|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Робототехника и комплексная автоматизация (РК) |
| КАФЕДРА | Системы автоматизированного проектирования (РК6) |

**ОТЧЕТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ**

Студент Гунько Никита Макарович

Группа РК6-81Б

Тип практики Преддипломная

Название предприятия ООО «ЮБС»

|  |  |  |
| --- | --- | --- |
| Студент РК6-81Б А | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Гунько Н.М. В |
| (группа) | (Подпись, дата) | И.О. Фамилия |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Витюков Ф.А. В |
|  | (Подпись, дата) | И.О. Фамилия |

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2023 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой РК6

А.П. Карпенко

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

**ЗАДАНИЕ**

**на прохождение преддипломной практики**

Студент Гунько Никита Макарович, 4 курса, группы РК6-81Б в

(фамилия, имя, отчество, № курса, индекс группы)

в период с «15» мая 2023 г. по «28» мая 2023 г. В

Предприятие: ООО «ЮБС» а

Подразделение: отдел разработки а

Руководитель практики от предприятия:

Градинар Роман Игоревич, ведущий специалист отдела разработки в

(фамилия, имя, отчество, должность)

Руководитель практики от кафедры:

Витюков Фёдор Андреевич, преподаватель в

(фамилия, имя, отчество, должность)

Задание:

1. Реализация локального решения задачи построения банковского чат-бота с обученной языко-

вой моделью.

Дата выдачи задания «15» мая 2023 г.

|  |  |  |
| --- | --- | --- |
| **Руководитель практики от предприятия** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Градинар Р.И. В |
|  | (подпись, дата) | И.О. Фамилия |
| **Руководитель практики от кафедры** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Витюков Ф.А. В |
|  | (подпись, дата) | И.О. Фамилия |
| **Студент**  РК6-81Б А | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Гунько Н.М. В |
| (группа) | (подпись, дата) | И.О. Фамилия |

### ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4

ОСНОВНАЯ ЧАСТЬ 5

Теоретическая часть 5

Создание документа со знаниями 5

Алгоритм работы 6

Реализация 7

ЗАКЛЮЧЕНИЕ 8

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 9

ПРИЛОЖЕНИЕ 10

Приложение 1. Программная реализация загрузки и представления данных в векторном виде 10

Приложение 2. Программная реализация взаимодействия с загруженной языковой моделью 13

### ВВЕДЕНИЕ

В современном мире банки сталкиваются с растущей потребностью обеспечить клиентам мгновенный доступ к информации и услугам, особенно в сфере обслуживания через интернет. Однако, несмотря на все преимущества технологического прогресса, вопрос безопасности и конфиденциальности остается одной из основных проблем. Многие клиенты все еще опасаются использовать онлайн-сервисы из-за возможности утечки их личных данных или несанкционированного доступа к ним.

В этой работе представляется полностью локальная реализация банковского чат-бота, который позволяет клиентам задавать вопросы чат-боту, а самому виртуальному ассистенту работать без необходимости подключения к интернету. В решении использованы возможности LLM (Large Language Model) для создания интеллектуального ассистента, работающего исключительно внутри клиентской среды выполнения. Это означает, что все данные остаются строго конфиденциальными и никогда не покидают устройство пользователя.

Цель практики: поиск и реализация локального решения задачи построения банковского чат-бота с обученной языковой моделью.

Для выполнения поставленной цели необходимо решение следующих задач:

− найти подходящую архитектуру решения;

− найти готовую языковую модель.

### ОСНОВНАЯ ЧАСТЬ

Теоретическая часть

Для решения поставленного задания было решено использовать технологию генерации человеческой речи предобученной языковой моделью на основе встроенного контекста по причине того, что это не требует поиска готовой к дообучению языковой модели и траты времени на это дообучение. Реализация такой программы была разбита на две части: поиск языковой модели, способной генерировать естественную речь и модели эмбеддингов для получения наиболее релевантных по смыслу отрывков из векторной базы данных. Языковая модель была выбрана на основе технических характеристик рабочего компьютера. В качестве реализации программного решения был выбрал язык программирования Python.

Создание документа со знаниями

Для базы знаний был создан текстовый документ. Было собрано 470 строк банковских данных, взятых с сайта банка СберБанк. Все данные были переведены на английский язык из-за возникновения сложностей с работой на русском языке, так как происходит иное разбиение на токены. Небольшой отрывок получившегося документа показан на рисунке 1.

Изображение выглядит как текст, Шрифт, снимок экрана, документ

Автоматически созданное описание

Рисунок 1 – Часть подготовленных данных

Алгоритм работы

Алгоритм работы следующий:

1. Импорт библиотек: Программа начинает свою работу с импорта необходимых библиотек, таких как LangChain, Transformers и Chroma. Эти библиотеки обеспечивают функциональность анализа документов, создания вложений и работу с готовыми языковыми моделями.

2. Анализ документа: первый запуск программы начинается с анализа прикрепленных документов. Программа использует библиотеку LangChain для анализа входящих документов. LangChain обрабатывает текстовые данные и разделяет их на токены (слова, фразы и символы), выполняет поиск шаблонов и извлекает ключевые слова. Результаты анализа сохраняются в памяти для дальнейшей обработки.

3. Создание вложений: далее программа использует модель эмбеддингов для создания локальных вложений на основе обработанных текстовых данных.

4. Создание хранилища векторов: с использованием библиотеки Chroma программа создает локальное хранилище векторов, где сохраняются полученные вложения. Хранилище векторов служит в качестве базы данных, в которой векторные представления связаны с соответствующими контекстами из документов.

5. Загрузка языковой модели: программа загружает локально предварительно обученную языковую модель при помощи библиотеки Transformer. Эта модель будет использоваться для понимания вопросов пользователей и генерации ответов.

6. Поиск контекста: программа использует алгоритм для нахождения наиболее релевантного к запросу пользователя контекста из документов. Это позволяет языковой модели генерировать правдивый ответ на естественном языке.

7. Генерация ответа: на основе найденного контекста языковая модель генерирует ответ на заданный вопрос.

Реализация

Проект реализован в виде трех файлов и директории с документами, в которых хранятся данные. Первые два файла – это программы на языке программирования Python. В первом файле ingest.py реализована логика для подготовки векторного пространства для дальнейшего поиска в нем наиболее релевантной информации. Второй файл privateGPT.py отвечает за основую часть работы программы, а именно взаимодействие с загруженной языковой моделью. Третий файл называется .env. В нем собраны основные глобальные переменные, которые используются в описанных программах.

После запуска файла ingest.py создается директория database, содержащая локальный векторный индекс. Можно загрузить столько документов, сколько надо, и все они будут собраны в локальной базе данных вложений. Во время загрузки никакие данные не покидают локальную среду. Можно выполнять загрузку без подключения к Интернету, за исключением первого запуска сценария загрузки, когда загружается модель встраивания.

При запуске файла privateGPT.py будет показан этап активации языковой модели и выведена строка для ввода запросов.

### ЗАКЛЮЧЕНИЕ

В данной работе была представлена полностью локальная реализация банковского чат-бота без необходимости обращения в интернет. Используя мощность и возможности современных библиотек и инструментов, таких как LangChain, Transformers, Chroma и загруженных моделей, которые находятся в открытом доступе, был разработан проект чата с ботом, который позволяет клиентам задавать вопросы и получать на них правильные ответы, обеспечивая полную конфиденциальность данных.

В зависимости от системных характеристик рабочей станции можно использовать различные модели языкового моделирования, от размера которых будет зависеть качество формулирования естественной речи.

Результаты данной работы демонстрируют перспективы и возможности полностью локальной реализации банковского чат-бота. Этот подход открывает новые горизонты в сфере обслуживания клиентов, обеспечивая высокую степень безопасности. Одним из ключевых аспектов этой работы является применимость такого подхода не только в банковской сфере, но и в других отраслях.

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство по языку программирования Python [Электронный ресурс] // URL: https://metanit.com/python/tutorial/ (дата обращения: 25.05.2023);

2. Hugging Face. The AI community [Электронный ресурс] // URL: https://huggingface.co/ (дата обращения: 25.05.2023);

3. How to create a private ChatGPT with your own data? [Электронный ресурс] // URL: https://medium.com/@imicknl/how-to-create-a-private-chatgpt-with-your-own-data-15754e6378a1 (дата обращения: 25.05.2023).

### ПРИЛОЖЕНИЕ

Приложение 1. Программная реализация загрузки и представления данных в векторном виде

#!/usr/bin/env python3

import os

import glob

from typing import List

from dotenv import load\_dotenv

from multiprocessing import Pool

from tqdm import tqdm

from langchain.document\_loaders import (

CSVLoader,

EverNoteLoader,

PDFMinerLoader,

TextLoader,

UnstructuredEmailLoader,

UnstructuredEPubLoader,

UnstructuredHTMLLoader,

UnstructuredMarkdownLoader,

UnstructuredODTLoader,

UnstructuredPowerPointLoader,

UnstructuredWordDocumentLoader,

)

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.vectorstores import Chroma

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.docstore.document import Document

from constants import CHROMA\_SETTINGS

load\_dotenv()

# Load environment variables

persist\_directory = os.environ.get('PERSIST\_DIRECTORY')

source\_directory = os.environ.get('SOURCE\_DIRECTORY', 'source\_documents')

embeddings\_model\_name = os.environ.get('EMBEDDINGS\_MODEL\_NAME')

chunk\_size = 500

chunk\_overlap = 50

# Custom document loaders

class MyElmLoader(UnstructuredEmailLoader):

"""Wrapper to fallback to text/plain when default does not work"""

def load(self) -> List[Document]:

"""Wrapper adding fallback for elm without html"""

try:

try:

doc = UnstructuredEmailLoader.load(self)

except ValueError as e:

if 'text/html content not found in email' in str(e):

# Try plain text

self.unstructured\_kwargs["content\_source"]="text/plain"

doc = UnstructuredEmailLoader.load(self)

else:

raise

except Exception as e:

# Add file\_path to exception message

raise type(e)(f"{self.file\_path}: {e}") from e

return doc

# Map file extensions to document loaders and their arguments

LOADER\_MAPPING = {

".csv": (CSVLoader, {}),

# ".docx": (Docx2txtLoader, {}),

".doc": (UnstructuredWordDocumentLoader, {}),

".docx": (UnstructuredWordDocumentLoader, {}),

".enex": (EverNoteLoader, {}),

".eml": (MyElmLoader, {}),

".epub": (UnstructuredEPubLoader, {}),

".html": (UnstructuredHTMLLoader, {}),

".md": (UnstructuredMarkdownLoader, {}),

".odt": (UnstructuredODTLoader, {}),

".pdf": (PDFMinerLoader, {}),

".ppt": (UnstructuredPowerPointLoader, {}),

".pptx": (UnstructuredPowerPointLoader, {}),

".txt": (TextLoader, {"encoding": "utf8"}),

# Add more mappings for other file extensions and loaders as needed

}

def load\_single\_document(file\_path: str) -> List[Document]:

ext = "." + file\_path.rsplit(".", 1)[-1]

if ext in LOADER\_MAPPING:

loader\_class, loader\_args = LOADER\_MAPPING[ext]

loader = loader\_class(file\_path, \*\*loader\_args)

return loader.load()

raise ValueError(f"Unsupported file extension '{ext}'")

def load\_documents(source\_dir: str, ignored\_files: List[str] = []) -> List[Document]:

"""

Loads all documents from the source documents directory, ignoring specified files

"""

all\_files = []

for ext in LOADER\_MAPPING:

all\_files.extend(

glob.glob(os.path.join(source\_dir, f"\*\*/\*{ext}"), recursive=True)

)

filtered\_files = [file\_path for file\_path in all\_files if file\_path not in ignored\_files]

with Pool(processes=os.cpu\_count()) as pool:

results = []

with tqdm(total=len(filtered\_files), desc='Loading new documents', ncols=80) as pbar:

for i, docs in enumerate(pool.imap\_unordered(load\_single\_document, filtered\_files)):

results.extend(docs)

pbar.update()

return results

def process\_documents(ignored\_files: List[str] = []) -> List[Document]:

"""

Load documents and split in chunks

"""

print(f"Loading documents from {source\_directory}")

documents = load\_documents(source\_directory, ignored\_files)

if not documents:

print("No new documents to load")

exit(0)

print(f"Loaded {len(documents)} new documents from {source\_directory}")

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=chunk\_size, chunk\_overlap=chunk\_overlap)

texts = text\_splitter.split\_documents(documents)

print(f"Split into {len(texts)} chunks of text (max. {chunk\_size} tokens each)")

return texts

def does\_vectorstore\_exist(persist\_directory: str) -> bool:

"""

Checks if vectorstore exists

"""

if os.path.exists(os.path.join(persist\_directory, 'index')):

if os.path.exists(os.path.join(persist\_directory, 'chroma-collections.parquet')) and os.path.exists(os.path.join(persist\_directory, 'chroma-embeddings.parquet')):

list\_index\_files = glob.glob(os.path.join(persist\_directory, 'index/\*.bin'))

list\_index\_files += glob.glob(os.path.join(persist\_directory, 'index/\*.pkl'))

# At least 3 documents are needed in a working vectorstore

if len(list\_index\_files) > 3:

return True

return False

def main():

# Create embeddings

embeddings = HuggingFaceEmbeddings(model\_name=embeddings\_model\_name)

if does\_vectorstore\_exist(persist\_directory):

# Update and store locally vectorstore

print(f"Appending to existing vectorstore at {persist\_directory}")

db = Chroma(persist\_directory=persist\_directory, embedding\_function=embeddings, client\_settings=CHROMA\_SETTINGS)

collection = db.get()

texts = process\_documents([metadata['source'] for metadata in collection['metadatas']])

print(f"Creating embeddings. May take some minutes...")

db.add\_documents(texts)

else:

# Create and store locally vectorstore

print("Creating new vectorstore")

texts = process\_documents()

print(f"Creating embeddings. May take some minutes...")

db = Chroma.from\_documents(texts, embeddings, persist\_directory=persist\_directory, client\_settings=CHROMA\_SETTINGS)

db.persist()

db = None

print(f"Ingestion complete! You can now run privateGPT.py to query your documents")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Приложение 2. Программная реализация взаимодействия с загруженной языковой моделью

#!/usr/bin/env python3

from dotenv import load\_dotenv

from langchain.chains import RetrievalQA

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.callbacks.streaming\_stdout import StreamingStdOutCallbackHandler

from langchain.vectorstores import Chroma

from langchain.llms import GPT4All, LlamaCpp

import os

import argparse

load\_dotenv()

embeddings\_model\_name = os.environ.get("EMBEDDINGS\_MODEL\_NAME")

persist\_directory = os.environ.get('PERSIST\_DIRECTORY')

model\_type = os.environ.get('MODEL\_TYPE')

model\_path = os.environ.get('MODEL\_PATH')

model\_n\_ctx = os.environ.get('MODEL\_N\_CTX')

target\_source\_chunks = int(os.environ.get('TARGET\_SOURCE\_CHUNKS',4))

from constants import CHROMA\_SETTINGS

def main():

# Parse the command line arguments

args = parse\_arguments()

embeddings = HuggingFaceEmbeddings(model\_name=embeddings\_model\_name)

db = Chroma(persist\_directory=persist\_directory, embedding\_function=embeddings, client\_settings=CHROMA\_SETTINGS)

retriever = db.as\_retriever(search\_kwargs={"k": target\_source\_chunks})

# activate/deactivate the streaming StdOut callback for LLMs

callbacks = [] if args.mute\_stream else [StreamingStdOutCallbackHandler()]

# Prepare the LLM

match model\_type:

case "LlamaCpp":

llm = LlamaCpp(model\_path=model\_path, n\_ctx=model\_n\_ctx, callbacks=callbacks, verbose=False)

case "GPT4All":

llm = GPT4All(model=model\_path, n\_ctx=model\_n\_ctx, backend='gptj', callbacks=callbacks, verbose=False)

case \_default:

print(f"Model {model\_type} not supported!")

exit;

qa = RetrievalQA.from\_chain\_type(llm=llm, chain\_type="stuff", retriever=retriever, return\_source\_documents= not args.hide\_source)

# Interactive questions and answers

while True:

query = input("\nEnter a query: ")

if query == "exit":

break

# Get the answer from the chain

res = qa(query)

answer, docs = res['result'], [] if args.hide\_source else res['source\_documents']

# Print the result

print("\n\n> Question:")

print(query)

print("\n> Answer:")

print(answer)

# Print the relevant sources used for the answer

for document in docs:

print("\n> " + document.metadata["source"] + ":")

print(document.page\_content)

def parse\_arguments():

parser = argparse.ArgumentParser(description='privateGPT: Ask questions to your documents without an internet connection, '

'using the power of LLMs.')

parser.add\_argument("--hide-source", "-S", action='store\_true',

help='Use this flag to disable printing of source documents used for answers.')

parser.add\_argument("--mute-stream", "-M",

action='store\_true',

help='Use this flag to disable the streaming StdOut callback for LLMs.')

return parser.parse\_args()

if \_\_name\_\_ == "\_\_main\_\_":

main()