:

If you use an embedder for Persian sentences that was generally not trained on Persian data (trained only on English text), several issues could arise:

- **Loss of Context and Meaning:** Persian has structural and linguistic characteristics distinct from those of English. An embedder not trained in Persian will likely fail to capture these nuances, leading to a loss of accuracy in understanding and representing sentence meaning.
- **Mishandled Idioms and Cultural Expressions:** Every language has idioms and expressions that are culturally specific. An embedder unfamiliar with Persian might not properly interpret these, leading to significant errors in tasks like translation or sentiment analysis.
- **Poor Handling of Grammar and Syntax:** Persian grammar and syntax (such as the use of suffixes, verb placement, etc.) differ from those in English. An inadequately trained embedder might misrepresent these elements, affecting the grammatical integrity of generated or interpreted text.
- **Reduced Model Effectiveness:** Overall, using a non-Persian trained model will generally result in poorer performance across all tasks involving the understanding or generating of Persian text. This can manifest as lower accuracy in classification tasks, poorer user experience in conversational AI, errors in information extraction, etc.

## PART THREE: SEMANTIC RETRIEVAL IMPLEMENTATION

**Question 1: "What is Markov chain?"**

- **Content:** This is a topic that is directly covered in the seen documents.
- **Results:** The documents retrieved are highly relevant. They provide a detailed explanation of Markov chains, discussing the concept itself, the Markov assumption, and related models like the Hidden Markov Model (HMM). This indicates that the retriever effectively locates and pulls out the pertinent section from its document collection when the query matches directly with the content it has already encountered.
- **Evaluation:** This is the optimal performance of the retriever, as it successfully identifies and returns information directly related to the query from the seen documents.

**Question 2: "What is Binary search tree (BST)?"**

- **Content:** This query pertains to a subject (data structures and algorithms) that is similar to the seen content (as it relates to computer science) but hasn't been specifically seen in the documents.
- **Results:** The returned documents don't directly discuss Binary Search Trees but instead include related topics on data structures and parsing trees. The retriever seems to choose documents based on related keywords or concepts in computing and data processing.
- **Evaluation:** Here, the retriever manages, to a certain extent, to grasp the general area or domain of the query but fails to precisely match the specific concept of a Binary Search Tree. It shows that while the retriever is good at sticking to the broader domain, it may struggle with specificity if the exact topic hasn't been covered in the training or seen in documents.

**Question 3: "Who is the president of Bolivia?"**

- **Content:** This query is entirely outside the scope of the seen documents, which are technical and focus on topics related to computing and linguistics.
- **Results:** The returned documents are unrelated to the query and include general considerations of discourse models and language interpretation. Since the topic of "president of Bolivia" is not related to the documents' content, the retriever retrieves documents based generally on processing language, proving its incapacity in this case to understand or produce useful information pertaining to political figures or current events.
- **Evaluation:** This showcases a limitation of the retriever when faced with queries completely outside its training domain or seen documents. The retriever is unable to identify the lack of relevant information and instead returns the best match it can find, even if unrelated.

## PART FOUR: IMPLEMENT CHAIN ROUTER

**Understanding Temperature:**

- High Temperature (> 1): Increases the randomness of predictions. With a high temperature, the model is more likely to generate diverse and sometimes more creative or unpredictable responses. This is because the probability distribution becomes flatter, allowing less likely words to be picked more frequently.
- Low Temperature (< 1 but > 0): Reduces randomness and can lead to more predictable or safe responses. A low temperature sharpens the probability distribution, meaning the model increasingly favors words that it calculates as more likely.
- Temperature = 0: This setting is a special case. When the temperature is set to zero, it essentially turns the generation process into a deterministic one, where the model always picks the most likely next word or token given the previous context. This mode is often referred to as "argmax" sampling.

**Why Set Temperature to Zero?**

- **Consistency and Repeatability:** The responses from the model become completely deterministic. Given the same prompt and model state, it will always return the same response. This can be very useful in scenarios where predictable, reliable outputs are needed.
- **Maximize Confidence:** It prioritizes the highest probability outputs according to the model's training, which might be desirable in applications where deviations from expected responses can cause confusion or errors (customer service bots, informational queries, etc.).
- **Simplicity and Debugging:** Easier to debug or understand model behavior when testing, as there is no variability in the model's output to account for during analysis.

However, while setting the temperature to zero increases predictability, it also removes much of the nuanced or varied expression that higher temperatures can offer. This can sometimes result in responses that feel less natural or engaging to human users, especially in conversational interfaces where some level of spontaneity or adaptation may be more engaging. setting the temperature to zero in `ChatTogether` using `llama-3-70b-chat-hf` aligns with achieving deterministic, consistent, and high-confidence outputs. It is a choice that needs to be balanced against the need for creativity and engagement, depending on the application's specific requirements.

PART FIVE: IMPLEMENT SEARCH ENGINE CHAIN

```
from langchain_core.documents import Document
tool = TavilySearchResults()
def create_docs(results: list):
 documents = []
 for result in results[:5]:
   documents.append(Document(
         page_content=result['content'],
         metadata={"source": result['url']}
     )
   )
 return documents
search_engine_chain = tool | create_docs
prompt = "What is red-black tree?"
documents = search_engine_chain.invoke({"query": prompt})
print(documents)
```

## PART SEVEN: IMPLEMENT FALLBACK CHAIN

```
router_prompt_template = (
   "You are an expert in routing user queries to the most relevant category
based on the content of the query.\n"
   "Your expertise allows you to choose between 'Natural Language Processing
(NLP)' and 'Computer Science (CS)' or no category at all.\n"
   "If the given query is specifically about topics within Natural Language
Processing, choose VectorStore.\n"
   "If the query pertains to topics in Computer Science that are not related to
Natural Language Processing, choose SearchEngine.\n"
   "If the query is not related to these specified fields, return 'None'.\n"
   "Remember, give me only the name of the category you chose or 'None' if none
applies. Provide only this information and nothing more.\n"
   "{output_instructions}"
   "query: {query}"
)
prompt = ChatPromptTemplate.from_template(
   template=router_prompt_template,
)
from typing import Literal
class ChosenTool(BaseModel):
   tool_name: Literal['None', "VectorStore", "SearchEngine"] =
Field(description="the tool that was chosen by LLM in question routing stage")
question_router_parser = PydanticOutputParser(pydantic_object=ChosenTool)
question_router_parser.get_format_instructions()
chain_router = prompt | llm | question_router_parser
```

## PART EIGHT: GENERATE WITH CONTEXT CHAIN

```
generate_with_context_template = (
   "Please respond to the query below based solely on the context provided. If
the context given is irrelevant or does not directly relate to the query,
refrain from using your own knowledge to provide an answer.\n\n"
   "context: {context}\n\n"
   "query: {query}"
)
generate_with_context_prompt =
ChatPromptTemplate.from_template(generate_with_context_template)
generate_with_context_chain = generate_with_context_prompt | llm |
StrOutputParser()
query = "What is HMM?"
context = retriever.get_relevant_documents(query)
```

```
response = generate_with_context_chain.invoke({"query": query, "context":
context})
Markdown(response)
```

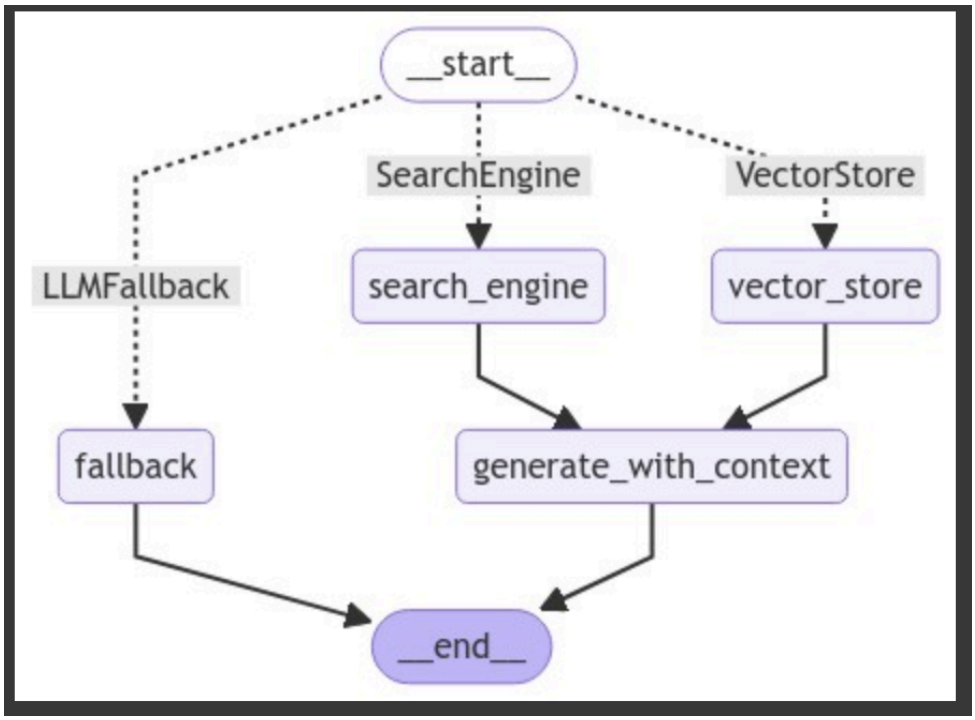
## PART NINE: GRAPH PREPARATION USING LANGGRAPH

```
class AgentSate(TypedDict):
   """The dictionary keeps track of the data required by the various nodes in
the graph"""
   query: str
   chat_history:list[BaseMessage]
   generation: str
   documents: list[Document]
def retrieve_node(state: dict):
   """
   Retrieve relevant documents from the vectorstore

   query: str

   return list[Document]
   """
   query = state["query"]
   documents = retriever.invoke(input=query)
   return {"documents": documents}
def fallback_node(state: dict):
   """
   Fallback to this node when there is no tool call
   """
   query = state["query"]
   chat_history = state["chat_history"]
   generation = fallback_chain.invoke({"query": query, "chat_history":
chat_history})
   return {"generation": generation}
def generate_with_context_node(state: dict):
   query = state["query"]
   documents = state["documents"]
   generation = generate_with_context_chain.invoke({"query": query, "context":
documents})
   return {"generation": generation}
def search_engine_node(state: dict):
   query = state["query"]
   documents = search_engine_chain.invoke({"query": query})
   return {"documents": documents}
```

```
def question_router_node(state: dict):
    query = state["query"]
    try:
        response = chain_router.invoke({"query": query,
                                        "output_instructions":
question_router_parser.get_format_instructions()
        })
    except Exception:
        print("Exception in getting response")
        return "LLMFallback"
    try:
      chosen_tool = response.tool_name.lower()
    except Exception:
        return "LLMFallback"
    if chosen_tool == 'none':
        print("---No tool called---")
        return "LLMFallback"
    if chosen_tool == "vectorstore":
        print("---Routing to VectorStore---")
        return "VectorStore"
    if chosen_tool == "searchengine":
        print("---Routing to SearchEngine---")
        return "SearchEngine"
    return "LLMFallback"
```



## PART TEN: SOME SAMPLE RESULTS

**Q:** `Who is the president of Iran?`

---No tool called—

**A:** I'm happy to help! However, I must acknowledge that my expertise is limited to Computer Science, and I'm not knowledgeable about political figures or current events. I can only provide information and answer questions related to Computer Science. If you have a question about algorithms, data structures, programming languages, or any other CS-related topic, I'd be happy to help!

**Q:** `What is NLP?`

---Routing to VectorStore—

**A:** Based on the provided context, NLP is not explicitly defined. However, it can be inferred that NLP stands for Natural Language Processing, as it is mentioned in the context of various applications such as machine translation, information retrieval, question answering, dialogue systems, and speech recognition.

**Q:** `What is Binary Search Tree?`

---Routing to SearchEngine—

**A:** A Binary Search Tree is a data structure used in computer science for organizing and storing data in a sorted manner. Each node in a Binary Search Tree has at most two children, a left child and a right child, with the left child containing values less than the parent node and the right child containing values greater than the parent node.