perplexity

как загружать файлы в локальную нейросеть через код langchain и ollama

Для загрузки файлов в локальную нейросеть через код с использованием LangChain и Ollama обычно применяется следующий подход:

1. Загрузка файлов и подготовка данных

- Используйте загрузчики документов LangChain (например, `JSONLoader`, `TextLoader`, `PDFLoader` и др.) для чтения локальных файлов и преобразования их в объекты `Document`.
- Для больших текстов применяйте текстовые сплиттеры (например, `RecursiveCharacterTextSplitter`), чтобы разбить содержимое на удобные по размеру чанки.
- Создайте эмбеддинги (векторные представления) чанков с помощью подходящей модели эмбеддингов (например, локальной модели через LocalAI или Ollama, если поддерживает embeddings).
- Сохраните эмбеддинги в векторном хранилище (например, Chroma, FAISS, или in-memory), чтобы потом эффективно искать релевантные куски текста.

2. Интеграция с локальной моделью Ollama через LangChain

- В LangChain используйте класс `Ollama` из `langchain_community.llms` для вызова локальной модели Ollama.
- Создайте цепочку (`LLMChain`) с промптом, в который можно передавать контекст из загруженных документов (например, объединённые чанки).
- При необходимости реализуйте RAG (retrieval-augmented generation), где сначала происходит поиск по векторной базе, а затем найденный контекст передаётся в промпт модели.

3. Пример кода (Python)

```
from langchain.document_loaders import JSONLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.llms import Ollama
from langchain.vectorstores import Chroma
from langchain.embeddings import OllamaEmbeddings  # если поддерживается
from langchain.chains import RetrievalQA

# Шаг 1: загрузка JSON файла
loader = JSONLoader("path/to/file.json", jq_schema=".data[]")
documents = loader.load()
```

```
# Шаг 2: разбивка на чанки
text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=20)
texts = text_splitter.split_documents(documents)

# Шаг 3: создание эмбеддингов (пример с Ollama, если поддерживает)
embeddings = OllamaEmbeddings(model="text-embedding-ada-002")  # пример, может отличаться

# Шаг 4: создание векторного хранилища
vectorstore = Chroma.from_documents(texts, embeddings)

# Шаг 5: инициализация локальной модели Ollama
llm = Ollama(model="mistral")

# Шаг 6: создание цепочки с retrieval
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=vectorstore.as_retriever())

# Запрос к модели с использованием загруженных данных
query = "Ваш вопрос по загруженным данным"
answer = qa_chain.run(query)
print(answer)
```

4. Особенности и рекомендации

- Для локальной работы с Ollama нужно, чтобы модель была предварительно загружена и запущена локально.
- Если Ollama не поддерживает эмбеддинги, можно использовать отдельную локальную модель для эмбеддингов (например, через LocalAI).
- При большом объёме данных рекомендуется использовать векторные базы с постоянным хранением (Chroma, FAISS), а не только in-memory.
- Весь процесс загрузки, разбиения, создания эмбеддингов и вызова модели можно автоматизировать и интегрировать в единую цепочку LangChain.
- Для загрузки моделей и управления ими через API можно использовать Postman или curl, если Ollama предоставляет REST API.

Таким образом, загрузка файлов в локальную нейросеть через LangChain и Ollama сводится к:

- Загрузке и подготовке документов через LangChain loaders и splitters.
- Созданию эмбеддингов и векторного хранилища.
- Вызову локальной модели Ollama через LangChain с передачей контекста из загруженных данных.

Это позволяет реализовать локальный RAG-подход для работы с пользовательскими файлами и локальными LLM.
[1] [2] [3]

⁂

1. https://habr.com/ru/companies/reksoft/articles/792496/

2. https://habr.com/ru/companies/reksoft/articles/796613/
3. https://tproger.ru/articles/privategpt-yazykovaya-model-dlya-raboty-s-dokumentami