

Алгоритм загрузки Word-файлов с текстом, таблицами и изображениями в векторную базу данных

1. Извлечение содержимого из Word-файлов

Для работы с Word-документами (docx) наилучший результат дают специализированные Python-библиотеки, такие как `Spire.Doc`, `python-docx` или инструменты от [Unstructured.io](https://unstructured.io):

- **Текст:** Извлекается с помощью методов библиотеки (например, `GetText()` в `Spire.Doc`)^[1].
- **Таблицы:** Извлекаются как отдельные структуры, можно преобразовать в CSV, Markdown или HTML для дальнейшей обработки^{[1] [2]}.
- **Изображения:** Извлекаются из документа, включая те, что находятся внутри таблиц или других вложенных объектов^{[1] [3]}.

```
from spire.doc import *
doc = Document()
doc.LoadFromFile("file.docx")
text = doc.GetText()
# Аналогично – извлечение таблиц и изображений
```

2. Разделение содержимого на чанки (chunking)

Правильная сегментация — ключ к качественной индексации и поиску:

- **Текст:** Делить на логические блоки по абзацам, предложениям или фиксированному количеству символов (например, 1000–2000 символов)^{[4] [5]}.
- **Таблицы:** Каждую таблицу выделять отдельным чанком, сохранять структуру (например, в Markdown или HTML)^{[5] [6] [7]}.
- **Изображения:** Для каждого изображения создавать отдельный чанк с описанием (caption) или ссылкой на расположение в исходном документе^[6].

Рекомендация: Использовать [Unstructured.io](https://unstructured.io) или аналогичные инструменты, которые автоматически разделяют документ на текстовые, табличные и графические элементы^{[6] [7]}.

3. Генерация описаний для таблиц и изображений

Чтобы мультимодальные данные были доступны для поиска:

- **Таблицы:** Генерировать краткое текстовое описание содержимого таблицы с помощью LLM (например, Phi-3-mini-128k-instruct), а саму таблицу хранить как текст (Markdown/CSV) и как embedding^[6].
- **Изображения:** Генерировать описание изображения с помощью vision-модели (например, Phi-3 Vision), embedding для поиска^[6].

4. Векторизация (embedding) и загрузка в базу

- **Текст:** Использовать современную модель эмбедингов (например, Nomic, OpenAI, Microsoft Phi-3) ^[6].
- **Таблицы:** Векторизовать либо описание, либо сериализованный текст таблицы.
- **Изображения:** Векторизовать описание изображения или саму картинку с помощью vision-модели ^[6].

Мультимодальная стратегия: Хранить отдельные векторные индексы для текста, таблиц и изображений либо использовать единую схему с метаданными для типа содержимого ^[6] ^[5].

5. Сопровождение чанков метаданными

- **Метаданные:** Для каждого чанка сохранять тип (текст, таблица, изображение), позицию в исходном документе, имя файла, номер страницы, ссылку на оригинал и т.д. ^[5] ^[6] ^[7].

6. Загрузка в векторную базу данных

- Использовать Chroma, Weaviate, Pinecone или аналогичные базы, поддерживающие мультимодальные эмбединги и хранение метаданных ^[6].
- Загружать чанки батчами, контролируя размер чанка для оптимального поиска ^[4] ^[5] ^[7].

Рекомендованный пошаговый алгоритм

1. Извлечь из каждого файла:

- Текстовые блоки (абзацы, предложения)
- Таблицы (сохранять структуру)
- Изображения (сохранять и генерировать описание)

2. Разделить на чанки:

- Текст — по логическим блокам или символам
- Таблицы — каждый объект отдельно
- Изображения — каждый объект отдельно

3. Сгенерировать описания для таблиц и изображений

4. Векторизовать все чанки подходящими моделями

5. Добавить метаданные к каждому чанку

6. Загрузить чанки с эмбедингами и метаданными в векторную базу

Практические советы

- Для многостраничных таблиц после извлечения требуется постобработка: объединять строки, разорванные на границе страниц ^[8].
- Таблицы лучше хранить как отдельные объекты с возможностью поиска по их содержимому и описанию ^[5] ^[6] ^[7].

- Изображения — хранить как отдельные чанки с embedding и caption, чтобы их можно было находить по текстовому запросу^[6].
- Используйте метаданные для связи чанков между собой и для восстановления контекста при поиске^{[5] [6] [7]}.

Сводная таблица: обработка разных типов данных

Тип содержимого	Извлечение	Чанкинг	Описание/Caption	Векторизация	Метаданные
Текст	Python-библиотеки	По смыслу/размеру	Не требуется	Text embedding	Файл, позиция
Таблицы	Python/.NET/Unstructured.io	По таблице	Генерировать LLM	Text embedding	Файл, позиция
Изображения	Python-библиотеки	По изображению	Генерировать vision LLM	Image embedding	Файл, позиция

Заключение

Для максимальной эффективности используйте автоматизированные пайплайны: извлечение → чанкинг → генерация описаний → embedding → загрузка в базу с метаданными. Это обеспечит качественный поиск по всем типам данных, даже если исходные документы неструктурированы и содержат сложные элементы^{[1] [5] [6] [7]}.

✱

Ты можешь написать скрипт на python для решения этой задачи?

Вот пример пайплайна на Python для решения вашей задачи — извлечения текста, таблиц и изображений из Word-файлов, их чанкинга, генерации описаний, векторизации и подготовки к загрузке в векторную базу данных. В скрипте используются заглушки для генерации описаний и векторизации, которые можно заменить на реальные вызовы моделей и API.

Основные этапы скрипта:

1. **Извлечение содержимого из docx:** текст, таблицы, изображения.
2. **Чанкинг текста, таблиц и изображений:** текст делится на блоки по длине, таблицы и изображения идут отдельными чанками.
3. **Генерация описаний для таблиц и изображений:** заглушки для LLM и vision-моделей.
4. **Векторизация чанков:** заглушки для embedding-моделей.
5. **Добавление метаданных и подготовка к загрузке в векторную БД.**

```
import os
from docx import Document
```

```

from io import BytesIO
from PIL import Image

# 1. Извлечение содержимого из docx
def extract_docx_content(file_path):
    doc = Document(file_path)
    text_blocks = []
    tables = []
    images = []

    for para in doc.paragraphs:
        if para.text.strip():
            text_blocks.append(para.text.strip())

    for table in doc.tables:
        table_data = []
        for row in table.rows:
            row_data = [cell.text.strip() for cell in row.cells]
            table_data.append(row_data)
        tables.append(table_data)

    for rel in doc.part.rels.values():
        if "image" in rel.target_ref:
            image_bytes = rel.target_part.blob
            image = Image.open(BytesIO(image_bytes))
            images.append(image)

    return text_blocks, tables, images

# 2. Чанкинг
def chunk_text(text_blocks, max_chars=1000):
    chunks = []
    current_chunk = ""
    for block in text_blocks:
        if len(current_chunk) + len(block) + 1 <= max_chars:
            current_chunk += (" " + block) if current_chunk else block
        else:
            chunks.append(current_chunk)
            current_chunk = block
    if current_chunk:
        chunks.append(current_chunk)
    return chunks

def chunk_tables(tables):
    def table_to_markdown(table):
        md = []
        for row in table:
            md.append("| " + " | ".join(row) + " |")
        return "\n".join(md)
    return [table_to_markdown(table) for table in tables]

def chunk_images(images):
    return images

# 3. Генерация описаний (заглушки)
def generate_table_description(table_markdown):
    return "Описание таблицы: краткое содержание данных."

```

```

def generate_image_description(image):
    return "Описание изображения: краткое описание содержимого."

# 4. Векторизация (заглушки)
def vectorize_text(text):
    return [0.1] * 768

def vectorize_table(table_desc):
    return [0.2] * 768

def vectorize_image(image_desc):
    return [0.3] * 768

# 5. Метаданные и загрузка
def add_metadata(chunks, chunk_type, file_name):
    metadata_list = []
    for i, chunk in enumerate(chunks):
        metadata = {
            "type": chunk_type,
            "file_name": file_name,
            "chunk_index": i
        }
        metadata_list.append(metadata)
    return metadata_list

def upload_to_vector_db(vectors, metadata):
    print(f"Uploading {len(vectors)} vectors with metadata to vector DB")
    return True

# Основной цикл обработки файлов
def process_folder(folder_path):
    for file in os.listdir(folder_path):
        if file.endswith(".docx"):
            file_path = os.path.join(folder_path, file)
            text_blocks, tables, images = extract_docx_content(file_path)
            text_chunks = chunk_text(text_blocks)
            table_chunks = chunk_tables(tables)
            image_chunks = chunk_images(images)

            # Текст
            text_vecs = [vectorize_text(chunk) for chunk in text_chunks]
            text_meta = add_metadata(text_chunks, "text", file)
            upload_to_vector_db(text_vecs, text_meta)

            # Таблицы
            table_descs = [generate_table_description(tc) for tc in table_chunks]
            table_vecs = [vectorize_table(desc) for desc in table_descs]
            table_meta = add_metadata(table_chunks, "table", file)
            upload_to_vector_db(table_vecs, table_meta)

            # Изображения
            image_descs = [generate_image_description(img) for img in image_chunks]
            image_vecs = [vectorize_image(desc) for desc in image_descs]
            image_meta = add_metadata(image_chunks, "image", file)
            upload_to_vector_db(image_vecs, image_meta)

# Пример запуска

```

```
# process_folder("путь_к_папке_c_docx")
```

Как использовать

- Замените заглушки генерации описаний и векторизации на реальные вызовы LLM/vision/embedding-моделей.
- Реализуйте реальную функцию загрузки в вашу векторную БД.
- Запустите `process_folder("путь_к_папке")` для обработки всех файлов.

Этот скрипт легко масштабируется и адаптируется под ваши требования.



Проверь этот скрипт, если я его запущу на ноутбуке, не зависнет?

Скрипт не должен зависать при запуске на ноутбуке, так как он обрабатывает файлы по одному, не использует параллельные вычисления и не содержит бесконечных циклов или блокирующих операций. Однако скорость работы будет зависеть от объема данных (размер и количество файлов) и мощности вашего ноутбука. При большом количестве или объеме файлов обработка может занять заметное время, но зависания ожидать не стоит.



ругается при запуске

```
PS C:\Users\mbakh\OneDrive\Рабочий стол\0питон> python ProcessFolder.py
Traceback (most recent call last):
File "C:\Users\mbakh\OneDrive\Рабочий стол\0питон\ProcessFolder.py", line 4, in <module>
from PIL import Image
ModuleNotFoundError: No module named 'PIL'
```

Ошибка `ModuleNotFoundError: No module named 'PIL'` возникает потому, что в вашей среде Python не установлена библиотека Pillow, которая является современной заменой устаревшей PIL и предоставляет модуль PIL для работы с изображениями [\[9\]](#) [\[10\]](#).

Как исправить

1. Откройте терминал (командную строку) и выполните:

```
pip install Pillow
```

или, если у вас несколько версий Python и используется Python 3:

```
pip3 install Pillow
```

2. **Убедитесь, что используете тот же интерпретатор Python, который запускает ваш скрипт.** Иногда бывает несколько сред или версий Python, и Pillow может быть установлен не туда [\[11\]](#) [\[12\]](#) [\[13\]](#).

3. **Проверьте импорт в скрипте:**

```
from PIL import Image
```

Это корректный способ импортировать модуль после установки Pillow [\[9\]](#) [\[10\]](#).

После установки Pillow повторно запустите скрипт — ошибка должна исчезнуть.



Посмотри все нормально?

```
PS C:\Users\mbakh\OneDrive\Рабочий стол\Опитон> python ProcessFolder.py
```

```
Traceback (most recent call last):
```

```
File "C:\Users\mbakh\OneDrive\Рабочий стол\Опитон\ProcessFolder.py", line 4, in <module>
```

```
from PIL import Image
```

```
ModuleNotFoundError: No module named 'PIL'
```

```
PS C:\Users\mbakh\OneDrive\Рабочий стол\Опитон> pip install Pillow
```

```
Collecting Pillow
```

```
Downloading pillow-11.2.1-cp312-cp312-win_amd64.whl.metadata (9.1 kB)
```

```
Downloading pillow-11.2.1-cp312-cp312-win_amd64.whl (2.7 MB)
```

```
2.7/2.7 MB 9.7 MB/s eta 0:00:00
```

```
Installing collected packages: Pillow
```

```
Successfully installed Pillow-11.2.1
```

```
PS C:\Users\mbakh\OneDrive\Рабочий стол\Опитон> python ProcessFolder.py
```

```
PS C:\Users\mbakh\OneDrive\Рабочий стол\Опитон> pip3 install Pillow
```

```
Requirement already satisfied: Pillow in
```

```
c:\users\mbakh\appdata\local\programs\python\python312\lib\site-packages (11.2.1)
```

Судя по вашему выводу, Pillow установлен корректно, но скрипт не выдает никаких сообщений и просто завершает выполнение без ошибок. Это значит:

- Ошибка `ModuleNotFoundError: No module named 'PIL'` действительно исчезла после установки Pillow — это хорошо.
- Скрипт не выводит ничего в консоль, потому что в нем нет явных `print()`-команд, кроме заглушки в функции `upload_to_vector_db`, которая сработает только если в папке есть `.docx`-файлы и они обрабатываются.

Возможные причины "тишины":

- В папке, которую вы указали в `process_folder("путь_к_папке_с_docx")`, нет файлов с расширением `.docx`.
- Вы не раскомментировали строку запуска в конце скрипта:

```
# process_folder("путь_к_папке_с_docx")
```

Нужно убрать решетку и указать правильный путь, например:

```
process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\@питон\docs")
```

(где docs — папка с вашими Word-файлами).

Что делать:

1. Проверьте, что в папке есть .docx-файлы.
2. Убедитесь, что строка вызова функции process_folder раскомментирована и путь к папке указан верно.
3. Для контроля добавьте в начало функции process_folder строку:

```
print(f"Обрабатывается папка: {folder_path}")
```

и после каждого ключевого шага (text_chunks, table_chunks, image_chunks) добавьте вывод количества обработанных объектов.

Пример для отладки:

```
def process_folder(folder_path):
    print(f"Обрабатывается папка: {folder_path}")
    for file in os.listdir(folder_path):
        if file.endswith(".docx"):
            print(f"Обрабатывается файл: {file}")
            file_path = os.path.join(folder_path, file)
            text_blocks, tables, images = extract_docx_content(file_path)
            print(f"Текстовых блоков: {len(text_blocks)}, таблиц: {len(tables)}, изображений
            # Далее по коду...
```

Вывод:

Ошибки с Pillow больше нет, скрипт работает, но сейчас он ничего не выводит, если не находит файлов или не запущен с нужным путем. Добавьте отладочные сообщения и проверьте путь к папке — тогда вы увидите, что происходит на каждом этапе [\[14\]](#) [\[15\]](#) [\[16\]](#).

✱

Проверь скрипт и исправь ошибки

Ваш скрипт был проверен и исправлен. Вот основные улучшения и исправления, которые были внесены:

- **Добавлена обработка ошибок** при открытии Word-файлов и изображений, чтобы скрипт не прерывался при повреждённых или нестандартных файлах.
- **Добавлена проверка существования папки** и наличия в ней файлов .docx, чтобы избежать "тишины" при отсутствии файлов.
- **Добавлены информативные сообщения** на каждом этапе обработки для удобства отладки.

- Пример вызова функции теперь комментируется с использованием raw-строки (r"путь"), что важно для путей в Windows.

Исправленный фрагмент для запуска:

```
# Пример вызова функции
# process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\0питон\docs")
```

Уберите решётку (раскомментируйте строку) и укажите путь к папке с вашими .docx-файлами.

Итог

Скрипт теперь устойчив к ошибкам, информативен и готов к использованию на вашем ноутбуке. Просто раскомментируйте строку вызова и укажите путь к нужной папке, после чего скрипт выведет подробную информацию о ходе обработки файлов.

✱

import os

```
from docx import Document
from io import BytesIO
from PIL import Image
```

1. Извлечение содержимого из docx

```
def extract_docx_content(file_path):
    doc = Document(file_path)
    text_blocks = []
    tables = []
    images = []
```

```
    for para in doc.paragraphs:
        if para.text.strip():
            text_blocks.append(para.text.strip())

    for table in doc.tables:
        table_data = []
        for row in table.rows:
            row_data = [cell.text.strip() for cell in row.cells]
            table_data.append(row_data)
        tables.append(table_data)

    for rel in doc.part.rels.values():
        if "image" in rel.target_ref:
            image_bytes = rel.target_part.blob
            image = Image.open(BytesIO(image_bytes))
            images.append(image)
```

```
return text_blocks, tables, images
```

2. Чанкинг

```
def chunk_text(text_blocks, max_chars=1000):
    chunks = []
    current_chunk = ""
    for block in text_blocks:
        if len(current_chunk) + len(block) + 1 <= max_chars:
            current_chunk += (" " + block) if current_chunk else block
        else:
            chunks.append(current_chunk)
            current_chunk = block
    if current_chunk:
        chunks.append(current_chunk)
    return chunks

def chunk_tables(tables):
    def table_to_markdown(table):
        md = []
        for row in table:
            md.append("| " + " | ".join(row) + " |")
        return "\n".join(md)
    return [table_to_markdown(table) for table in tables]

def chunk_images(images):
    return images
```

3. Генерация описаний (заглушки)

```
def generate_table_description(table_markdown):
    return "Описание таблицы: краткое содержание данных."

def generate_image_description(image):
    return "Описание изображения: краткое описание содержимого."
```

4. Векторизация (заглушки)

```
def vectorize_text(text):
    return [0.1] * 768

def vectorize_table(table_desc):
    return [0.2] * 768

def vectorize_image(image_desc):
    return [0.3] * 768
```

5. Метаданные и загрузка

```
def add_metadata(chunks, chunk_type, file_name):
    metadata_list = []
    for i, chunk in enumerate(chunks):
        metadata = {
            "type": chunk_type,
            "file_name": file_name,
            "chunk_index": i
        }
        metadata_list.append(metadata)
    return metadata_list

def upload_to_vector_db(vectors, metadata):
    print(f"Uploading {len(vectors)} vectors with metadata to vector DB")
    return True
```

Основной цикл обработки файлов

```
def process_folder(folder_path):
    for file in os.listdir(folder_path):
        if file.endswith(".docx"):
            file_path = os.path.join(folder_path, file)
            text_blocks, tables, images = extract_docx_content(file_path)
            text_chunks = chunk_text(text_blocks)
            table_chunks = chunk_tables(tables)
            image_chunks = chunk_images(images)
```

```
        # Текст
        text_vecs = [vectorize_text(chunk) for chunk in text_chunks]
        text_meta = add_metadata(text_chunks, "text", file)
        upload_to_vector_db(text_vecs, text_meta)

        # Таблицы
        table_descs = [generate_table_description(tc) for tc in table_chunks]
        table_vecs = [vectorize_table(desc) for desc in table_descs]
        table_meta = add_metadata(table_chunks, "table", file)
        upload_to_vector_db(table_vecs, table_meta)

        # Изображения
        image_descs = [generate_image_description(img) for img in image_chunks]
        image_vecs = [vectorize_image(desc) for desc in image_descs]
        image_meta = add_metadata(image_chunks, "image", file)
        upload_to_vector_db(image_vecs, image_meta)
```

Пример запуска

```
process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\Опитон\docx")
```

Ваш скрипт в целом хорош, но есть несколько важных моментов, которые нужно исправить и улучшить:

Исправления и улучшения

1. Синтаксическая ошибка:

Перед вызовом `process_folder(...)` не должно быть пробела!

Исправьте:

```
process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\Опитон\docx")
```

на

```
process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\Опитон\docx")
```

2. Обработка ошибок:

Добавьте обработку ошибок при открытии файлов и изображений, чтобы скрипт не "падал" на повреждённых или необычных документах.

3. Проверка наличия папки и файлов:

Добавьте проверку, что папка существует и содержит `.docx`-файлы.

4. Более информативный вывод:

Добавьте `print`-сообщения для контроля процесса.

Исправленный и улучшенный скрипт

```
import os
from docx import Document
from io import BytesIO
from PIL import Image

def extract_docx_content(file_path):
    text_blocks = []
    tables = []
    images = []
    try:
        doc = Document(file_path)
    except Exception as e:
        print(f"Ошибка при открытии {file_path}: {e}")
        return text_blocks, tables, images

    for para in doc.paragraphs:
        if para.text.strip():
            text_blocks.append(para.text.strip())

    for table in doc.tables:
        table_data = []
```

```

    for row in table.rows:
        row_data = [cell.text.strip() for cell in row.cells]
        table_data.append(row_data)
    tables.append(table_data)

for rel in doc.part.rels.values():
    if "image" in rel.target_ref:
        try:
            image_bytes = rel.target_part.blob
            image = Image.open(BytesIO(image_bytes))
            images.append(image)
        except Exception as e:
            print(f"Ошибка при извлечении изображения из {file_path}: {e}")

return text_blocks, tables, images

def chunk_text(text_blocks, max_chars=1000):
    chunks = []
    current_chunk = ""
    for block in text_blocks:
        if len(current_chunk) + len(block) + 1 <= max_chars:
            current_chunk += (" " + block) if current_chunk else block
        else:
            chunks.append(current_chunk)
            current_chunk = block
    if current_chunk:
        chunks.append(current_chunk)
    return chunks

def chunk_tables(tables):
    def table_to_markdown(table):
        md = []
        for row in table:
            md.append("| " + " | ".join(row) + " |")
        return "\n".join(md)
    return [table_to_markdown(table) for table in tables]

def chunk_images(images):
    return images

def generate_table_description(table_markdown):
    return "Описание таблицы: краткое содержание данных."

def generate_image_description(image):
    return "Описание изображения: краткое описание содержимого."

def vectorize_text(text):
    return [0.1] * 768

def vectorize_table(table_desc):
    return [0.2] * 768

def vectorize_image(image_desc):
    return [0.3] * 768

def add_metadata(chunks, chunk_type, file_name):
    metadata_list = []
    for i, chunk in enumerate(chunks):

```

```

        metadata = {
            "type": chunk_type,
            "file_name": file_name,
            "chunk_index": i
        }
        metadata_list.append(metadata)
    return metadata_list

def upload_to_vector_db(vectors, metadata):
    print(f"Uploading {len(vectors)} vectors with metadata to vector DB")
    return True

def process_folder(folder_path):
    if not os.path.exists(folder_path):
        print(f"Папка {folder_path} не найдена!")
        return
    files = [f for f in os.listdir(folder_path) if f.endswith(".docx")]
    if not files:
        print(f"В папке {folder_path} нет файлов .docx")
        return
    for file in files:
        print(f"\nОбрабатывается файл: {file}")
        file_path = os.path.join(folder_path, file)
        text_blocks, tables, images = extract_docx_content(file_path)
        print(f"  Текстовых блоков: {len(text_blocks)}, таблиц: {len(tables)}, изображений:

        text_chunks = chunk_text(text_blocks)
        table_chunks = chunk_tables(tables)
        image_chunks = chunk_images(images)

        # Текст
        text_vecs = [vectorize_text(chunk) for chunk in text_chunks]
        text_meta = add_metadata(text_chunks, "text", file)
        upload_to_vector_db(text_vecs, text_meta)

        # Таблицы
        table_descs = [generate_table_description(tc) for tc in table_chunks]
        table_vecs = [vectorize_table(desc) for desc in table_descs]
        table_meta = add_metadata(table_chunks, "table", file)
        upload_to_vector_db(table_vecs, table_meta)

        # Изображения
        image_descs = [generate_image_description(img) for img in image_chunks]
        image_vecs = [vectorize_image(desc) for desc in image_descs]
        image_meta = add_metadata(image_chunks, "image", file)
        upload_to_vector_db(image_vecs, image_meta)

    # Пример запуска (убедитесь, что путь указан правильно и без лишних пробелов!)
    process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\опитон\docx")

```

Теперь скрипт устойчив, информативен и не содержит синтаксических ошибок.

Если возникнут вопросы — пишите!

Мне нужен результат в виде md файла

Чтобы получить результат обработки ваших Word-файлов в виде Markdown-файлов (.md), используйте предложенный скрипт, который автоматически сохраняет все извлечённые текстовые чанки, таблицы (в формате Markdown) и описания изображений в отдельный .md-файл для каждого документа.

Как это работает

- Для каждого файла .docx в указанной папке скрипт:
 - Извлекает текст, таблицы и изображения.
 - Делит текст на чанки, таблицы переводит в Markdown, для изображений генерирует текстовые описания.
 - Сохраняет результат в .md-файл с удобной структурой:
 - Чанки текста с номерами.
 - Все таблицы в формате Markdown.
 - Описания всех изображений.
- Каждый .md-файл сохраняется в отдельную папку, которую вы указываете для результатов.

Пример структуры Markdown-файла

```
# Результаты обработки документа

## Текстовые чанки
### Чанк 1
Текст чанка 1...

### Чанк 2
Текст чанка 2...

## Таблицы (в формате Markdown)
### Таблица 1
| Столбец1 | Столбец2 |
|-----|-----|
| ... | ... |

## Описания изображений
### Изображение 1
Описание изображения: краткое описание содержимого.
```

Как запустить

1. Убедитесь, что у вас установлены все необходимые библиотеки (python-docx, Pillow).
2. Создайте папку для результатов, например, results.
3. Раскомментируйте и отредактируйте строку вызова функции, указав путь к папке с .docx и папке для результатов:

```
process_folder(
    r"C:\Users\mbakh\OneDrive\Рабочий стол\0питон\docx",
    r"C:\Users\mbakh\OneDrive\Рабочий стол\0питон\results"
)
```

4. Запустите скрипт.

Особенности

- Скрипт не вставляет изображения в .md (только текстовые описания), так как Markdown не поддерживает встроенные бинарные данные.
- Все результаты сохраняются по одному .md-файлу на каждый исходный документ.

Вы получите удобные Markdown-файлы с полной структурой содержимого каждого Word-документа для дальнейшей работы или анализа.

Если нужно, чтобы изображения сохранялись как файлы и вставлялись в .md с помощью ссылок — дайте знать, добавлю соответствующий код.

✱

Да, сделай чтобы все файлы сохранялись в виде одного файла, а изображения встраивались в виде ссылок. Папка для результата md

Сделай один окончательный скрипт

Отлично! Вот финальный скрипт, который:

- Обрабатывает все .docx-файлы из указанной папки,
- Извлекает текст, таблицы и изображения,
- Делит текст на чанки, таблицы конвертирует в Markdown,
- Сохраняет все изображения как отдельные файлы (PNG) в папке md/images/,
- Формирует **один общий Markdown-файл** (например, md/result.md), где:
 - Текстовые чанки идут с подзаголовками,
 - Таблицы вставляются в формате Markdown,
 - Для каждого изображения вставляется ссылка на сохранённый файл и его описание.

Финальный скрипт

```
import os
from docx import Document
from io import BytesIO
from PIL import Image

def extract_docx_content(file_path):
```

```

text_blocks = []
tables = []
images = []
try:
    doc = Document(file_path)
except Exception as e:
    print(f"Ошибка при открытии {file_path}: {e}")
    return text_blocks, tables, images

for para in doc.paragraphs:
    if para.text.strip():
        text_blocks.append(para.text.strip())

for table in doc.tables:
    table_data = []
    for row in table.rows:
        row_data = [cell.text.strip() for cell in row.cells]
        table_data.append(row_data)
    tables.append(table_data)

for rel in doc.part.rels.values():
    if "image" in rel.target_ref:
        try:
            image_bytes = rel.target_part.blob
            image = Image.open(BytesIO(image_bytes))
            images.append(image)
        except Exception as e:
            print(f"Ошибка при извлечении изображения из {file_path}: {e}")

return text_blocks, tables, images

def chunk_text(text_blocks, max_chars=1000):
    chunks = []
    current_chunk = ""
    for block in text_blocks:
        if len(current_chunk) + len(block) + 1 <= max_chars:
            current_chunk += (" " + block) if current_chunk else block
        else:
            chunks.append(current_chunk)
            current_chunk = block
    if current_chunk:
        chunks.append(current_chunk)
    return chunks

def table_to_markdown(table):
    md = []
    for row in table:
        md.append("| " + " | ".join(row) + " |")
    return "\n".join(md)

def generate_table_description(table_markdown):
    return "Описание таблицы: краткое содержание данных."

def generate_image_description(image):
    return "Описание изображения: краткое описание содержимого."

def save_image(image, images_folder, doc_name, img_index):
    # Уникальное имя файла изображения

```

```

img_filename = f"{doc_name}_img_{img_index+1}.png"
img_path = os.path.join(images_folder, img_filename)
try:
    image.save(img_path)
    return img_filename
except Exception as e:
    print(f"Ошибка при сохранении изображения {img_filename}: {e}")
    return None

def process_folder(folder_path, result_folder):
    # Создаем папку для результатов и изображений
    images_folder = os.path.join(result_folder, "images")
    os.makedirs(images_folder, exist_ok=True)

    result_md_path = os.path.join(result_folder, "result.md")
    with open(result_md_path, "w", encoding="utf-8") as md_file:
        md_file.write(f"# Сводный отчет по всем docx-файлам\n\n")

    for file in sorted(os.listdir(folder_path)):
        if file.endswith(".docx"):
            doc_name = os.path.splitext(file)[0]
            file_path = os.path.join(folder_path, file)
            print(f"Обрабатывается файл: {file}")

            text_blocks, tables, images = extract_docx_content(file_path)
            text_chunks = chunk_text(text_blocks)
            table_markdowns = [table_to_markdown(table) for table in tables]

            # --- Запись в md-файл ---
            md_file.write(f"\n\n---\n\n")
            md_file.write(f"### Документ: {file}\n\n")

            # Текстовые чанки
            if text_chunks:
                md_file.write(f"### Текстовые чанки\n")
                for i, chunk in enumerate(text_chunks, 1):
                    md_file.write(f"#### Чанк {i}\n")
                    md_file.write(chunk + "\n\n")

            # Таблицы
            if table_markdowns:
                md_file.write(f"### Таблицы\n")
                for i, table_md in enumerate(table_markdowns, 1):
                    md_file.write(f"#### Таблица {i}\n")
                    md_file.write(table_md + "\n\n")
                    md_file.write(f"*{generate_table_description(table_md)}*\n\n")

            # Изображения
            if images:
                md_file.write(f"### Изображения\n")
                for i, image in enumerate(images):
                    img_filename = save_image(image, images_folder, doc_name, i)
                    if img_filename:
                        img_rel_path = f"images/{img_filename}"
                        md_file.write(f"#### Изображение {i+1}\n")
                        md_file.write(f"![{img_filename}]({img_rel_path})\n\n")
                        md_file.write(f"*{generate_image_description(image)}*\n\n")
                    else:

```

```

md_file.write(f"#### Изображение {i+1}\n")
md_file.write(f"Ошибка при сохранении изображения\n\n")

print(f"\nГотово! Результат в файле: {result_md_path}")
print(f"Изображения сохранены в папке: {images_folder}")

# Пример запуска:
# process_folder(r"C:\Users\mbakh\OneDrive\Рабочий стол\0питон\docx", r"C:\Users\mbakh\OneDr

```

Как использовать

1. **Создайте папку для результата**, например, md в нужном месте.
2. **Запустите скрипт** с нужными путями, раскомментировав и изменив строку запуска:

```

process_folder(
    r"C:\Users\mbakh\OneDrive\Рабочий стол\0питон\docx",
    r"C:\Users\mbakh\OneDrive\Рабочий стол\0питон\md"
)

```

3. В результате:

- Все содержимое будет собрано в один файл md/result.md.
- Все изображения будут лежать в папке md/images/ и вставлены в Markdown как ссылки.

Если потребуется доработать генерацию описаний или добавить дополнительные метаданные — дайте знать!

1. <https://www.yotec.net/extracting-text-and-images-from-word-documents-using-python/>
2. <https://docs.groupdocs.com/parser/net/extract-tables-from-microsoft-office-word-documents/>
3. <https://stackoverflow.com/questions/58566764/how-to-extract-image-from-table-in-ms-word-document-with-docx-library>
4. https://www.reddit.com/r/LangChain/comments/16pedi1/which_approach_would_be_better_for_embedding_my/
5. https://www.reddit.com/r/LangChain/comments/15jo7t2/best_chunking_practice_for_mixed_documents_with/
6. <https://blog.gopenai.com/rag-loading-multimodal-documents-into-vector-databases-with-microsoft-phi-3-phi-3-vision-and-30b142e0d26e>
7. <https://www.linkedin.com/pulse/how-index-data-vector-db-from-highly-unstructured-pdfs-gaurav-chopra-apblc>
8. <https://learn.microsoft.com/en-us/answers/questions/1658828/issues-with-handling-multi-page-tables-in-document>
9. <https://sky.pro/wiki/python/oshibka-import-error-no-module-named-pil-v-python-reshenie/>
10. <https://www.geeksforgeeks.org/how-to-fix-modulenotfounderror-no-module-named-pil-in-python/>
11. https://www.reddit.com/r/learnpython/comments/tzi219/no_module_named_pil_need_help_vscode/
12. https://www.reddit.com/r/learnpython/comments/112og9p/pillow_already_installed_but_modulenotfounderro/

13. https://www.reddit.com/r/learnpython/comments/f2b3bd/pil_is_not_working/
14. <https://www.geeksforgeeks.org/how-to-fix-modulenotfounderror-no-module-named-pil-in-python/>
15. https://www.reddit.com/r/learnpython/comments/gve6v4/unable_to_import_pil_when_i_have_pillow_installed/
16. <https://stackoverflow.com/questions/54055909/pillow-is-installed-but-still-getting-an-error>