# Proposed Strucutre

To create a high-quality markdown documentation file (embed.md) for the 2\_chunk.py module, modeled after the well-structured scrape.md, here's a detailed plan for structuring the documentation. This structure ensures clarity, logical flow, and a comprehensive linkage to all relevant helper functions and modules.

**📘 Proposed Structure for embed.md**

**📚 Table of Contents**

1. [Overview](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#overview)
2. [Architecture](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#architecture)
3. [Core Functions](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#core-functions)
   * run\_script()
   * chunk\_file\_many()
   * chunk\_file\_one()
   * get\_file\_paths()
4. [Text Extraction Pipeline](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#text-extraction-pipeline)
5. [Chunking Process](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#chunking-process)
6. [Cleaning Strategy](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#cleaning-strategy)
7. [Database Operations](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#database-operations)
8. [Error Handling](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#error-handling)
9. [Usage](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#usage)
10. [Performance Considerations](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#performance-considerations)
11. [Contributing](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#contributing)
12. [Appendix](https://chatgpt.com/g/g-p-67fe726484c8819191dd1403485129af-ds205-group-project/c/68384556-2ad0-8002-b7fb-acc12c0110e3#appendix)

**🧾 Overview**

* Purpose of 2\_chunk.py within the pipeline (PDF processing, extraction, chunking).
* Connection to project goals (automated NDC document analysis).
* Why chunking is necessary for embeddings and downstream RAG tasks.

**🏗️ Architecture**

* High-level diagram or narrative:
  + PDF → extractor.py → raw elements
  + Raw elements → chunker.py → structured chunks
  + Chunks → cleaner.py → clean, semantically-valid chunks
  + Final → Database (models.py, operations.py)
* Mention concurrency (FILE\_PROCESSING\_CONCURRENCY) and progress bar via tqdm.

**⚙️ Core Functions**

**run\_script(force\_reprocess: bool = False)**

* Orchestrates the full file processing workflow.
* Annotated usage with CLI flag (--force).

**chunk\_file\_many(file\_path)**

* Adds concurrency control via async semaphores.
* Used internally by run\_script.

**chunk\_file\_one(file\_path: str, force\_reprocess: bool = False)**

* Core logic function.
* Explain every step: metadata creation → text extraction → chunking → cleaning → ORM → upload.

**get\_file\_paths()**

* Fetches PDF paths from data directory.

**🔍 Text Extraction Pipeline**

Describe how extract\_text\_from\_pdf() from extractor.py is:

* Strategy-based: 'fast', 'auto', 'ocr\_only'.
* Fault-tolerant: falls back on OCR if necessary.
* Handles multiple output formats and metadata normalization.

**✂️ Chunking Process**

From chunker.py:

* Explain sentence-based strategy: chunk\_document\_by\_sentences.
* Smart handling of headings, paragraphs, overlapping.
* Metadata propagation (page number, country, etc.).
* Each chunk maintains semantic coherence.

**🧹 Cleaning Strategy**

From cleaner.py:

* Sequence: merge\_short\_chunks → split\_long\_chunks → remove\_gibberish.
* Justify parameters (e.g., min/max length thresholds).
* Explain why text needs cleaning before embeddings.

**💾 Database Operations**

From operations.py and models.py:

* Creation of NDCDocumentORM if document is new.
* Chunk creation as DocChunkORM objects.
* upload() to insert into doc\_chunks table.
* update\_processed() to mark document as processed.

From auth.py:

* Use of PostgresConnection().get\_session() for session management.

**❗ Error Handling**

* Try-except logic wrapped around all critical operations.
* Rollback logic for safe database writes.
* traceback used to preserve full error logs.
* Fallback for failed extraction (dummy/fallback chunk inserted).

**🚀 Usage**

**CLI Example**

python 2\_chunk.py --force

**Programmatic Example**

from group4py.src.chunk import run\_script

await run\_script(force\_reprocess=True)

**🧠 Performance Considerations**

* Async chunking with semaphore control.
* tqdm for user visibility on progress.
* Efficient session use (get\_session as context manager).

**🤝 Contributing**

* Follow existing module separation.
* Maintain exception chaining and type hinting.
* Update helper scripts if the chunk format or metadata schema evolves.

**📎 Appendix**

**Linked Helper Functions**

* extract\_text\_from\_pdf → extractor.py
* chunk\_document\_by\_sentences / cleaning\_function → chunker.py
* merge\_short\_chunks, split\_long\_chunks, remove\_gibberish → cleaner.py
* upload, check\_document\_processed, update\_processed → operations.py
* NDCDocumentORM, DocChunkORM → models.py

Let me know if you'd like to proceed with drafting the actual embed.md file based on this structure.

# Proposed Code Snippets and Graphical Depictions

Here’s an expanded, detailed implementation plan for your embed.md documentation — now including:

* Specific **code snippets** to include in each section (concise and purposeful).
* Where to insert **flowcharts or diagrams** to enhance clarity.
* Careful curation of helper function references to maintain cohesion.

**✅ Full Plan for embed.md Documentation**

**📚 Table of Contents**

No code or diagram needed here — just structured links to the sections below.

**📘 Overview**

**Purpose:** Explain what this script does (chunking PDFs into clean, metadata-rich chunks for vector embedding and querying).

✅ **Code Snippet:**

asyncio.run(run\_script(force\_reprocess=args.force))

📊 **Graphical Element:**  
**System Context Diagram** (flowchart)

PDF File

↓

extract\_text\_from\_pdf()

↓

chunk\_document\_by\_sentences()

↓

cleaning\_function()

↓

Database: doc\_chunks table

**🏗️ Architecture**

**Purpose:** Describe how the components fit together.

✅ **Code Snippet:**  
Show imports from helper files:

from group4py.src.chunk.extractor import extract\_text\_from\_pdf

from group4py.src.chunk.chunker import DocChunker

from databases.operations import check\_document\_processed, upload, update\_processed

📊 **Graphical Element:**  
**Modular Dependency Map** showing function calls across extractor.py, chunker.py, cleaner.py, operations.py.

**⚙️ Core Functions**

**run\_script()**

✅ **Code Snippet:**

@Logger.log(log\_file = project\_root / "logs/chunk.log", log\_level="INFO")

async def run\_script(force\_reprocess: bool = False):

🗒️ **Purpose:** Entry-point with tqdm tracking and asyncio.

**chunk\_file\_many()**

✅ **Code Snippet:**

semaphore = asyncio.Semaphore(FILE\_PROCESSING\_CONCURRENCY)

async with semaphore:

return await chunk\_file\_one(file\_path)

🗒️ **Purpose:** Adds concurrency via semaphores.

**chunk\_file\_one()**

This section will show the **key stages** of the file processing lifecycle.

✅ **Code Snippets (each as inline subsections)**:

* **Check if already processed**:

is\_processed, document = check\_document\_processed(session, doc\_id)

* **Text extraction loop**:

for strategy in ['fast', 'auto', 'ocr\_only']:

raw\_extracted\_elements = extract\_text\_from\_pdf(file\_path, strategy=strategy)

* **Chunking logic**:

chunks = DocChunker.chunk\_document\_by\_sentences(extracted\_elements)

* **Cleaning logic**:

cleaned\_chunks = DocChunker.cleaning\_function(chunks)

* **Upload logic**:

upload\_success = upload(session, db\_chunks, table='doc\_chunks')

* **Mark as processed**:

update\_processed(session, NDCDocumentORM, doc\_id, chunks=cleaned\_chunks)

📊 **Graphical Element:**  
**Detailed Data Flow Diagram** showing flow within chunk\_file\_one:

[PDF File]

↓

[extract\_text\_from\_pdf()]

↓

[chunk\_document\_by\_sentences()]

↓

[cleaning\_function()]

↓

[upload() to database]

↓

[update\_processed()]

**🔍 Text Extraction Pipeline**

✅ **Code Snippet:**

extracted\_elements = extract\_text\_from\_pdf(file\_path, strategy=strategy)

🗒️ Link to fallback strategy:

if has\_corruption and len(processed\_elements) < 5:

return retry\_with\_ocr(pdf\_path, languages\_list)

📊 **Graphical Element:**  
**Strategy Decision Tree**:

* Strategy = fast → success?
* No → try auto
* Still fails → OCR fallback
* Still fails → fallback chunk

**✂️ Chunking Process**

✅ **Code Snippet:**

chunks = DocChunker.chunk\_document\_by\_sentences(extracted\_elements)

🗒️ Highlight sentence overlap logic and metadata aggregation.

**🧹 Cleaning Strategy**

✅ **Code Snippet:**

cleaned\_chunks = DocChunker.cleaning\_function(chunks)

🗒️ Then reference helper stages:

merge\_short\_chunks → split\_long\_chunks → remove\_gibberish

📊 **Graphical Element:**  
Mini-pipeline:

Raw Chunks

↓

Merge Short

↓

Split Long

↓

Remove Gibberish

↓

Cleaned Chunks

**💾 Database Operations**

✅ **Code Snippet:**

new\_document = NDCDocumentORM(...)

upload\_success = upload(session, db\_chunks, table='doc\_chunks')

update\_processed(session, NDCDocumentORM, doc\_id, chunks=cleaned\_chunks)

🗒️ Reference:

* check\_document\_processed() in operations.py
* upload() and UUID validation logic
* ORM models from models.py (NDCDocumentORM, DocChunkORM)

**❗ Error Handling**

✅ **Code Snippet:**

try:

...

except Exception as e:

logger.error(...)

session.rollback()

🗒️ Discuss use of:

* Try/catch in every stage
* Fallback chunk creation:

extracted\_elements = [{'text': fallback\_text, 'metadata': {...}}]

**🚀 Usage**

✅ **Code Snippet:**

python 2\_chunk.py --force

await run\_script(force\_reprocess=True)

**🧠 Performance Considerations**

✅ **Code Snippet:**

with tqdm(total=len(file\_paths), desc="Chunking PDF files", unit="file") as pbar:

🗒️ Discuss:

* Async control
* Batched memory usage
* No unnecessary data retention

**🤝 Contributing**

🗒️ Reference:

* Respect modularity (add new chunking methods in DocChunker)
* Ensure UUID consistency when adding database operations

**📎 Appendix: Linked Helper Functions**

✅ Include table mapping function → location.

| **Function** | **File** | **Purpose** |
| --- | --- | --- |
| extract\_text\_from\_pdf() | extractor.py | Extracts structured text |
| chunk\_document\_by\_sentences() | chunker.py | Converts elements to chunks |
| cleaning\_function() | chunker.py | Applies merge/split/clean steps |
| merge\_short\_chunks() | cleaner.py | Merges small chunks |
| upload() | operations.py | Uploads chunk ORM objects |
| update\_processed() | operations.py | Flags document as processed |

This plan ensures:

* **Technical accuracy**
* **Modular traceability**
* **Clear visual storytelling**
* **User guidance**

Let me know when you're ready for me to begin generating the actual markdown file based on this structure.