KDnuggets | Visit KDnuggets.com for more cheatsheets and additional learning resources.

LangChain Cheat Sheet

LangChain simplifies building applications with language models through reusable components and pre-built chains. It makes models data-aware and agentic for more dynamic interactions. The modular architecture supports rapid development and customization.

LLMs

An interface for OpenAI GPT-3.5-turbo LLM

```
from langchain.llms import OpenAI
llm = OpenAI(temperature=0.9)
text = "What do you know about KDnuggets?"
llm(text)
```

```
>>> KDnuggets is one of the most popular data science
websites which focusses....
```

An interface for HugginFace LLM

```
from langchain import HuggingFaceHub
llm = HuggingFaceHub(repo_id="togethercomputer/LLaMA-
2-7B-32K", model_kwargs={"temperature":0,
"max_length":64})

llm("How old is KDnuggets?")
```

```
>>> KDnuggets was founded in 1997, making it 23 years old.
```

Prompt Templates

LangChain facilitates prompt management and optimization through the use of prompt templates.

```
from langchain import PromptTemplate
template = """Question: {question}
Make the answer more engaging by incorporating puns.
Answer: """

prompt = PromptTemplate.from_template(template)

llm(prompt.format(question="Could you provide some
information on the impact of global warming?"))
```

```
>>> Global warming is no laughing matter, but that doesn....
```

Chains

Combining LLMs and prompt template can enhance multi-step workflows.

```
from langchain import LLMChain
llm_chain = LLMChain(prompt=prompt, llm=llm)
question = "Could you provide some information on the
impact of global warming?"

llm_chain.run(question)
```

```
>>> Global warming is no laughing matter—but it sure is.....
```

Agents and Tools

Tool refers to a function that performs a specific task, such as Google Search, database lookup, or Python REPL. Agents use LLMs to choose a sequence of actions to execute.

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent

tools = load_tools(["wikipedia", "llm-math"], llm=llm)
agent = initialize_agent(tools, llm, agent="zero-shot-react-
description", verbose=True)

agent.run("Can you tell me the distance between Earth and
the moon? And could you please convert it into miles?
Thank you.")
```

```
>>> Action: Wikipedia
Action Input: Earth-moon distance
Action: Calculator
Action Input: 385400/1.609
Final Answer: The distance between Earth and the Moon is
approximately 239,527.66 miles.
```

Memory

LangChain simplifies persistent state management in chain or agent calls with a standard interface

```
from langchain.chains import ConversationChain
from langchain.memory import
ConversationBufferMemory

conversation = ConversationChain(
    llm=llm, verbose=True,
memory=ConversationBufferMemory()
)

conversation.predict(input="How can one overcome
anxiety?")
```

```
>>> To overcome anxiety, it may be helpful to focus on the....
```

```
conversation.predict(input="Tell me more..")
```

```
>>> To be mindful of the present, it can be helpful to pra.....
```

Document Loaders

By combining language models with your own text data, you can answer personalized queries. You can load CSV, Markdown, PDF, and more.

```
from langchain.document_loaders import TextLoader

raw_document =
TextLoader("/work/data/Gregory.txt").load()
```

Vector Stores

One common method for storing and searching unstructured data is to embed it as vectors, then embed queries and retrieve the most similar vectors.

```
from langchain.embeddings.openai import
OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import FAISS

# Text Splitter
text_splitter = CharacterTextSplitter(chunk_size=1000,
chunk_overlap=0)
documents = text_splitter.split_documents(raw_document)

# Vector Store
db = FAISS.from_documents(documents,
OpenAIEmbeddings())

# Similarity Search
query = "When was Gregory born?"
docs = db.similarity_search(query)
print(docs[0].page_content)
```

```
>>> Gregory I. Piatetsky-Shapiro (born 7 April 1958) is a data
scientist and the co-founder of the KDD conferences.....
```

A retriever is an interface that returns documents based on an unstructured query. When combined with LLM, it generates a natural response instead of simply displaying the text from the document.

```
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(model_name="gpt-3.5-turbo",
temperature=0)
qa_chain =
RetrievalQA.from_chain_type(llm,retriever=db.as_retriever(
))
qa_chain({"query": "When was Gregory born?"})
```

```
>>> {'query': 'When was Gregory born?',
 'result': 'Gregory Piatetsky-Shapiro was born on April 7,
1958.'}
```

Subscribe to KDnuggets News

Abid Ali Awan | 2023