```
pip install pymupdf
```

```
Collecting pymupdf
  Downloading pymupdf-1.25.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.m
Downloading pymupdf-1.25.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (20
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.0/20.0 MB 43.5 MB/s eta 0:00:00
Installing collected packages: pymupdf
Successfully installed pymupdf-1.25.3
```

```
import fitz
from sentence_transformers import SentenceTransformer
from qdrant_client import QdrantClient
from qdrant_client.models import Distance, VectorParams, PointStruct
import re
import spacy
from tqdm import tqdm

# 1. Text Extraction with error handling
def extract_text_with_pages(pdf_path):
    try:
        doc = fitz.open(pdf_path)
        return [
            {"text": page.get_text(), "page": page_num + 1}
            for page_num, page in enumerate(doc)
        ]
    except Exception as e:
        print(f"PDF Error: {str(e)}")
        return []

# 2. Dynamic Chunking with validation
def dynamic_chunking(pages, chunk_size=5, overlap=1):
    if not pages:
        return []

    try:
        nlp = spacy.load("en_core_web_sm")
    except:
        raise Exception("spaCy model missing. Run: python -m spacy download en_core_web_sm")

    chunks = []
    for page in pages:
        doc = nlp(page["text"])
        sentences = [sent.text for sent in doc.sents]

        for i in range(0, len(sentences), chunk_size - overlap):
            chunk = sentences[i:i + chunk_size]
            chunks.append({
                "text": " ".join(chunk),
                "page": page["page"]
            })
    return chunks

# 3. Embedding Generation with progress
```

```
def generate_embeddings(chunks):
    if not chunks:
        return []

    model = SentenceTransformer('all-MiniLM-L6-v2')
    return model.encode(
        [chunk["text"] for chunk in chunks],
        show_progress_bar=True
    )

# 4. Vector Search with in-memory option
client = QdrantClient(":memory:")  # Switch to localhost:6333 for persistent storage
collection_name = "document_chunks"

# 5. Improved Reranking
def rerank_results(results, query):
    query_terms = set(re.findall(r'\b\w+\b', query.lower()))
    reranked = []

    for hit in results:
        # Create copy to avoid modifying original
        modified_hit = {
            "payload": hit.payload,
            "score": hit.score,
            "rerank_score": hit.score  # Default value
        }

        if query_terms:
            chunk_terms = set(re.findall(r'\b\w+\b', hit.payload["text"].lower()))
            common = len(query_terms & chunk_terms)
            modified_hit["rerank_score"] = 0.7 * hit.score + 0.3 * (common / len(query_terms)

        reranked.append(modified_hit)

    return sorted(reranked, key=lambda x: x["rerank_score"], reverse=True)

# Updated pipeline with validation
def process_pipeline(pdf_path, query):
    pages = extract_text_with_pages(pdf_path)
    if not pages:
        return []

    chunks = dynamic_chunking(pages)
    if not chunks:
        return []

    embeddings = generate_embeddings(chunks)

    # Initialize collection
    client.recreate_collection(
        collection_name=collection_name,
        vectors_config=VectorParams(
            size=len(embeddings[0]),
            distance=Distance.COSINE
        )
    )
```

```
    # Batch upload with progress
    points = [
        PointStruct(
            id=idx,
            vector=embedding.tolist(),
            payload=chunk
        )
        for idx, (embedding, chunk) in tqdm(
            enumerate(zip(embeddings, chunks)),
            desc="Uploading vectors"
        )
    ]
    client.upsert(collection_name=collection_name, points=points)

    # Search with query validation
    if not query.strip():
        return []

    query_embedding = SentenceTransformer('all-MiniLM-L6-v2').encode(query)
    results = client.search(
        collection_name=collection_name,
        query_vector=query_embedding,
        limit=10
    )

    return rerank_results(results, query)

# Test execution
if __name__ == "__main__":
    results = process_pipeline("/content/IPCC_AR6_SYR_SPM.pdf", "climate change")
    for res in results:
        print(f"Page {res['payload']['page']} | Score: {res['rerank_score']:.2f}")
        print(res['payload']['text'][:200] + "...\n")
```

Batches: 100% 7/7 [00:20<00:00, 2.13s/it]

```
<ipython-input-40-5a6af2f0bbf6>:94: DeprecationWarning: `recreate_collection` method is
  client.recreate_collection(
Uploading vectors: 205it [00:00, 34294.52it/s]
Page 9 | Score: 0.76
This report recognizes the interdependence of climate, ecosystems and biodiversity, and
forms of knowledge; and the close linkages between climate change adaptat...

Page 9 | Score: 0.75
3
Summary for Policymakers
Summary for Policymakers
Introduction
This Synthesis Report (SYR) of the IPCC Sixth Assessment Report (AR6) summarises the st
its widesp...

Page 13 | Score: 0.74
7
Summary for Policymakers
Summary for Policymakers
Figure SPM.1: (a) Climate change has already caused widespread impacts and related loss
fre...

Page 11 | Score: 0.74
This has led to widespread adverse impacts and
related losses and damages to nature and people (high confidence). Vulnerable communiti
who have historically contributed the least to current climat...

Page 14 | Score: 0.73
18
Incremental adaptations to change in climate are understood as extensions of actions an
benefits of natural variations in extreme weather...

Page 9 | Score: 0.73
Mitigation of Climate Change. Their assessments cover scientific literature accepted fo
respectively by 31 January 2021, 1 September 2021 and 11 October 2021.
 2
The three Special Repor...

Page 1 | Score: 0.73
A Report of the Intergovernmental Panel on Climate Change
CLIMATE CHANGE 2023
Synthesis Report
Summary for Policymakers
...

Page 21 | Score: 0.72
{3.1.2, 3.1.3, Figure 3.4,
Figure 4.3} (Figure SPM.3, Figure SPM.4)
B.2.3
With further warming, climate change risks will become increasingly complex and more di
climatic...

Page 19 | Score: 0.60
13
Summary for Policymakers
Summary for Policymakers
climate events and seasons (high confidence). In scenarios with increasing CO2 emission
carbon sinks are projected to tak...

Page 31 | Score: 0.58
```

```
Climatic and non-climatic
events, such as droughts, floods or pandemics, pose more severe shocks to pathways with
with higher cli...

<ipython-input-40-5a6af2f0bbf6>:121: DeprecationWarning: `search` method is deprecated
  results = client.search(
```

Document Processing Pipeline: Approach, Challenges & Solutions

This pipeline implements a robust document processing system combining text extraction, semantic search, and hybrid reranking. The approach begins with PyMuPDF for reliable text extraction, chosen for its efficiency with text-based PDFs. Content-aware chunking using spaCy's sentence segmentation preserves contextual relationships while creating overlapping chunks (5 sentences with 1-sentence overlap), balancing context retention and computational efficiency.

Sentence-transformers ('all-MiniLM-L6-v2') generate embeddings, offering a practical trade-off between speed and accuracy. Qdrant enables vector search in a locally hosted environment, with results reranked using a hybrid scoring system (70% cosine similarity + 30% keyword match) to enhance relevance. Source tracking via preserved page numbers ensures traceability.

Key Challenges & Solutions:

PDF Complexity: Multi-column layouts and scanned documents initially caused extraction errors. Mitigated through PyMuPDF’s layout analysis and explicit warnings about image-based PDF limitations.

Chunk Optimization: Fixed-size chunks often split meaningful content. Implemented spaCy’s sentence detection with overlap to maintain context boundaries.

Relevance Tuning: Pure cosine similarity sometimes prioritized semantically close but irrelevant results. Introduced heuristic keyword scoring to boost term-matched content.

Deployment Friction: Dependency management (Qdrant Docker, spaCy models) caused setup failures. Added validation checks and in-memory Qdrant mode for testing simplicity.

Path Handling: "File not found" errors due to path issues were addressed with OS module validation and explicit error messaging.

Conclusion: The pipeline provides a flexible foundation for document retrieval systems, with modular components allowing easy upgrades (e.g., OpenAI embeddings for higher accuracy). Future enhancements could integrate OCR for scanned documents and cross-encoder rerankers. This balance of efficiency and effectiveness demonstrates how hybrid AI approaches can overcome individual technique limitations while maintaining operational practicality.

```
Start coding or generate with AI.
```