



# Парсинг документов для эмбедингов

11 Aug 2025, 7:15 pm

Итак привет! Сегодня мы будем подготавливать документы для преобразования в эмбединги. 1.способы парсинга документов, так чтобы никакая информация не потерялась (например при двухколоночном PDF) Подготовь код для локального использования, можно использовать только open ворс решения для данного решения.



11 Aug 2025, 7:15 pm • Model: Deep Research (o3)

Привет! Чтобы подготовить оптимальный код, можешь уточнить:

1. Какие форматы документов ты планируешь обрабатывать? (например, PDF, DOCX, TXT)
2. Какие типы содержимого критично сохранить? (текст, таблицы, формулы, изображения, порядок элементов и т.д.)
3. Предпочтения по языкам программирования и библиотекам? Python с PyMuPDF, pdfplumber, PDFMiner, или что-то ещё?
4. Планируется ли извлекать документы пакетно (много файлов) или по одному?

Как только уточнишь — соберу код с учётом всех требований.

11 Aug 2025, 7:23 pm

1. PDF, DOCX, TXT, pptx, excel, csv, все содержащие текстовую информацию, а также подключение к базам данных. Postgres 2.текст, таблицы, формулы, изображения, порядок элементов и т.д.) 3. python используй библиотеки доступные как из лонгчайн так и из других библиотек 4 пакетно - это промышленное решение, при котором производится сразу определение категорий вот пример нашего стартового набора документов:  
{‘Нерудные\_материалы’: [‘doc/Нерудные\_материалы/Условия поставки нерудных материалов железнодорожным транспортом (от 17.06.2025).pdf’, ‘doc/Нерудные\_материалы/Общие условия к Договору поставки нерудных материалов (от 16.04.2024).pdf’, ‘doc/Нерудные\_материалы/Условия поставки нерудных материалов автомобильным транспортом (от 16.04.2024).pdf’, ‘doc/Нерудные\_материалы/Условия поставки нерудных

материалов водным транспортом (от 16.04.2024).pdf', 'doc/  
Нерудные\_материалы/Условия о стандартных оговорках к договору поставки  
нерудных материалов (от 16.04.2024).pdf'], 'Железобетонные\_изделия':  
['doc/Железобетонные\_изделия/Условия поставки автомобильным  
транспортом (от 25.12.2024).pdf', 'doc/Железобетонные\_изделия/Общие  
условия к Договору поставки (от 25.12.2024).pdf', 'doc/  
Железобетонные\_изделия/Условия о стандартных оговорках (от  
25.12.2024).pdf'], 'Информация\_для\_Поставщиков': ['doc/  
Информация\_для\_Поставщиков/73a0e7e8k3cv397txl78fryp7myj1yk8.pdf',  
'doc/  
Информация\_для\_Поставщиков/jck2g2n6hg0hvwtdm9ab8bnoub49v17l.pdf',  
'doc/  
Информация\_для\_Поставщиков/hrzic06ymeyukmw574wrmox1z5uzx25q.pdf',  
'doc/  
Информация\_для\_Поставщиков/jfqjp0z614mtcc6s9rkib9i2q07dks21.pdf',  
'doc/  
Информация\_для\_Поставщиков/zlwf7y6ef1wp1m4lpn0vzmq2v9za9t8.pdf',  
'doc/  
Информация\_для\_Поставщиков/sqrr5bf1spmusuz38fzdov7wcydhabh3.pdf',  
'doc/  
Информация\_для\_Поставщиков/xdc987jlb4mmk9mb6pd0sodbq8hbvn3e.pdf',  
'doc/Информация\_для\_Поставщиков/dptc5tkjnz2ll94sslu97dl6wc2lj1c8.pdf',  
'doc/  
Информация\_для\_Поставщиков/500ebb2f3fe47c94831b8bec83407a3b.pdf',  
'doc/  
Информация\_для\_Поставщиков/ecc0pm2wmuqhg11dyi6b4l2ajnlhj23x.pdf',  
'doc/  
Информация\_для\_Поставщиков/o72451xgkopg5cw22lcz0i2tao0uasb2.pdf',  
'doc/  
Информация\_для\_Поставщиков/gjqyo1x2pyelz58y7wt4891dm1ijw7q4.pdf',  
'doc/  
Информация\_для\_Поставщиков/yg7319dhabuq1lfz7ovn29tti0qx7fnp.pdf',  
'doc/  
Информация\_для\_Поставщиков/nkstax3697flq0etdj76qmii2emyy3jv.pdf',  
'doc/  
Информация\_для\_Поставщиков/4ugb038jzanlkmugm09nxpqs48lr5q94.pdf',  
'doc/  
Информация\_для\_Поставщиков/xcnz1l70u3n5sfib1zh2o8kl93cgn1ox.pdf',  
'doc/  
Информация\_для\_Поставщиков/501myrerq2ysc6gk4seoln9mhfc9r4ro.pdf',  
'doc/  
Информация\_для\_Поставщиков/p9mp7nyfv8uzjivfi0n8nedqocnobzt8.pdf',  
'doc/Информация\_для\_Поставщиков/

Кодекс\_делового\_поведения\_для\_поставщиков.pdf', 'doc/  
Информация\_для\_Поставщиков/vmf9ppwqznxg3plqnm9rr2mwm4gth7ww.pdf',  
'doc/  
Информация\_для\_Поставщиков/4dvwvpuke6gxzn7fpj3yw0o62ofoy518.pdf',  
'doc/Информация\_для\_Поставщиков/bx8vlk1desdestrpczriqfjp4tcetdvw.pdf',  
'doc/  
Информация\_для\_Поставщиков/ad5sbxhmaeydamw1jkxtdgy9jimpaw43.pdf',  
'doc/  
Информация\_для\_Поставщиков/wg2ancbv2hucsmwx6ah7wz3wz5vhv9ka.pdf'],  
'Политики': ['doc/Политики/re6002fnyfoojr1memk1f3c1vq2qp6j2.pdf', 'doc/  
Политики/.ipynb\_checkpoints', 'doc/  
Политики/37a9rifoqkd801mtu222kczm0xm1vmt3.pdf', 'doc/Политики/  
Кодекс\_делового\_поведения\_для\_поставщиков.pdf', 'doc/  
Политики/bxlhd9e78jbno6t5104mxg5lqi3443vd.pdf'], 'Цемент': ['doc/  
Цемент/Условия поставки автомобильным транспортом (от 25.12.2024).pdf',  
'doc/Цемент/2zmvnznz6qubyicjeodzrlqtvhucr3l5.pdf', 'doc/Цемент/Условия  
о стандартных оговорках (от 25.12.2024).pdf', 'doc/Цемент/Условия поставки  
железнодорожным транспортом (от 17.06.2025).pdf', 'doc/Цемент/Общие  
условия к договору поставки (от 17.06.2025).pdf']] в папке doc. Также учти  
что необходимо строить иерархический граф по документам и внутри них.  
Можно использовать по спи локальную LLM для таких решений. Помни мы  
пытаемся создать онпремис решение для корпорации. Для референта есть  
вот такой инструмент внимательно его изучи:  
<https://github.com/dmayboroda/minima>



11 Aug 2025, 7:23 pm • Model: Deep Research (o3)

Отлично, я подготовлю промышленный пайплайн на Python для парсинга PDF, DOCX, TXT, PPTX, Excel, CSV и подключения к PostgreSQL, с сохранением всех текстовых, табличных и визуальных данных без потерь, включая двухколоночные PDF. Также учту построение иерархических графов по структуре документов, поддержку пакетной обработки и локальную совместимость с LLM, ориентируясь на возможности проекта Minima. Как только решение будет готово, я предоставлю тебе код и инструкции по развёртыванию.



11 Aug 2025, 7:33 pm • Model: Deep Research (o3)

# Подготовка документов для эмбедингов

## Обзор задачи и поддерживаемые форматы

Для качественного **встраивания (embedding)** текстовой информации из корпоративных данных необходимо предварительно разобрать исходные документы разных типов, не потеряв при этом содержимое. В корпоративной среде часто встречаются PDF-файлы (в том числе со сложной версткой), документы Word (DOC/DOCX), презентации PowerPoint (PPT/PPTX), электронные таблицы Excel (XLS/XLSX, CSV), простые тексты (TXT/Markdown) и данные, хранящиеся в реляционных базах (например, PostgreSQL). Необходимо обеспечить парсинг всех этих форматов с *сохранением всей информации* – текста, таблиц, формул, изображений, порядка следования элементов и т.д. В рамках on-premises решения будем использовать только открытые библиотеки (open source). К счастью, существуют готовые инструменты для локальной обработки: например, проект **Minima** представляет контейнерное RAG-решение с поддержкой локального индексирования PDF, XLS, DOCX, TXT, MD и CSV файлов [kdjinpai.com](https://kdjinpai.com). Мы рассмотрим, как реализовать подобную функциональность самостоятельно с помощью Python-библиотек (в том числе интеграций LangChain для загрузки документов различных типов [stackoverflow.com](https://stackoverflow.com)).

**Пакетная обработка и категории:** В промышленном сценарии документы обрабатываются *пакетно*. Часто им заранее назначаются категории (например, по папкам). В примере задан набор категорий с привязанными файлами, например:

Python

```
{ 'Нерудные_материалы': [ 'doc/Нерудные_материалы/Условия поставки ...  
.pdf', ... ],  
  'Железобетонные_изделия': [ 'doc/Железобетонные_изделия/Общие условия ...  
.pdf', ... ], ... }
```

При обработке можно автоматически присваивать каждому документу метаданные категории (на основе имени папки или предварительно заданного словаря). Это облегчит построение иерархической структуры знаний.

Ниже представлены подходы к парсингу каждого типа данных и кодовые примеры для локального использования.

## Парсинг PDF без потери содержимого

**Проблема:** Простое извлечение текста из PDF постранично (например, методом `PdfReader.extract_text()` из PyPDF) может **перемешать колонки** и нарушить порядок текста. Особенно это актуально для многоколоночных макетов отчетов, статей, где текст идет параллельно в двух колонках. Также PDF могут содержать *таблицы и формулы/диаграммы* (часто как изображения или сложная разметка), которые нужно корректно извлечь.

**Решения:**

- Использовать библиотеку **PyMuPDF (fitz)**, умеющую анализировать блоки текста и координаты. В PyMuPDF есть утилита `multi_column.py`, определяющая границы колонок на странице с помощью встроенного механизма обнаружения текстовых блоков [artifex.com](https://artifex.com). Она возвращает список прямоугольных областей (bounding boxes) для колонок, отсортированных в порядке чтения. Затем можно извлекать текст по каждой области поочередно, сохраняя правильный порядок. Например:

Python

```
import fitz # PyMuPDF
doc = fitz.open("report.pdf")
for page in doc:
    boxes = column_boxes(page) # функция из multi_column.py, определяющая
    колонки
    page_text = ""
    for rect in boxes:
        # Извлекаем текст внутри прямоугольника колонки, сортируя по строкам
        page_text += page.get_text(clip=rect, sort=True) + "\n"
    print(page_text)
```

В примере функция `column_boxes` возвращает координаты каждой колонки, и мы объединяем текст колонок по порядку [artifex.com](https://artifex.com). Такой подход позволяет «видеть» колонки и извлекать текст *в логической последовательности*, как если бы мы читали документ. PyMuPDF также может извлекать *изображения* из PDF при необходимости (через `page.get_images()`), и поддерживает поиск по координатам, что полезно для выделения областей, содержащих формулы или подписи к рисункам.

- **PDFPlumber** – еще одна open source библиотека для PDF. Хотя напрямую она возвращает весь текст страницы (и может сшивать соседние колонки), можно обойти это, *искусственно разделив страницу*. Например, с помощью метода `crop` можно разрезать страницу на левую и правую половину и извлечь их текст отдельно [stackoverflow.com](https://stackoverflow.com). Этот метод работает, если в документе ровно 2 колонки фиксированной ширины. Однако он требует аккуратности (например, страницы с плавающими изображениями или переменным числом колонок могут потребовать динамического определения границ). PyMuPDF в этом плане более гибок, т.к. анализирует *блоки текста* автоматически.
- **Unstructured** – универсальный фреймворк для парсинга документов. Имеет функцию `partition_pdf`, которая возвращает список структурированных элементов (`Element`), каждый с типом (текстовый абзац, заголовок, список, таблица и пр.) и метаданными [unstructured.io](https://unstructured.io). Unstructured умеет при необходимости применять OCR: например, для *таблиц* можно использовать стратегию `hi_res`, чтобы компьютерное зрение + OCR извлекли таблицу как текст **с сохранением структуры (даже HTML-кодом таблицы)** [unstructured.io](https://unstructured.io). Это позволяет не потерять данные, заключенные в ячейки, и восстановить табличный формат при выводе или передаче в LLM. Пример:

Python

```
from unstructured.partition.pdf import partition_pdf
elements = partition_pdf("report.pdf", strategy="hi_res")
for elem in elements:
    print(elem.type, ":", elem.text[:100])
```

Здесь `elements` может включать в себя объекты типа `Title`, `NarrativeText`, `Table`, `Figure` и т.д., каждый с соответствующим содержимым. Использование `Unstructured` в сочетании с `LangChain` позволяет сразу получать `Document` с разделением по элементам. Например, `LangChain` предоставляет `PyPDFLoader` или интеграцию `UnstructuredPDFLoader`. В словаре соответствия форматов мы можем указать, что для `.pdf` применяется, скажем, `PyPDFLoader` (который внутри может использовать `PyMuPDF`) [stackoverflow.com](https://stackoverflow.com). В результате текст PDF будет загружен либо целиком, либо разбит на сегменты, готовые для дальнейшей обработки (например, chunking для эмбединга).

**Извлечение таблиц и формул:** Если PDF содержит таблицы, помимо упомянутого подхода с `Unstructured` (OCR), существуют специализированные библиотеки: **Camelot**, **Tabula-py** – для табличных данных (они пытаются распознать границы таблиц по линиям/структуре PDF и вернуть данные в CSV/DataFrame формат). Для формул в PDF ситуаций несколько: если формула написана текстом (LaTeX-символами) – она извлечется как текст; если она встроена как изображение – можно применить OCR или специализированные решения распознавания формул (например, *im2latex* модели). В сложных случаях, для 100% сохранения формул, можно хранить изображение формулы и заменить его на специальный токен в тексте (например, `<FORMULA_IMG>`), с возможностью последующей обработки. Главная цель – не потерять контент: либо текстом, либо ссылкой на объект формулы.

## Парсинг документов Microsoft Word (DOCX)

**Особенности DOCX:** Документ Word хранится в формате Office Open XML. Его содержимое включает текст с форматированием, таблицы, изображения, списки, заголовки разделов, а также *формулы*, созданные средствами Word (OMML). При разборе DOCX важно извлечь весь основной текст и табличные данные. Формулы Word представляют проблему – библиотека **python-docx** (самый популярный инструмент для DOCX) *не поддерживает уравнения* через свой API [stackoverflow.com](https://stackoverflow.com). Формулы внутри DOCX фактически хранятся в XML (теги `<m:oMath>`), но `python-docx` их видит как непрозрачные объекты. Поэтому возможны два подхода: либо конвертировать Word в другой формат (PDF или HTML) и разобрать формулы на этом этапе, либо использовать низкоуровневый разбор XML.

### Библиотеки и подходы:

- **Python-docx:** Позволяет открывать `.docx` и извлекать текст параграфов, ячеек таблиц, списков. Пример:

Python



```

from docx import Document
doc = Document("file.docx")
full_text = []
for para in doc.paragraphs:
    full_text.append(para.text)
for table in doc.tables:
    for row in table.rows:
        cells = [cell.text for cell in row.cells]
        full_text.append("\t".join(cells))
text = "\n".join(full_text)

```

Этот код собирает все абзацы и таблицы (таблицы конкатенируются строками с разделителем). Он захватывает основной текст и табличные данные. **Однако:** формулы (если есть) не появятся, а изображения можно только сохранить отдельно ( `doc.part.related_parts` содержит объекты изображений). Таким образом, `python-docx` покрывает текстовую часть. Если формулы для нас критичны, можно дополнительно пройти по XML: найти элементы `<m:oMath>` и извлечь их XML или MathML. В качестве упрощения можно заменить такие объекты плейсхолдером `[FORMULA]` и сохранить картинку формулы (например, конвертировав документ в HTML, Word обычно сохраняет формулы как изображения с alt-текстом в MathML).

- **Docx2txt:** Утилита, конвертирующая `.docx` в простой текст (сохраняя примерно расположение). `LangChain` предоставляет `Docx2txtLoader`, использующий эту утилиту [python.langchain.com](https://python.langchain.com). Он быстро получает весь текст, но может опустить нюансы форматирования (например, разделитель таблицы может быть пробелами). Если структурная точность не ключевая, это простой вариант. В противном случае – лучше парсить с `python-docx` или `Unstructured`.
- **UnstructuredWordDocumentLoader:** `LangChain`-обертка над `Unstructured` для `DOCX` [python.langchain.com](https://python.langchain.com). Она способна разделять документ на элементы. `Unstructured` распознает заголовки, списки, таблицы в Word-файле. Как и для PDF, можно указать `mode="elements"`, чтобы не склеивать всё в один текст, а получить список элементов-документов [python.langchain.com](https://python.langchain.com). Например, заголовки разделов можно получить отдельно от абзацев. В метаданных таких элементов будет указана категория (например, 'Title', 'List', 'Table') и даже уровень вложенности заголовка (если определимо) [python.langchain.com](https://python.langchain.com). Это очень удобно для построения иерархии документа (см. ниже).

Пример кода:

Python

```

from langchain_community.document_loaders import UnstructuredWordDocumentLoader
loader = UnstructuredWordDocumentLoader("file.docx", mode="elements")
docs = loader.load()
for doc in docs:
    print(doc.metadata.get("category"), ":", doc.page_content[:30])

```

На выходе `docs` – список `Document`, где каждый соответствует элементу (например, заголовок или абзац). По метаданным `category` можно определить тип (например,

‘Title’ для заголовков, ‘NarrativeText’ для обычных абзацев и т.д.). Если не нужны отдельные элементы, можно использовать `mode="single"` (объединить все в один Document) – тогда структура потеряется, но текст сохранится.

**Вывод изображений:** Изображения в DOCX можно извлечь через `python-docx` ( `Document.parts` хранит `media`) или `Unstructured` (он должен пометить в элементах `Figure` с указанием подписи, однако сам файл изображения надо сохранять отдельно). Для целей текстового эмбединга, как правило, изображения либо игнорируются, либо их подписи включаются как текст. Если необходимо включить информацию с изображения (например, скриншот с текстом), можно прогнать его через OCR (например, `pytesseract` ). Это выйдет за рамки стандартного парсинга, но обеспечит отсутствие потери информации даже из картинок.

## Обработка текстовых файлов (TXT, Markdown)

**TXT** – самый простой случай. Нужно прочитать файл с правильной кодировкой (обычно UTF-8). Например:

Python

```
with open("file.txt", "r", encoding="utf-8") as f:
    text = f.read()
```

Затем текст можно разделять на документы/строки по необходимости. В `LangChain` есть готовый `TextLoader`. Markdown-файлы ( `.md` ) также можно считывать как текст; при желании можно конвертировать их HTML-парсером или использовать `UnstructuredMarkdownLoader` для разбиения по элементам (заголовки, списки и т.д.).

При обработке *больших* TXT стоит учитывать **chunking** – разбивку на фрагменты для эмбединга, но это уже следующий этап после извлечения.

## Презентации PowerPoint (PPTX)

Презентации содержат слайды с различными полями текста, заголовками, списками и иногда табличными объектами или изображениями. Задача – вытащить текст с каждого слайда в понятном порядке.

**Подходы:**

- Библиотека **python-pptx**: Позволяет открыть `.pptx` и итерироваться по слайдам и их `shape`-объектам. Каждый `shape` может содержать `text_frame` с параграфами текста или быть таблицей ( `shape.table` ) или картинкой. Код для извлечения текста:

Python

```
from pptx import Presentation
prs = Presentation("slides.pptx")
```



```

all_text = []
for slide in prs.slides:
    # Например, сначала заголовок слайда:
    title_shapes = [sh for sh in slide.shapes if sh.name.startswith("Title")]
    for sh in title_shapes:
        if sh.has_text_frame:
            all_text.append(sh.text) # заголовок
# Затем все текстовые объекты
for shape in slide.shapes:
    if shape.has_text_frame:
        all_text.append(shape.text)
    elif shape.has_table:
        for row in shape.table.rows:
            cells = [cell.text for cell in row.cells]
            all_text.append("\t".join(cells))
text = "\n".join(all_text)

```

Такой код собирает тексты из всех текстовых рамок и таблиц. Мы специально обрабатываем заголовки (если имя shape указывает на Title, это часто главный заголовок слайда). В результате получим текст, при необходимости разбитый по строкам/таблицам. Изображения можно сохранить из `shape.image` (если нужно OCR, как обсуждалось ранее).

- **UnstructuredPowerPointLoader** (LangChain): Использует `python-pptx` и `Unstructured`. Он аналогично извлекает все текстовые элементы. По умолчанию объединяет их в один Document (на слайд или весь файл), но можно задать `mode="elements"` для разбивки по отдельным элементам с метаданными `python.langchain.com` (как и в случае Word). Преимущество – наличие метаданных о слайде, дате модификации, языке и т.п. (подтягивается `Unstructured`, см. пример с `'category_depth', 'page_number'` и др. в выводе метаданных `python.langchain.com`). Если нужна структура, можно, например, группировать элементы по номеру слайда.

**Примечание:** Формат PPT (старый бинарный) лучше предварительно конвертировать в PPTX (например, открыть и сохранить в новой версии), так как многие инструменты ориентированы на PPTX (Open XML).

## Таблицы Excel (XLSX) и CSV

**CSV:** текстовый формат таблицы, легко читается стандартными средствами. В LangChain есть `CSVLoader`, который по умолчанию читает CSV и формирует Document, где каждая строка представлена как строка текста, а первая строка как заголовок колонок. Можно и самостоятельно:

Python

```

import csv
docs = []
with open("data.csv", newline='', encoding='utf-8') as f:

```

```

reader = csv.reader(f)
headers = next(reader, None)
for row in reader:
    # Создаем строку вида "Col1: value1; Col2: value2; ..."
    if headers:
        content = "; ".join(f"{h}: {val}" for h, val in zip(headers, row))
    else:
        content = ", ".join(row)
    docs.append(content)

```

Здесь `docs` будет списком строковых документов – по одному на каждую запись CSV. Их можно дальше преобразовать в объекты Document с метаданными. Такой подход не теряет данных. При очень больших CSV может потребоваться стриминг/постраничная загрузка, но с точки зрения эмбединга часто CSV не бывают огромными текстами (скорее, структурированные данные).

**Excel (XLSX):** Можно использовать **pandas** или **openpyxl**. Pandas способен прочитать файл Excel целиком ( `pd.read_excel` ) или лист за листом. Например:

Python

```

import pandas as pd
xls = pd.ExcelFile("book.xlsx")
for sheet_name in xls.sheet_names:
    df = pd.read_excel(xls, sheet_name)
    for index, row in df.iterrows():
        # Преобразуем строку DataFrame в текст
        content = "; ".join(f"{col}: {row[col]}" for col in df.columns)
        docs.append(content)

```

Мы обходим каждый лист и каждую строку, превращая её в текст аналогично CSV. Если же Excel-файл – это *не структурированные данные*, а, скажем, финансовый отчет с формулами, сводными таблицами – такой метод представит только вычисленные значения ячеек. Формулы Excel как таковые (их выражения) получить можно через **openpyxl** ( `cell.value` вернет формулу, если **openpyxl** открыть файл с `keep_links=True` ), но для целей семантического поиска формулы Excel обычно не нужны. Главное – сохранить текст из ячеек и заголовков.

**Примечание:** Excel .xls (старый формат) можно читать через `pd.read_excel` (требуется установленных движков, например `xlrd` для xls; версии `xlrd` > 2.0 не поддерживают xls, тогда можно использовать **openpyxl** или конвертировать файл).

## Извлечение текстов из базы данных (PostgreSQL)

Подключение к базе данных позволяет включить в наш корпус текста данные, которые хранятся не в файловой системе, а в SQL-таблицах. Здесь возможно два сценария:

1. **Хранение документов в БД** – например, таблица содержит столбец с полным текстом документов (или JSON). Тогда выгружаем эти тексты как единицы для эмбединга.
2. **Структурированные данные** – например, у нас есть таблица с колонками, и мы хотим использовать её содержимое в поиске знаний. Можно превратить каждую строку таблицы в кусок текста (как мы делали для CSV).

Для подключения используем библиотеку **psycopg2** (либо SQLAlchemy). Пример извлечения:

Python

```
import psycopg2
conn = psycopg2.connect(host="localhost", dbname="mydb", user="user",
password="pwd")
cur = conn.cursor()
cur.execute("SELECT id, title, content FROM documents;")
rows = cur.fetchall()
docs = []
for (doc_id, title, content) in rows:
    text = f"Document {doc_id}: {title}\n{content}"
    docs.append(text)
conn.close()
```

Здесь мы из каждой строки делаем текст: вставляем заголовок, содержимое. Далее `docs` можно обрабатывать аналогично файлам. Если данных очень много, можно выгружать порциями или фильтровать по дате/категории на уровне SQL. Также можно прикрепить к Document метаданные из полей (например, автор, дата), чтобы потом с их помощью делать фильтрацию в векторном хранилище.

LangChain предоставляет утилиту `SQLDatabase` и даже цепочку для SQL-вопросов, однако для нашей задачи (подготовка эмбедингов) проще явно выполнить нужные SELECT-запросы и сформировать текст.

## Сохранение структуры и построение иерархического графа

После парсинга всех документов важно **сохранить структуру**: как между документами, так и внутри них. В нашем случае структура многоуровневая: *категория* → *документ* → *разделы/элементы*.

**Междокументная иерархия (категории):** Если документы рассортированы по папкам или меткам категорий, можно отразить это в графе знаний. Например, узел “Нерудные материалы” связан с узлами-документами (условия поставки и др.). В процессинг-пайплайне это реализуется присвоением каждому Document поля

`metadata["category"]` . Тогда при ответах на вопросы или при фильтрации можно использовать эту категорию.

**Внутренняя структура документов:** Благодаря инструментам вроде Unstructured, мы можем получить *заголовки разделов* и другие элементы. Например, элементы с `category="Title"` и разной глубиной могут соответствовать структуре подразделов `python.langchain.com` . Построим дерево: для каждого документа собираем его элементы; первые элементы `Title` (depth 0) – это, возможно, заголовок документа, или разделы верхнего уровня. Более мелкие заголовки (depth 1, 2) – вложенные подразделы. Такое дерево можно хранить в виде, например, вложенного словаря или специальной структуры. Также можно использовать графовую библиотеку (NetworkX) для хранения и traversal: вершины – идентификаторы элементов, ребра – “часть/подразделение”. Это пригодится, если нужно отвечать на вопросы по разделам или делать тематический поиск.

### Пример иерархии:

Text

Категория "Цемент"

- └─ Документ "Общие условия к договору поставки (17.06.2025)"
  - └─ Раздел 1: ...
    - └─ Подраздел 1.1: ...
  - └─ Раздел 2: ...
  - └─ ...

Мы получили бы такое, если в тексте документа присутствуют заголовки. Извлекая их (например, текст, оформленный стилем Heading 1, Heading 2 в Word, или крупный полужирный шрифт в PDF – Unstructured умеет определять заголовки), можем воссоздать эту структуру.

**Связи между документами:** Помимо иерархии, в граф можно добавить связи типа “документ А ссылается на документ В” (если, скажем, обнаружены ссылки). Это уже выходит за рамки простой иерархии, но упоминание возможно: на этапе парсинга можно поискать шаблоны ссылок (названия других документов, гиперссылки) и собирать такие связи.

## Генерация эмбеддингов и хранилище векторів

Когда тексты извлечены и структурированы, их можно конвертировать в эмбеддинги. На 100% локальном решении мы используем модели **Sentence Transformers** или аналогичные. К примеру, Minima применяет модель *all-mpnet-base-v2* из SentenceTransformers для получения эмбеддингов размерности 768 `github.com` . Выбор модели зависит от языка документов (для русского можно взять многоязычные модели, например `paraphrase-multilingual-MiniLM` ). Пример кода эмбеддинга:

Python

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')
vectors = [model.encode(doc) for doc in docs] # docs — список строк или объектов Document
```

Здесь `vectors` – список numpy-массивов (эмбеддингов). Далее их обычно загружают в векторное хранилище для поиска. В open-source доступны, например, **FAISS** (библиотека Facebook для анн поиска) или **Qdrant** – высокопроизводительное векторное хранилище (Minima использует Qdrant для индексации локальных файлов [9†L9-L12]). С Vector Store можно выполнять семантический поиск, а связав его с LLM – реализовать *Retrieval Augmented Generation* (RAG), когда при вопросе пользователя находятся релевантные фрагменты документов и передаются локальной LLM для генерации ответа.

## Заключение

Подытоживая: для *полноценной* подготовки корпоративных документов к эмбеддингу необходим конвейер, который: (1) **парсит каждый документ** с учетом его формата, используя специализированные open-source библиотеки (PyMuPDF, PDFPlumber, python-docx, python-pptx, pandas и др., а также интеграции Unstructured/LangChain для унификации работы); (2) **не теряет информации** – обрабатывает многоколоночный текст `artifex.com` `artifex.com` , сохраняет содержимое таблиц `unstructured.io` , выявляет структуры (заголовки, списки) и по возможности формулы/изображения; (3) **добавляет метаданные** о структуре (категории, разделы), строя иерархический граф знаний; (4) **получает эмбеддинги** локальной моделью и сохраняет их во внутреннем хранилище.

Такое решение будет полностью on-premises, без внешних зависимостей, и обеспечит максимальную полноту данных при последующем поиске или диалоговом взаимодействии с документами. Используя рассмотренные библиотеки и подходы, можно добиться высокого качества индексирования разнородных корпоративных данных, аналогично готовым решениям вроде Minima `kdjinpai.com` , но с необходимой гибкостью под задачи организации.

11 Aug 2025, 8:18 pm

Плохо!!! Позор!!! o3 был лучше!!!



11 Aug 2025, 8:18 pm • Model: gpt-5-thinking

Понял, принял. Мой прошлый ответ не дотянул до вашего продакшн-уровня. Исправляюсь: ниже — готовый **on-prem, open-source** пайплайн для **пакетного** парсинга PDF/DOCX/TXT/PPTX/Excel/CSV + извлечения из PostgreSQL, с учётом **двухколоночных PDF**, таблиц, формул, изображений и **иерархического графа**. Код рассчитан на локальный запуск без внешних API и подготовлен «под эмбеддинги».

# 1) Установка

Bash

```
# Python 3.10+ рекомендуем
python -m venv venv
source venv/bin/activate # Windows: venv\Scripts\activate

pip install --upgrade pip

pip install pymupdf pdfplumber python-docx python-pptx pandas openpyxl lxml \
pillow pytesseract opencv-python-headless networkx tqdm pydantic \
sqlalchemy psycpg2-binary unstructured langchain-unstructured

# (опционально для PDF-таблиц; требует system deps: ghostscript, tk):
# pip install camelot-py[cv]
# (опционально для OCR качества)
# brew install tesseract # macOS
# sudo apt-get install tesseract-ocr # Linux
```

## 2) Структура проекта

Text

```
doc_ingest/
  main.py
  config.py
  models.py
  utils_io.py
  graph_builder.py
  parsers/
    __init__.py
    pdf_parser.py
    docx_parser.py
    pptx_parser.py
    excel_csv_parser.py
    txt_parser.py
    pg_loader.py
  outputs/
    dataset/      # JSONL/CSV для эмбедингов
    media/        # извлечённые изображения/формулы
    graphs/       # GraphML/JSON иерархий
```



## 3) Модели данных (единный формат элементов)

Python

```
# models.py
from pydantic import BaseModel, Field
from typing import Optional, List, Dict, Literal
from uuid import uuid4
from pathlib import Path

ElementType =
Literal["paragraph", "title", "list", "table", "formula", "image", "metadata", "slide", "row"]

class DocElement(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid4()))
    category: Optional[str] = None
    doc_path: Optional[str] = None
    doc_id: Optional[str] = None
    source_type: Optional[str] = None # pdf/docx/pptx/xlsx/csv/txt/pg
    page: Optional[int] = None
    order: Optional[int] = None
    bbox: Optional[List[float]] = None # [x0,y0,x1,y1] для PDF
    element_type: ElementType
    text: Optional[str] = None # основной текст
    html: Optional[str] = None # табличный/формульный HTML
    media_path: Optional[str] = None # сохранённое изображение/формула (png)
    headings_path: Optional[List[str]] = None # ["Гл.1", "1.1", ...]
    parents: Optional[List[str]] = None # id узлов-родителей в графе
    metadata: Dict = Field(default_factory=dict)
```

## 4) Утилиты ввода/вывода

Python

```
# utils_io.py
import json, os
from typing import Iterable
from pathlib import Path
from models import DocElement

def ensure_dirs():
    Path("outputs/dataset").mkdir(parents=True, exist_ok=True)
```

```
Path("outputs/media").mkdir(parents=True, exist_ok=True)
Path("outputs/graphs").mkdir(parents=True, exist_ok=True)
```

```
def dump_jsonl(elements: Iterable[DocElement], out_path: str):
    with open(out_path, "w", encoding="utf-8") as f:
        for el in elements:
            f.write(json.dumps(el.model_dump(), ensure_ascii=False) + "\n")
```

## 5) Конфигурация

Python

```
# config.py
from typing import Dict, List, Optional
from dataclasses import dataclass, field

@dataclass
class PipelineConfig:
    # входы
    categories: Dict[str, List[str]] # {"Категория": ["doc/..pdf", ...], ...}
    # postgres как источник (опционально)
    pg_dsn: Optional[str] = None      # "postgresql://user:pwd@host:5432/db"
    pg_query: Optional[str] = None    # "select id, title, content from documents"

    # парсинг
    pdf_ocr_fallback: bool = True     # OCR для сканов
    pdf_table_extraction: bool = True # попытка вытаскивать таблицы
    keep_images: bool = True          # сохранять изображения
    keep_formulas: bool = True

    # chunk/порядок
    max_chars_per_element: int = 15000 # «сырой» элемент (резка для эмбедингов
    # отдельно)

    # производительность
    workers: int = 4
```

## 6) PDF: корректный порядок, двухколоночные, таблицы, OCR

Python

```

# parsers/pdf_parser.py
from typing import List, Dict, Iterable, Optional
from models import DocElement
from pathlib import Path
import fitz  # PyMuPDF
import pdfplumber
from PIL import Image
import io, os
import numpy as np
try:
    import camelot  # optional
    HAS_CAMELOT = True
except Exception:
    HAS_CAMELOT = False

def _save_image(pix, stem: str, idx: int) -> str:
    img_path = Path(f"outputs/media/{stem}_img_{idx}.png")
    pix.save(img_path.as_posix())
    return img_path.as_posix()

def _two_means_threshold(xs: List[float]) -> Optional[float]:
    """Простейший k=2 clustering по x-центрам для разделения колонок (без
sklearn)."""
    if len(xs) < 8:  # мало блоков — вероятно, одна колонка
        return None
    x = np.array(xs, dtype=float).reshape(-1, 1)
    c1, c2 = np.min(x), np.max(x)
    for _ in range(12):
        d1 = np.abs(x - c1)
        d2 = np.abs(x - c2)
        g1 = x[d1 <= d2]; g2 = x[d2 < d1]
        nc1 = np.mean(g1) if len(g1) else c1
        nc2 = np.mean(g2) if len(g2) else c2
        if abs(nc1 - c1) < 1e-2 and abs(nc2 - c2) < 1e-2:
            break
        c1, c2 = nc1, nc2
    thr = float((c1 + c2) / 2.0)
    # убедимся, что разделение осмысленно
    left = (x < thr).sum(); right = (x >= thr).sum()
    if left >= 3 and right >= 3:
        return thr
    return None

def _extract_text_blocks(page: fitz.Page):
    # blocks: (x0,y0,x1,y1, text, block_no, block_type, ...)
    blocks = page.get_text("blocks")
    txt_blocks = []
    for b in blocks:

```

```

        if len(b) >= 5 and isinstance(b[4], str) and b[4].strip():
            x0, y0, x1, y1, text = b[:5]
            txt_blocks.append((x0, y0, x1, y1, text))
    return txt_blocks

def parse_pdf(
    path: str,
    category: Optional[str] = None,
    keep_images: bool = True,
    table_extraction: bool = True,
    ocr_fallback: bool = True,
) -> List[DocElement]:
    doc = fitz.open(path)
    stem = Path(path).stem
    out: List[DocElement] = []
    doc_id = stem
    order = 0

    for pno in range(len(doc)):
        page = doc[pno]
        width, height = page.rect.width, page.rect.height
        blocks = _extract_text_blocks(page)
        # Попытка детекта 2-х колонок
        centers = [ (b[0]+b[2])/2 for b in blocks ]
        thr = _two_means_threshold(centers) if centers else None

        def emit_par(text, bbox):
            nonlocal order
            out.append(DocElement(
                category=category, doc_path=path, doc_id=doc_id,
                source_type="pdf",
                page=pno+1, order=order, bbox=list(bbox),
                element_type="paragraph", text=text
            ))
            order += 1

        # Текст: либо по колонкам, либо по естественному порядку
        if thr:
            left = [b for b in blocks if (b[0]+b[2])/2 < thr]
            right = [b for b in blocks if (b[0]+b[2])/2 >= thr]
            for col in (left, right): # порядок чтения: слева направо
                col.sort(key=lambda b: (b[1], b[0]))
                for (x0,y0,x1,y1, text) in col:
                    emit_par(text.strip(), (x0,y0,x1,y1))
        else:
            # одна колонка/нехватает блоков – сортируем сверху вниз
            blocks.sort(key=lambda b: (b[1], b[0]))
            if blocks:

```

```

        for (x0,y0,x1,y1, text) in blocks:
            emit_par(text.strip(), (x0,y0,x1,y1))
    else:
        # Возможно скан: OCR (fallback)
        if ocr_fallback:
            pix = page.get_pixmap(dpi=300)
            img = Image.open(io.BytesIO(pix.tobytes("png")))
            try:
                import pytesseract
                txt = pytesseract.image_to_string(img, lang="rus+eng")
                if txt.strip():
                    emit_par(txt.strip(), (0,0,width,height))
            except Exception:
                pass

# Изображения
if keep_images:
    imgs = page.get_images(full=True)
    for i,(xref, *_rest) in enumerate(imgs):
        try:
            pix = fitz.Pixmap(doc, xref)
            if pix.n > 4: # CMYK
                pix = fitz.Pixmap(fitz.csRGB, pix)
            mpath = _save_image(pix, stem=f"{stem}_p{pno+1}", idx=i)
            out.append(DocElement(
                category=category, doc_path=path, doc_id=doc_id,
source_type="pdf",
                page=pno+1, element_type="image", media_path=mpath,
order=order
            ))
            order += 1
        except Exception:
            continue

# Таблицы из PDF (best-effort)
if table_extraction:
    try:
        with pdfplumber.open(path) as pl:
            pg = pl.pages[pno]
            tables = pg.extract_tables() # stream-алгоритм
            for ti, table in enumerate(tables):
                if not table:
                    continue
                rows = ["\t".join([c or "" for c in row]) for row in
table]

                text = "\n".join(rows)
                out.append(DocElement(
                    category=category, doc_path=path, doc_id=doc_id,

```

```

source_type="pdf",
                                page=pno+1, element_type="table", text=text,
order=order
        ))
        order += 1
    except Exception:
        pass

    if HAS_CAMELOT:
        try:
            # Camelot сам определит страницы с сеткой
            import camelot
            tables = camelot.read_pdf(path, pages=str(pno+1),
flavor="lattice")

            for i, t in enumerate(tables):
                csv_text = t.df.to_csv(index=False)
                out.append(DocElement(
                    category=category, doc_path=path, doc_id=doc_id,
source_type="pdf",
                                page=pno+1, element_type="table", text=csv_text,
order=order,
                                metadata=
{"extractor": "camelot", "shape": list(t.df.shape)})
                ))
                order += 1
        except Exception:
            pass

    return out

```

## 7) DOCX: текст, таблицы, формулы (OMML)

Python

```

# parsers/docx_parser.py
from typing import List, Optional
from models import DocElement
from pathlib import Path
from docx import Document
from lxml import etree

OMML_NS = {"m": "http://schemas.openxmlformats.org/officeDocument/2006/math"}

def parse_docx(path: str, category: Optional[str]=None) -> List[DocElement]:

```



```

doc = Document(path)
stem = Path(path).stem
out: List[DocElement] = []
doc_id = stem
order = 0

# Параграфы и списки
for para in doc.paragraphs:
    text = para.text.strip()
    if text:
        out.append(DocElement(
            category=category, doc_path=path, doc_id=doc_id,
source_type="docx",
            element_type="paragraph", text=text, order=order
        ))
        order += 1

# Таблицы
for tbl in doc.tables:
    for row in tbl.rows:
        cells = [c.text.replace("\n", " ").strip() for c in row.cells]
        out.append(DocElement(
            category=category, doc_path=path, doc_id=doc_id,
source_type="docx",
            element_type="table", text="\t".join(cells), order=order
        ))
        order += 1

# Формулы (OMML) — низкоуровневый доступ к XML
root = doc.element.body._tc.getparent().getroottree() # XML дерево документа
xml = root.getroot()
for node in xml.xpath("./m:oMath", namespaces=OMML_NS):
    xml_str = etree.tostring(node, encoding="unicode")
    out.append(DocElement(
        category=category, doc_path=path, doc_id=doc_id, source_type="docx",
        element_type="formula", html=xml_str, order=order
    ))
    order += 1

# Изображения — опционально: doc.part.related_parts
for r_id, part in doc.part.related_parts.items():
    if part.partname and part.partname.ext.lower() in
[".png", ".jpg", ".jpeg", ".bmp"]:
        # Можно сохранить бинарник; тут пропустим для компактности
        pass

return out

```

## 8) PPTX: слайды, текст, таблицы

Python

```
# parsers/pptx_parser.py
from typing import List, Optional
from models import DocElement
from pathlib import Path
from pptx import Presentation

def parse_pptx(path: str, category: Optional[str]=None) -> List[DocElement]:
    prs = Presentation(path)
    stem = Path(path).stem
    out: List[DocElement] = []
    doc_id = stem
    order = 0

    for idx, slide in enumerate(prs.slides, start=1):
        # Заголовки и текстовые фреймы
        for shape in slide.shapes:
            if hasattr(shape, "has_text_frame") and shape.has_text_frame:
                txt = shape.text.strip()
                if txt:
                    out.append(DocElement(
                        category=category, doc_path=path, doc_id=doc_id,
source_type="pptx",
                        element_type="paragraph", text=txt, order=order,
                        metadata={"slide": idx}
                    ))
                    order += 1
            if hasattr(shape, "has_table") and shape.has_table:
                for row in shape.table.rows:
                    cells = [c.text.replace("\n", " ").strip() for c in row.cells]
                    out.append(DocElement(
                        category=category, doc_path=path, doc_id=doc_id,
source_type="pptx",
                        element_type="table", text="\t".join(cells), order=order,
                        metadata={"slide": idx}
                    ))
                    order += 1
        # Отметим границу слайда (метаданные)
        out.append(DocElement(
            category=category, doc_path=path, doc_id=doc_id, source_type="pptx",
            element_type="slide", text=f"--- Слайд {idx} ---", order=order
        ))
        order += 1
```

```
return out
```

## 9) Excel/CSV

Python

```
# parsers/excel_csv_parser.py
from typing import List, Optional
from models import DocElement
from pathlib import Path
import pandas as pd
import csv

def parse_csv(path: str, category: Optional[str]=None) -> List[DocElement]:
    out: List[DocElement] = []
    doc_id = Path(path).stem
    order = 0
    with open(path, newline="", encoding="utf-8") as f:
        reader = csv.reader(f)
        headers = next(reader, None)
        for row in reader:
            if headers:
                text = "; ".join(f"{h}: {v}" for h, v in zip(headers, row))
            else:
                text = ", ".join(row)
            out.append(DocElement(
                category=category, doc_path=path, doc_id=doc_id,
                source_type="csv",
                element_type="row", text=text, order=order
            ))
            order += 1
    return out

def parse_excel(path: str, category: Optional[str]=None) -> List[DocElement]:
    out: List[DocElement] = []
    xls = pd.ExcelFile(path)
    doc_id = Path(path).stem
    order = 0
    for sheet in xls.sheet_names:
        df = pd.read_excel(xls, sheet)
        for _, row in df.iterrows():
            text = "; ".join(f"{col}: {row[col]}" for col in df.columns)
            out.append(DocElement(
                category=category, doc_path=path, doc_id=doc_id,
                source_type="xlsx",
                element_type="row", text=str(text), order=order,
```

```
        metadata={"sheet": sheet}
    ))
    order += 1
return out
```

## 10) TXT

Python

```
# parsers/txt_parser.py
from typing import List, Optional
from models import DocElement
from pathlib import Path

def parse_txt(path: str, category: Optional[str]=None) -> List[DocElement]:
    with open(path, "r", encoding="utf-8") as f:
        text = f.read()
    doc_id = Path(path).stem
    return [DocElement(
        category=category, doc_path=path, doc_id=doc_id, source_type="txt",
        element_type="paragraph", text=text, order=0
    )]
```

## 11) PostgreSQL как источник

Python

```
# parsers/pg_loader.py
from typing import List
from models import DocElement
import psycopg2

def load_from_pg(dsn: str, query: str, category: str="DB") -> List[DocElement]:
    con = psycopg2.connect(dsn)
    cur = con.cursor()
    cur.execute(query)
    rows = cur.fetchall()
    out = []
    order = 0
    for r in rows:
        # адаптируйте под свои колонки
        # пример: (id, title, content)
        if len(r) == 3:
```

```

        rid, title, content = r
        text = f"{title}\n{content}"
    else:
        text = " | ".join(str(x) for x in r)
    out.append(DocElement(
        category=category, doc_path="postgres", doc_id=str(r[0]),
        source_type="pg", element_type="paragraph", text=text, order=order
    ))
    order += 1
cur.close(); con.close()
return out

```

## 12) Построение иерархического графа (категория → документ → элементы)

Python

```

# graph_builder.py
import networkx as nx
from typing import List, Dict
from models import DocElement
from pathlib import Path
import json

def build_graph(elements_by_doc: Dict[str, List[DocElement]]) -> nx.DiGraph:
    G = nx.DiGraph()
    for doc_key, elems in elements_by_doc.items():
        # doc_key: "{category}/{doc_path}"
        cat, doc_path = doc_key.split("::", 1)
        cat_id = f"cat::{cat}"
        doc_id = f"doc::{doc_path}"
        G.add_node(cat_id, type="category", name=cat)
        G.add_node(doc_id, type="document", path=doc_path)
        G.add_edge(cat_id, doc_id, rel="contains")
        for el in elems:
            G.add_node(el.id, type=el.element_type, page=el.page, order=el.order)
            G.add_edge(doc_id, el.id, rel="has_element")
    return G

def save_graph(G: nx.DiGraph, name="corpus_graph"):
    nx.write_graphml(G, f"outputs/graphs/{name}.graphml")
    nx.write_gexf(G, f"outputs/graphs/{name}.gexf")
    data = nx.readwrite.json_graph.node_link_data(G)
    with open(f"outputs/graphs/{name}.json", "w", encoding="utf-8") as f:

```

```
json.dump(data, f, ensure_ascii=False, indent=2)
```

## 13) Запуск пайплайна

Python

```
# main.py
import os, sys
from typing import Dict, List
from tqdm import tqdm
from pathlib import Path

from config import PipelineConfig
from models import DocElement
from utils_io import ensure_dirs, dump_jsonl
from graph_builder import build_graph, save_graph

from parsers.pdf_parser import parse_pdf
from parsers.docx_parser import parse_docx
from parsers.pptx_parser import parse_pptx
from parsers.excel_csv_parser import parse_excel, parse_csv
from parsers.txt_parser import parse_txt
from parsers.pg_loader import load_from_pg

SUPPORTED = {".pdf", ".docx", ".pptx", ".xlsx", ".xls", ".csv", ".txt"}

def route_parse(path: str, cat: str, cfg: PipelineConfig) -> List[DocElement]:
    ext = Path(path).suffix.lower()
    if ext == ".pdf":
        return parse_pdf(
            path, category=cat,
            keep_images=cfg.keep_images,
            table_extraction=cfg.pdf_table_extraction,
            ocr_fallback=cfg.pdf_ocr_fallback
        )
    if ext == ".docx":
        return parse_docx(path, category=cat)
    if ext == ".pptx":
        return parse_pptx(path, category=cat)
    if ext in (".xlsx", ".xls"):
        return parse_excel(path, category=cat)
    if ext == ".csv":
        return parse_csv(path, category=cat)
    if ext == ".txt":
        return parse_txt(path, category=cat)
    return []
```



```

def run_pipeline(cfg: PipelineConfig):
    ensure_dirs()
    all_elements: List[DocElement] = []
    per_doc: Dict[str, List[DocElement]] = {}

    # 1) Файлы по категориям
    for cat, files in cfg.categories.items():
        for f in tqdm(files, desc=f"[{cat}]"):
            try:
                elems = route_parse(f, cat, cfg)
                all_elements.extend(elems)
                key = f"{cat}::{f}"
                per_doc[key] = elems
            except Exception as e:
                print(f"ERROR {f}: {e}")

    # 2) Источник из PostgreSQL (опционально)
    if cfg.pg_dsn and cfg.pg_query:
        db_elems = load_from_pg(cfg.pg_dsn, cfg.pg_query, category="DB")
        all_elements.extend(db_elems)
        per_doc["DB::postgres"] = db_elems

    # 3) Сохранение
    dump_jsonl(all_elements, "outputs/dataset/elements.jsonl")
    G = build_graph(per_doc)
    save_graph(G, name="corpus_graph")
    print(f"OK: {len(all_elements)} элементов. JSONL:
outputs/dataset/elements.jsonl")

if __name__ == "__main__":
    # Пример вашего словаря категорий
    categories = {
        'Нерудные_материалы': [
            'doc/Нерудные_материалы/Условия поставки нерудных материалов
железнодорожным транспортом (от 17.06.2025).pdf',
            'doc/Нерудные_материалы/Общие условия к Договору поставки нерудных
материалов (от 16.04.2024).pdf',
            # ...
        ],
        'Железобетонные_изделия': [
            'doc/Железобетонные_изделия/Условия поставки автомобильным транспортом
(от 25.12.2024).pdf',
            # ...
        ],
        # добавьте остальные
    }

```

```
cfg = PipelineConfig(
    categories=categories,
    pdf_ocr_fallback=True,
    pdf_table_extraction=True,
    keep_images=True,
    pg_dsn=None, # при необходимости
    pg_query=None
)
run_pipeline(cfg)
```

## 14) Что вы получаете на выходе

- `outputs/dataset/elements.jsonl` — унифицированные элементы (параграфы/таблицы/формулы/изображения) с порядком чтения, bbox (для PDF), метаданными и категорией. Это готовая матрица для **эмбединга** и дальнейшего индексирования (FAISS/Qdrant/Minima).
- `outputs/media/*` — извлечённые изображения (в т.ч. формулы/диаграммы, если они были картинками).
- `outputs/graphs/corpus_graph.(graphml|gexf|json)` — иерархический граф **категория → документ → элемент**, пригодный для навигации/визуализации/политик доступа.

## 15) Как готовить к эмбедингам (намёк на следующий шаг)

Дальше обычно делается **семантическое чанкинг-правило** (по заголовкам/секциям/абзацам, не «каждые N символов»), затем — эмбединг локальной моделью (`sentence-transformers` многоязычные для RU), сохранение во **FAISS/Qdrant**. Формулу/изображение можно сопроводить подписью и OCR-текстом, чтобы не терять смысл.

Если хотите — в следующем сообщении добавлю **модуль эмбединга + сохранение в FAISS/Qdrant** и экспорт формата, совместимого с вашим референтом (Minima).

11 Aug 2025, 8:43 pm

```
TypeError Traceback (most recent call last) Cell In[1], line 7
4 from tqdm import
tqdm
5 from pathlib import Path
----> 7 from config import PipelineConfig
8 from
models import DocElement
9 from utils_io import ensure_dirs, dump_jsonl
```

File

```
~/Documents/Jupyter_notebooks/Cementum/RAG_ULTRA_v0.1/doc_ingest/config.py:6  
2 from typing import Dict, List, Optional 3 from dataclasses import dataclass, field  
5 @dataclass ----> 6 class PipelineConfig: 7 # входы 8 categories: Dict[str,  
List[str]] # {"Категория": ["doc/...pdf", ...], ...} 9 # postgres как источник  
(опционально)
```

```
File /opt/anaconda3/envs/corp-rag/lib/python3.10/dataclasses.py:1184, in  
dataclass(cls, init, repr, eq, order, unsafe_hash, frozen, match_args, kw_only, slots)  
1181 return wrap 1183 # We're called as @dataclass without parens. -> 1184  
return wrap(cls)
```

```
File /opt/anaconda3/envs/corp-rag/lib/python3.10/dataclasses.py:1175, in  
dataclass.<locals>.wrap(cls) 1174 def wrap(cls): -> 1175 return  
_process_class(cls, init, repr, eq, order, unsafe_hash, 1176 frozen, match_args,  
kw_only, slots)
```

```
File /opt/anaconda3/envs/corp-rag/lib/python3.10/dataclasses.py:1024, in  
_process_class(cls, init, repr, eq, order, unsafe_hash, frozen, match_args, kw_only,  
slots) 1019 if init: 1020 # Does this class have a post-init function? 1021  
has_post_init = hasattr(cls, _POST_INIT_NAME) 1023 _set_new_attribute(cls, 'init',  
-> 1024 _init_fn(all_init_fields, 1025 std_init_fields, 1026 kw_only_init_fields, 1027  
frozen, 1028 has_post_init, 1029 # The name to use for the "self" 1030 # param in  
init. Use "self" 1031 # if possible. 1032 'dataclass_self' if 'self' in fields 1033 else  
'self', 1034 globals, 1035 slots, 1036 )) 1038 # Get the fields as a list, and include  
only real fields. This is 1039 # used in all of the following methods. 1040 field_list  
= [f for f in fields.values() if f._field_type is _FIELD]
```

```
File /opt/anaconda3/envs/corp-rag/lib/python3.10/dataclasses.py:544, in  
_init_fn(fields, std_fields, kw_only_fields, frozen, has_post_init, self_name, globals,  
slots) 542 seen_default = True 543 elif seen_default: -> 544 raise  
TypeError(f'non-default argument {f.name!r} ' 545 'follows default argument') 547  
locals = {f'type{f.name}': f.type for f in fields} 548 locals.update({ 549 'MISSING':  
MISSING, 550 '_HAS_DEFAULT_FACTORY': _HAS_DEFAULT_FACTORY, 551  
'dataclass_builtins_object': object, 552 })
```

TypeError: non-default argument 'keep\_formulas' follows default argument



11 Aug 2025, 8:43 pm • Model: gpt-5-thinking

Спасибо за лог — причина понятна.

## Почему падает

В `dataclass` все поля **без значения по умолчанию** должны идти **до** полей с **умолчаниями**. У тебя поле `keep_formulas` записано как `keep_formulas: True` — это **аннотация без типа и без default**, поэтому интерпретируется как *не-дефолтное поле после дефолтных*, что и даёт:

Text

```
TypeError: non-default argument 'keep_formulas' follows default argument
```

## Правка `config.py`

Замени содержимое на это (исправил тип и default, порядок полей сохранён корректный):

Python

```
# config.py
from typing import Dict, List, Optional
from dataclasses import dataclass

@dataclass
class PipelineConfig:
    # входы (без дефолтов — первыми)
    categories: Dict[str, List[str]] # {"Категория": ["doc/..pdf", ...], ...}

    # postgres как источник (опционально)
    pg_dsn: Optional[str] = None      # "postgresql://user:pwd@host:5432/db"
    pg_query: Optional[str] = None    # "select id, title, content from documents"

    # парсинг
    pdf_ocr_fallback: bool = True     # OCR для сканов
    pdf_table_extraction: bool = True # попытка вытаскивать таблицы
    keep_images: bool = True          # сохранять изображения
    keep_formulas: bool = True        # <-- было 'keep_formulas: True' (ошибка)

    # chunk/порядок
    max_chars_per_element: int = 15000

    # производительность
    workers: int = 4
```

Перезапусти ячейку/скрипт — ошибка пропадёт. Если поймаешь следующую — кидай трейс, поправлю оперативно.

Итак переходим к следующему этапу вот что получилось: 14) Что вы получаете на выходе • `outputs/dataset/elements.jsonl` — унифицированные элементы (параграфы/таблицы/формулы/изображения) с порядком чтения, `bbox` (для PDF), метаданными и категорией. Это готовая матрица для эмбединга и дальнейшего индексирования (FAISS/Qdrant/Minima). • `outputs/media/*` — извлечённые изображения (в т.ч. формулы/диаграммы, если они были картинками). • `outputs/graphs/corpus_graph.(graphml|gexf|json)` — иерархический граф категория → документ → элемент, пригодный для навигации/визуализации/политик доступа.

📌 Готовимся к эмбедингам нам нужна локальная эмбединговая модель, Нельзя чанки просто нарезать нужно умное разделение на чанки!! 15) Как готовить к эмбедингам (намёк на следующий шаг)

Дальше обычно делается семантическое чанкинг-правило (по заголовкам/секциям/абзацам, не «каждые N символов»), затем — эмбединг локальной моделью (sentence-transformers многоязычные для RU), сохранение во FAISS/Qdrant. Формулу/изображение можно сопроводить подписью и OCR-текстом, чтобы не терять смысл.



11 Aug 2025, 9:05 pm • Model: gpt-5-thinking

Отлично, идём дальше: **локальная модель эмбедингов + умный семантический чанкинг** (без тупого N-символьного реза), с поддержкой таблиц/формул/изображений и сохранением в **FAISS** и/или **Qdrant**. Всё — on-prem, open-source.

## 0) Установка (добавить к уже установленному)

Bash

```
pip install sentence-transformers transformers faiss-cpu qdrant-client razdel
```

## 1) Конфиг для эмбедингов

Добавь во **вчерашний** `config.py` второй датакласс (не ломает парсер):

Python

```

# config.py (добавить)
from dataclasses import dataclass
from typing import Optional

@dataclass
class EmbeddingConfig:
    # модель эмбедингов (по умолчанию — высокое качество, RU-friendly)
    model_name: str = "intfloat/multilingual-e5-base" # альтернатива:
"sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
    device: str = "auto" # "cuda", "cpu", "auto"

    # правила семантического чанкинга
    chunk_target_tokens: int = 350
    chunk_max_tokens: int = 512
    min_chunk_tokens: int = 64
    sentence_overlap: int = 1
    cohesion_split: bool = True # разбиение по падению семантической
СВЯЗНОСТИ
    heading_aware: bool = True # использовать заголовки/номера разделов
    table_as_is: bool = True # таблица — отдельный чанк, но с
контекстом подписи/заголовка

    # индексация
    index_kind: str = "both" # "faiss" | "qdrant" | "both"
    faiss_out: str = "outputs/index/faiss.index"
    meta_out: str = "outputs/index/meta.jsonl"

    # Qdrant
    qdrant_host: str = "127.0.0.1"
    qdrant_port: int = 6333
    qdrant_collection: str = "cementum_docs"

```

## 2) Модель данных для чанков

Добавь в `models.py` :

Python

```

# models.py (добавить)
from typing import List, Optional
from pydantic import BaseModel, Field
from uuid import uuid4

class Chunk(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid4()))

```



```
category: Optional[str] = None
doc_id: Optional[str] = None
doc_path: Optional[str] = None
text: str
token_len: int
char_len: int
chunk_index: int
from_order: int
to_order: int
page_from: Optional[int] = None
page_to: Optional[int] = None
heading_path: Optional[List[str]] = None
element_types: Optional[List[str]] = None
media_refs: Optional[List[str]] = None
```

## 3) Умный семантический чанкинг

Создай chunking.py :

Python

```
# chunking.py
import json, re
from pathlib import Path
from typing import List, Dict, Tuple, Iterable, Optional
import numpy as np
from razdel import sentenize
from transformers import AutoTokenizer
from sentence_transformers import SentenceTransformer
from models import DocElement, Chunk
from config import EmbeddingConfig

_HEADING_RE = re.compile(
    r"^((раздел|глава|приложение)\s+\d+(\.\\.)?|(\d+(\.\\d+){0,3})(\\|\\.|\\.)?)\s+.\+",
    flags=re.IGNORECASE
)

def _is_heading(text: str) -> bool:
    t = text.strip()
    if len(t) <= 3:
        return False
    if _HEADING_RE.match(t):
        return True
    # короткая ВЕРХНИМ РЕГИСТРОМ строка как заголовок
    if len(t) < 120 and t == t.upper() and re.search(r"[А-ЯА-З]{3,}", t):
        return True
```

```
return False
```

```
def _token_count(tokenizer, text: str) -> int:  
    return len(tokenizer.encode(text, add_special_tokens=False))
```

```
def _prefix_for_model(model_name: str) -> str:  
    # E5 требует префиксы: "passage: " для документов  
    return "passage: " if "e5" in model_name.lower() else ""
```

```
def _sentences(text: str) -> List[str]:  
    return [s.text.strip() for s in sentenize(text) if s.text.strip()]
```

```
def _adjacent_sim_drop(embs: np.ndarray) -> List[int]:  
    """Ищем места падения связности: низкая косинусная близость между соседними предложениями."""  
    if len(embs) < 3:  
        return []  
    embs = embs / (np.linalg.norm(embs, axis=1, keepdims=True) + 1e-12)  
    sims = (embs[:-1] * embs[1:]).sum(axis=1) # косинус соседних  
    # кандидаты на разрыв – ниже квантиля (25%)  
    thr = float(np.quantile(sims, 0.25))  
    cuts = [i+1 for i, s in enumerate(sims) if s <= thr]  
    return cuts
```

```
def build_chunks(  
    elements: List[DocElement],  
    cfg: EmbeddingConfig,  
) -> List[Chunk]:  
    """Heading-aware + cohesion-aware чанкинг."""  
    Path("outputs/index").mkdir(parents=True, exist_ok=True)  
  
    tok = AutoTokenizer.from_pretrained(cfg.model_name)  
    model = SentenceTransformer(cfg.model_name)  
    prefix = _prefix_for_model(cfg.model_name)  
  
    chunks: List[Chunk] = []  
    cur_buf: List[Tuple[DocElement, str]] = [] # (element, text_part)  
    cur_tokens = 0  
    chunk_index = 0  
    heading_path: List[str] = []  
    media_refs: List[str] = []  
  
    def flush_buf():  
        nonlocal chunks, cur_buf, cur_tokens, chunk_index, media_refs  
        if not cur_buf:  
            return  
        texts = [t for _, t in cur_buf if t.strip()]  
        if not texts:
```

```

    cur_buf, cur_tokens, media_refs = [], 0, []
    return

full = "\n".join(texts)
# если cohesion_split включён — внутри буфера ещё делим по падениям
СВЯЗНОСТИ
final_texts = []
if cfg.cohesion_split:
    sents = []
    pos = []
    for el, t in cur_buf:
        ss = _sentences(t)
        if not ss:
            continue
        sents.extend(ss)
        pos.extend([el] * len(ss))
    if len(sents) > 2:
        s_embs = model.encode([prefix + s for s in sents],
                                convert_to_numpy=True,
                                normalize_embeddings=True,
                                batch_size=64, show_progress_bar=False)
        cuts = set(_adjacent_sim_drop(s_embs))
        # собираем сегменты по cuts, с учётом бюджетов токенов
        start = 0
        while start < len(sents):
            end = start
            buf_txt = []
            while end < len(sents):
                if end in cuts and buf_txt:
                    break
                candidate = "\n".join(buf_txt + [sents[end]])
                if _token_count(tok, candidate) > cfg.chunk_max_tokens:
                    break
                buf_txt.append(sents[end])
                end += 1
            final_texts.append("\n".join(buf_txt))
            start = end
        else:
            final_texts = [full]
    else:
        final_texts = [full]

# ЭМИТИМ ЧАНКИ
orders = [el.order for el, _ in cur_buf if isinstance(el.order, int)]
pages = [el.page for el, _ in cur_buf if isinstance(el.page, int)]
element_types = list({el.element_type for el, _ in cur_buf})
cat = cur_buf[0][0].category
doc_id = cur_buf[0][0].doc_id
doc_path = cur_buf[0][0].doc_path

```

```

for text in final_texts:
    if not text.strip():
        continue
    tokens = _token_count(tok, text)
    if tokens < cfg.min_chunk_tokens and chunks:
        # слить с предыдущим, если слишком мелкий
        prev = chunks[-1]
        merged = prev.text + "\n" + text
        if _token_count(tok, merged) <= cfg.chunk_max_tokens:
            prev.text = merged
            prev.token_len = _token_count(tok, prev.text)
            prev.char_len = len(prev.text)
            prev.to_order = max(prev.to_order, max(orders) if orders else
prev.to_order)
            prev.page_to = max(prev.page_to or 0, max(pages) if pages else
(prev.page_to or 0)) or None
            continue
        chunks.append(Chunk(
            category=cat,
            doc_id=doc_id,
            doc_path=doc_path,
            text=text,
            token_len=tokens,
            char_len=len(text),
            chunk_index=chunk_index,
            from_order=min(orders) if orders else 0,
            to_order=max(orders) if orders else 0,
            page_from=min(pages) if pages else None,
            page_to=max(pages) if pages else None,
            heading_path=heading_path[:] if heading_path else None,
            element_types=element_types,
            media_refs=media_refs[:] if media_refs else None
        ))
        chunk_index += 1

    cur_buf, cur_tokens, media_refs = [], 0, []

for el in elements:
    # обновляем heading_path
    if cfg.heading_aware and el.element_type in ("paragraph", "title") and
el.text:
        if _is_heading(el.text):
            # новый раздел — сбрасываем буфер перед сменой заголовка
            flush_buf()
            # эвристика уровня: количество уровней нумерации
            m = re.match(r"^(\\d+(?:\\.\\d+)){0,3})", el.text.strip())
            depth = (len(m.group(1).split(".")) if m else 1) - 1

```

```

        if depth < 0: depth = 0
        if len(heading_path) > depth:
            heading_path = heading_path[:depth]
        heading_path.append(el.text.strip())
        continue

# таблица — отдельный чанк (но с контекстом)
if el.element_type == "table" and cfg.table_as_is:
    flush_buf()
    tbl_text = el.text or (el.html or "")
    # префиксируем подписью раздела
    if heading_path:
        tbl_text = f"{' > '.join(heading_path)}\n{tbl_text}"
    # emit как самостоятельный буфер
    cur_buf = [(el, tbl_text)]
    flush_buf()
    continue

# изображения/формулы — соберём ссылки, а текст берём из OCR/подписей (если
были)
if el.element_type in ("image", "formula") and el.media_path:
    media_refs.append(el.media_path)

piece = (el.text or el.html or "").strip()
if not piece:
    continue

piece_tokens = _token_count(AutoTokenizer.from_pretrained(cfg.model_name),
piece)

# если переполняем буфер — сбрасываем
if cur_tokens + piece_tokens > cfg.chunk_target_tokens:
    flush_buf()
    cur_buf.append((el, piece))
    cur_tokens += piece_tokens

flush_buf()
return chunks

```

## 4) Индексация: FAISS

embed\_faiss.py :

Python

```

# embed_faiss.py
import json

```

```

from pathlib import Path
from typing import List
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
from models import Chunk
from config import EmbeddingConfig

def embed_and_build_faiss(chunks: List[Chunk], cfg: EmbeddingConfig):
    Path("outputs/index").mkdir(parents=True, exist_ok=True)
    model = SentenceTransformer(cfg.model_name)
    prefix = "passage: " if "e5" in cfg.model_name.lower() else ""
    texts = [prefix + c.text for c in chunks]

    vecs = model.encode(
        texts, convert_to_numpy=True, normalize_embeddings=True,
        batch_size=128, show_progress_bar=True
    )
    dim = vecs.shape[1]
    index = faiss.IndexFlatIP(dim) # т.к. вектора нормированы — IP == cosine
    index.add(vecs)

    faiss.write_index(index, cfg.faiss_out)

    with open(cfg.meta_out, "w", encoding="utf-8") as f:
        for c in chunks:
            f.write(json.dumps({
                "id": c.id,
                "category": c.category,
                "doc_id": c.doc_id,
                "doc_path": c.doc_path,
                "chunk_index": c.chunk_index,
                "text": c.text,
                "token_len": c.token_len,
                "heading_path": c.heading_path,
                "page_from": c.page_from,
                "page_to": c.page_to,
                "element_types": c.element_types,
                "media_refs": c.media_refs
            }, ensure_ascii=False) + "\n")
    print(f"FAISS index saved to {cfg.faiss_out}; meta -> {cfg.meta_out}")

```

## 5) Индексация: Qdrant (локальный)

embed\_qdrant.py :

```

# embed_qdrant.py
from typing import List
from qdrant_client import QdrantClient
from qdrant_client.models import Distance, VectorParams, PointStruct
from sentence_transformers import SentenceTransformer
from models import Chunk
from config import EmbeddingConfig

def push_to_qdrant(chunks: List[Chunk], cfg: EmbeddingConfig):
    client = QdrantClient(host=cfg.qdrant_host, port=cfg.qdrant_port)
    model = SentenceTransformer(cfg.model_name)
    prefix = "passage: " if "e5" in cfg.model_name.lower() else ""

    dim = model.get_sentence_embedding_dimension()
    # коллекция
    if cfg.qdrant_collection not in [c.name for c in
client.get_collections().collections]:
        client.recreate_collection(
            collection_name=cfg.qdrant_collection,
            vectors_config=VectorParams(size=dim, distance=Distance.COSINE)
        )

    # батчево
    B = 256
    for i in range(0, len(chunks), B):
        batch = chunks[i:i+B]
        texts = [prefix + c.text for c in batch]
        vecs = model.encode(texts, convert_to_numpy=True,
normalize_embeddings=True,
                                batch_size=128, show_progress_bar=False)

        points = []
        for c, v in zip(batch, vecs):
            payload = {
                "id": c.id,
                "category": c.category,
                "doc_id": c.doc_id,
                "doc_path": c.doc_path,
                "chunk_index": c.chunk_index,
                "text": c.text,
                "token_len": c.token_len,
                "heading_path": c.heading_path,
                "page_from": c.page_from,
                "page_to": c.page_to,
                "element_types": c.element_types,
                "media_refs": c.media_refs
            }

```

```
        points.append(PointStruct(id=c.id, vector=v.tolist(),
payload=payload))
        client.upsert(collection_name=cfg.qdrant_collection, points=points)
    print(f"Qdrant upsert OK -> collection '{cfg.qdrant_collection}'")
```

## 6) Оркестрация шага эмбедингов

main\_embed.py :

Python

```
# main_embed.py
import json
from pathlib import Path
from typing import List
from tqdm import tqdm

from config import EmbeddingConfig
from models import DocElement, Chunk
from chunking import build_chunks
from embed_faiss import embed_and_build_faiss
from embed_qdrant import push_to_qdrant

def load_elements(jsonl_path: str) -> List[DocElement]:
    items = []
    with open(jsonl_path, "r", encoding="utf-8") as f:
        for line in f:
            d = json.loads(line)
            items.append(DocElement(**d))
    return items

def group_by_document(elements: List[DocElement]):
    buckets = {}
    for el in elements:
        key = (el.category or "UNK", el.doc_id or el.doc_path or "doc")
        buckets.setdefault(key, []).append(el)
    # сортируем по порядку для стабильности
    for k in buckets:
        buckets[k].sort(key=lambda x: (x.page or 0, x.order or 0))
    return buckets

def run():
    cfg = EmbeddingConfig()
    Path("outputs/index").mkdir(parents=True, exist_ok=True)

    elements = load_elements("outputs/dataset/elements.jsonl")
```



```

buckets = group_by_document(elements)

all_chunks: List[Chunk] = []
for (cat, did), els in tqdm(buckets.items(), desc="Chunking per document"):
    ch = build_chunks(els, cfg)
    all_chunks.extend(ch)

print(f"Chunks total: {len(all_chunks)}")

if cfg.index_kind in ("faiss", "both"):
    embed_and_build_faiss(all_chunks, cfg)
if cfg.index_kind in ("qdrant", "both"):
    push_to_qdrant(all_chunks, cfg)

if __name__ == "__main__":
    run()

```

## Что это даёт по факту

- **Heading-aware:** разбиение учитывает заголовки/номерные разделы, не рубит середину раздела.
- **Cohesion-aware:** внутри раздела ищем падения семантической связности (косинус соседних предложений) — ставим границы там, где логично.
- **Budget-aware:** `target/max tokens`, с слиянием слишком мелких фрагментов.
- **Tables-aware:** таблицы — отдельные чанки с контекстом заголовка.
- **Images/Formulas:** в чанке сохраняются `media_refs`, можно подключить OCR/MathML-нормализацию отдельно.
- **Index-ready:** FAISS (IP с нормированием) и/или Qdrant (COSINE). Полезные payload-метаданные: `category`, `doc_id`, `heading_path`, `page_*`, `element_types`.

## Рекомендации по моделям (локально)

- **Качество (RU, multi):** `intfloat/multilingual-e5-base` — хороший баланс. (Помни формат `"passage: ..."` / `"query: ..."`)
- **Лёгкая/быстрая:** `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2`.
- **Большая:** `intfloat/multilingual-e5-large` (если есть GPU/память).

Переключается одной строкой в `EmbeddingConfig.model_name`.

## Дальше (если нужно)

- Экспорт схемы под ваш референт Minima (Qdrant уже совместим).
- Query-путь: `encode("query: {text}")` → поиск top-k по FAISS/Qdrant → перанкер (локальный, например, `BAAI/bge-reranker-base` ).
- Политики доступа: фильтрация по `category` / `doc_id` / `heading_path` на уровне `payload`.

Если где-то упрёмся в производительность/качество — скажи, докрутим (GPU batching, aggressive dedup, доп. правила выделения заголовков для PDF через размеры шрифта PyMuPDF, автоконтекст таблиц и пр.).



Convert your ChatGPT to PDF with **pdfchatgpt.com** for free