perplexity

Code Explanation: Document Chunker for RAG Pipeline Optimizer

This code is a **core component** of your RAG Pipeline Optimizer project that handles the crucial chunking phase. It tests multiple chunking strategies (256, 512, 768, 1024 tokens) to help determine which works best for your Canadian wilderness dataset.[1] [2]

Class Structure

`ChunkingStrategy` **Dataclass**

```
@dataclass
class ChunkingStrategy:
    name: str          # e.g., "small", "medium"
    chunk_size: int    # Maximum tokens per chunk
    chunk_overlap: int # Overlapping tokens between consecutive chunks
```

This defines configuration for each chunking approach you'll test in parallel.[2] [3]

`DocumentChunker` **Class**
The main processor that splits your Parks Canada trails, safety guides, and AllTrails data into retrievable chunks.

Key Methods Breakdown

1. `__init__(encoding_name="cl100k_base")`

Initializes with GPT-4's tokenizer. Sets up 4 strategies:[4]

- **Small**: 256 tokens, 50 overlap (fine-grained retrieval)
- **Medium**: 512 tokens, 100 overlap (balanced)
- **Large**: 768 tokens, 150 overlap (more context)
- **Extra Large**: 1024 tokens, 200 overlap (maximum context)[3]

The overlap prevents splitting related information and improves retrieval quality.[5]

2. `split_by_tokens(text, chunk_size, overlap)`

Simple token-based splitting:

- Encodes text to tokens using tiktoken
- Creates fixed-size chunks

- Overlaps chunks by specified amount
- May split mid-sentence (less semantic coherence)

3. `split_by_sentences(text, chunk_size, overlap)` [RECOMMENDED]

Semantic-aware splitting:

- Splits on sentence boundaries (`.!?`)
- Respects token limits while keeping sentences intact
- Better for RAG because retrieved chunks are more coherent[6] [3]
- Handles edge cases (sentences exceeding chunk_size)

Why this matters: When your RAG system retrieves "Bear safety tips," you want complete sentences, not "...spray within 7.5 meters. Never run f..." cut mid-thought.

4. `chunk_document(document, strategy, method)`

Processes a single document:

- Applies chosen strategy and method
- Creates chunk objects with rich metadata:
  - `chunk_id`: Unique identifier
  - `text`: The actual chunk content
  - `token_count`: For cost tracking in your evaluator[4]
  - `document_title`, `region`, `province`: Preserved from source

This metadata helps your LLM judge evaluate context relevance.[4]

5. `process_all_documents(documents, output_dir, method)`

The main orchestrator:

- Loops through all 4 strategies
- Chunks all documents for each strategy
- Saves separate JSON files: `chunks_small.json`, `chunks_medium.json`, etc.
- Tracks statistics (total chunks, avg tokens per chunk)
- Generates comparison report

For your 277k documents, this creates 4 complete index variations to test which performs best.[2]

6. `generate_comparison_report(output_dir)`

Creates `chunking_report.json` with statistics for each strategy:

```
{
  "strategies": [
    {
      "name": "medium",
      "chunk_size": 512,
      "total_chunks": 15430,
      "avg_tokens_per_chunk": 487.2
    }
  ]
}
```

How It Fits Your RAG Optimizer

In your 4-pipeline comparison:[1]

1. **Pipeline A**: Uses `chunks_small.json` with BERT embeddings
2. **Pipeline B**: Uses `chunks_medium.json` with Cohere v3
3. **Pipeline C**: Uses `chunks_large.json` with OpenAI embeddings + reranker
4. **Pipeline D**: Uses `chunks_extra_large.json` with custom config

Your GPT-4o judge then evaluates which chunking strategy produces the most accurate, relevant answers.[4]

Usage Flow

```
# 1. Load your collected data
documents = load_canada_wilderness_data()  # From your scrapers

# 2. Initialize chunker
chunker = DocumentChunker()

# 3. Process all documents with all strategies
chunker.process_all_documents(
    documents=documents,
    output_dir="data/chunked",
    method="sentences"  # Use semantic splitting
)

# 4. Outputs created:
# data/chunked/chunks_small.json     (fine-grained)
# data/chunked/chunks_medium.json    (balanced)
# data/chunked/chunks_large.json     (contextual)
# data/chunked/chunks_extra_large.json (max context)
# data/chunked/chunking_report.json  (comparison stats)
```

Performance Considerations

Test mode is crucial: With 277k documents from Parks Canada and AllTrails, full processing could take **1-2 hours**. The code includes:

```
if test_mode == 'y':
    documents = documents[:1000]  # Test with subset first
```

Run test mode first to verify your pipeline works, then do full processing overnight.[1]

Token Counting Importance

Using `tiktoken.get_encoding("cl100k_base")` ensures token counts match what OpenAI APIs charge you. Your cost tracking in the evaluator agent will be accurate when comparing "Pipeline B costs $0.12 per query vs Pipeline D costs $0.08".[2] [4]

Output Format Example

Each chunk JSON contains:

```
{
  "chunk_id": "banff_johnston_canyon_medium_chunk_5",
  "text": "Johnston Canyon Trail is one of Banff's most popular hikes...",
  "token_count": 487,
  "strategy": "medium",
  "document_title": "Johnston Canyon Trail - Banff",
  "region": "Canadian Rockies",
  "province": "Alberta"
}
```

This rich metadata helps your RAG system provide contextualized answers like "Based on Johnston Canyon Trail in Banff...".[1] [2]

Next Steps in Your Project

After running this chunker:

1. **Embed each chunk set** with different models (BERT, Cohere v3, OpenAI)[1]
2. **Store in vector databases** (4 separate indices)
3. **Query all 4 pipelines** in parallel with test questions[2]
4. **Let GPT-4o judge** which chunking strategy retrieves better context[4]
5. **Visualize in dashboard**: "For your data, medium chunks (512 tokens) are 20% more accurate than large chunks"[1]

This systematic approach answers the question you posed at the start: "Is chunk_size=512 better than 1024?"[3] [2] [1]

⁂

1. https://dev.to/gautamvhavle/building-production-rag-systems-from-zero-to-hero-2f1i
2. https://www.deepchecks.com/build-high-performance-rag-pipelines-scale/
3. https://developer.nvidia.com/blog/finding-the-best-chunking-strategy-for-accurate-ai-responses/
4. https://wandb.ai/ai-team-articles/evals/reports/Evaluate-your-RAG-pipeline-using-LLM-as-a-Judge-with-custom-dataset-creation-Part-2---VmlldzoxNTIwNjI2MQ
5. https://docs.cohere.com/page/chunking-strategies
6. https://www.cohorte.co/blog/multimodal-rag-for-comprehensive-pdf-document-processing