

Робототехника и комплексная автоматизация (РК)

Системы автоматизированного проектирования (РК6)

Студент	Гунько Никита Макарович
Группа	РК6-81Б
Тип практики	Преддипломная
Название предприятия	ООО «ЮБС»

Москва, 2023 г.

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.

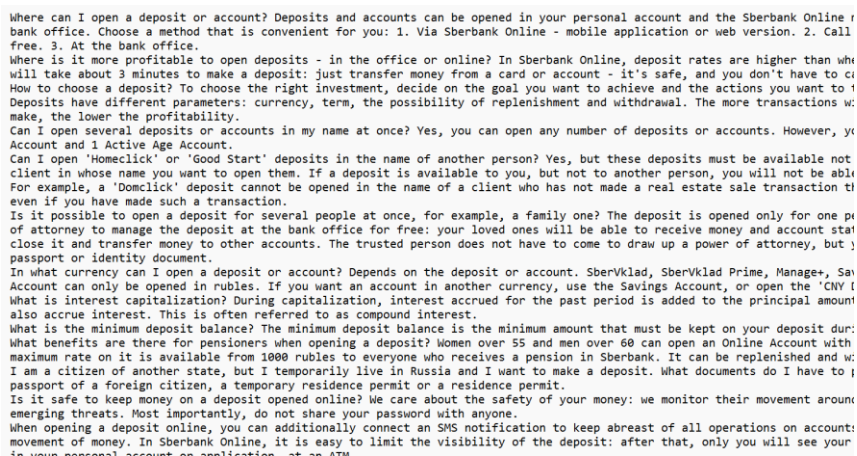
ОСНОВНАЯ ЧАСТЬ

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 эмбедингов 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 выбран 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.



Where can I open a deposit or account? Deposits and accounts can be opened in your personal account and the Sberbank Online bank office. Choose a method that is convenient for you: 1. Via Sberbank Online - mobile application or web version. 2. Call free. 3. At the bank office.

Where is it more profitable to open deposits - in the office or online? In Sberbank Online, deposit rates are higher than when you will take about 3 minutes to make a deposit: just transfer money from a card or account - it's safe, and you don't have to go to the bank.

How to choose a deposit? To choose the right investment, decide on the goal you want to achieve and the actions you want to take. Deposits have different parameters: currency, term, the possibility of replenishment and withdrawal. The more transactions you make, the lower the profitability.

Can I open several deposits or accounts in my name at once? Yes, you can open any number of deposits or accounts. However, you can open only 1 Active Age Account.

Can I open 'Homeclick' or 'Good Start' deposits in the name of another person? Yes, but these deposits must be available not only in whose name you want to open them. If a deposit is available to you, but not to another person, you will not be able to open it. For example, a 'Domclick' deposit cannot be opened in the name of a client who has not made a real estate sale transaction through Sberbank even if you have made such a transaction.

Is it possible to open a deposit for several people at once, for example, a family one? The deposit is opened only for one person. You need a power of attorney to manage the deposit at the bank office for free: your loved ones will be able to receive money and account statements, close it and transfer money to other accounts. The trusted person does not have to come to draw up a power of attorney, but you need a passport or identity document.

In what currency can I open a deposit or account? Depends on the deposit or account. SberVklad, SberVklad Prime, Manager+, Savings Account can only be opened in rubles. If you want an account in another currency, use the Savings Account, or open the 'GNY' (Gross National Product) Account.

What is interest capitalization? During capitalization, interest accrued for the past period is added to the principal amount and also accrue interest. This is often referred to as compound interest.

What is the minimum deposit balance? The minimum deposit balance is the minimum amount that must be kept on your deposit during the term of the deposit.

What benefits are there for pensioners when opening a deposit? Women over 55 and men over 60 can open an Online Account with a maximum rate on it is available from 1000 rubles to everyone who receives a pension in Sberbank. It can be replenished and withdrawn from at any time.

I am a citizen of another state, but I temporarily live in Russia and I want to make a deposit. What documents do I have to provide? A passport of a foreign citizen, a temporary residence permit or a residence permit.

Is it safe to keep money on a deposit opened online? We care about the safety of your money: we monitor their movement around the clock for emerging threats. Most importantly, do not share your password with anyone.

When opening a deposit online, you can additionally connect an SMS notification to keep abreast of all operations on account: deposits, withdrawals, transfers. In Sberbank Online, it is easy to limit the visibility of the deposit: after that, only you will see your deposit balance and movements.

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 эмбедингов 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 основную 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()

```