book_to_slide_BY_sections

June 24, 2025

1 Set up Paths

[1]:
```
# Cell 1: Setup and Configuration
import os
import re
import logging
import warnings
from docx import Document
import pdfplumber
import ollama
from tenacity import retry, stop_after_attempt, wait_exponential, RetryError
import json

# Setup Logger for this cell
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -␣
 ↪%(message)s')
logger = logging.getLogger(__name__)

# --- 1. CORE SETTINGS ---
# Set this to True for EPUB, False for PDF. This controls the entire notebook's␣
 ↪flow.
# PROCESS_EPUB = True # for EPUB
PROCESS_EPUB = False # for PDF

# --- 2. INPUT FILE NAMES ---
# The name of the Unit Outline file (e.g., DOCX, PDF)
UNIT_OUTLINE_FILENAME = "ICT311 Applied Cryptography.docx" # pdf
# UNIT_OUTLINE_FILENAME = "ICT312 Digital Forensic_Final.docx" # epub

# The names of the book files
EPUB_BOOK_FILENAME = "Bill Nelson, Amelia Phillips, Christopher Steuart - Guide␣
 ↪to Computer Forensics and Investigations_ Processing Digital␣
 ↪Evidence-Cengage Learning (2018).epub"
PDF_BOOK_FILENAME = "(Chapman & Hall_CRC Cryptography and Network Security␣
 ↪Series) Jonathan Katz, Yehuda Lindell - Introduction to Modern␣
 ↪Cryptography-CRC Press (2020).pdf"
```

```python
# --- 3. DIRECTORY STRUCTURE ---
# Define the base path to your project to avoid hardcoding long paths everywhere
PROJECT_BASE_DIR = "/home/sebas_dev_linux/projects/course_generator"

# Define subdirectories relative to the base path
DATA_DIR = os.path.join(PROJECT_BASE_DIR, "data")
PARSE_DATA_DIR = os.path.join(PROJECT_BASE_DIR, "Parse_data")

# Construct full paths for clarity
INPUT_UO_DIR = os.path.join(DATA_DIR, "UO")
INPUT_BOOKS_DIR = os.path.join(DATA_DIR, "books")
OUTPUT_PARSED_UO_DIR = os.path.join(PARSE_DATA_DIR, "Parse_UO")
OUTPUT_PARSED_TOC_DIR = os.path.join(PARSE_DATA_DIR, "Parse_TOC_books")
OUTPUT_DB_DIR = os.path.join(DATA_DIR, "DataBase_Chroma")

# --- 4. LLM & EMBEDDING CONFIGURATION ---
LLM_PROVIDER = "ollama"  # Can be "ollama", "openai", "gemini"
OLLAMA_HOST = "http://localhost:11434"
OLLAMA_MODEL = "mistral:latest"
EMBEDDING_MODEL_OLLAMA = "nomic-embed-text"
CHUNK_SIZE = 800
CHUNK_OVERLAP = 100

# --- 5. DYNAMICALLY GENERATED PATHS & IDs (DO NOT EDIT THIS SECTION) ---
# This section uses the settings above to create all the necessary variables␣
 ↪for later cells.

# Extract Unit ID from the filename
def extract_uo_id_from_filename(filename: str) -> str:
    match = re.match(r'^[A-Z]+\d+', os.path.basename(filename))
    if match:
        return match.group(0)
    raise ValueError(f"Could not extract a valid Unit ID from filename:␣
 ↪'{filename}'")

try:
    UNIT_ID = extract_uo_id_from_filename(UNIT_OUTLINE_FILENAME)
except ValueError as e:
    print(f"Error: {e}")
    UNIT_ID = "UNKNOWN_ID"

# Full path to the unit outline file
FULL_PATH_UNIT_OUTLINE = os.path.join(INPUT_UO_DIR, UNIT_OUTLINE_FILENAME)

# Determine which book and output paths to use based on the PROCESS_EPUB flag
if PROCESS_EPUB:
    BOOK_PATH = os.path.join(INPUT_BOOKS_DIR, EPUB_BOOK_FILENAME)
```

```
    PRE_EXTRACTED_TOC_JSON_PATH = os.path.join(OUTPUT_PARSED_TOC_DIR,␣
 ↪f"{UNIT_ID}_epub_table_of_contents.json")
else:
    BOOK_PATH = os.path.join(INPUT_BOOKS_DIR, PDF_BOOK_FILENAME)
    PRE_EXTRACTED_TOC_JSON_PATH = os.path.join(OUTPUT_PARSED_TOC_DIR,␣
 ↪f"{UNIT_ID}_pdf_table_of_contents.json")

# Define paths for the vector database
file_type_suffix = 'epub' if PROCESS_EPUB else 'pdf'
CHROMA_PERSIST_DIR = os.path.join(OUTPUT_DB_DIR,␣
 ↪f"chroma_db_toc_guided_chunks_{file_type_suffix}_v2")
CHROMA_COLLECTION_NAME = f"book_toc_guided_chunks_{file_type_suffix}_v2"

# Define path for the parsed unit outline
PARSED_UO_JSON_PATH = os.path.join(OUTPUT_PARSED_UO_DIR, f"{os.path.
 ↪splitext(UNIT_OUTLINE_FILENAME)[0]}_parsed.json")

# --- Sanity Check Printout ---
print("--- CONFIGURATION SUMMARY ---")
print(f"Processing Mode: {'EPUB' if PROCESS_EPUB else 'PDF'}")
print(f"Unit ID: {UNIT_ID}")
print(f"Unit Outline Path: {FULL_PATH_UNIT_OUTLINE}")
print(f"Book Path: {BOOK_PATH}")
print(f"Parsed UO Output Path: {PARSED_UO_JSON_PATH}")
print(f"Parsed ToC Output Path: {PRE_EXTRACTED_TOC_JSON_PATH}")
print(f"Vector DB Path: {CHROMA_PERSIST_DIR}")
print(f"Vector DB Collection: {CHROMA_COLLECTION_NAME}")
print("--- SETUP COMPLETE ---")
```

```
--- CONFIGURATION SUMMARY ---
Processing Mode: PDF
Unit ID: ICT311
Unit Outline Path:
/home/sebas_dev_linux/projects/course_generator/data/UO/ICT311 Applied
Cryptography.docx
Book Path: /home/sebas_dev_linux/projects/course_generator/data/books/(Chapman &
Hall_CRC Cryptography and Network Security Series) Jonathan Katz, Yehuda Lindell
- Introduction to Modern Cryptography-CRC Press (2020).pdf
Parsed UO Output Path:
/home/sebas_dev_linux/projects/course_generator/Parse_data/Parse_UO/ICT311
Applied Cryptography_parsed.json
Parsed ToC Output Path: /home/sebas_dev_linux/projects/course_generator/Parse_da
ta/Parse_TOC_books/ICT311_pdf_table_of_contents.json
Vector DB Path: /home/sebas_dev_linux/projects/course_generator/data/DataBase_Ch
roma/chroma_db_toc_guided_chunks_pdf_v2
Vector DB Collection: book_toc_guided_chunks_pdf_v2
--- SETUP COMPLETE ---
```

2 System Prompt

[19]:
```
UNIT_OUTLINE_SYSTEM_PROMPT_TEMPLATE = """
You are an expert academic assistant tasked with parsing a university unit␣
 ↪outline document and extracting key information into a structured JSON␣
 ↪format.

The input will be the raw text content of a unit outline. Your goal is to␣
 ↪identify and extract the following details and structure them precisely as␣
 ↪specified in the JSON schema below. Note: do not change any key name

**JSON Output Schema:**

```json
{{
  "unitInformation": {{
    "unitCode": "string | null",
    "unitName": "string | null",
    "creditPoints": "integer | null",
    "unitRationale": "string | null",
    "prerequisites": "string | null"
  }},
  "learningOutcomes": [
    "string"
  ],
  "assessments": [
    {{
      "taskName": "string",
      "description": "string",
      "dueWeek": "string | null",
      "weightingPercent": "integer | null",
      "learningOutcomesAssessed": "string | null"
    }}
  ],
  "weeklySchedule": [
    {{
      "week": "string",
      "contentTopic": "string",
      "requiredReading": "string | null"
    }}
  ],
  "requiredReadings": [
    "string"
  ],
  "recommendedReadings": [
    "string"
  ]
```
```

```
}}

Instructions for Extraction:
Unit Information: Locate Unit Code, Unit Name, Credit Points. Capture 'Unit␣
 ↪Overview / Rationale' as unitRationale. Identify prerequisites.
Learning Outcomes: Extract each learning outcome statement.
Assessments: Each task as an object. Capture full task name, description, Due␣
 ↪Week, Weighting % (number), and Learning Outcomes Assessed.
weeklySchedule: Each week as an object. Capture Week, contentTopic, and␣
 ↪requiredReading.
Required and Recommended Readings: List full text for each.
**Important Considerations for the LLM**:
Pay close attention to headings and table structures.
If information is missing, use null for string/integer fields, or an empty list␣
 ↪[] for array fields.
Do no change keys in the template given
Ensure the output is ONLY the JSON object, starting with {{{{ and ending with␣
 ↪}}}}. No explanations or conversational text before or after the JSON.
Now, parse the following unit outline text:
--- UNIT_OUTLINE_TEXT_START ---
{outline_text}
--- UNIT_OUTLINE_TEXT_END ---
"""
```

3 Extrac Unit outline details to process following steps - output raw json with UO details

[20]:
```
# Cell 3: Parse Unit Outline


# --- Helper Functions for Parsing ---
def extract_text_from_file(filepath: str) -> str:
    _, ext = os.path.splitext(filepath.lower())
    if ext == '.docx':
        doc = Document(filepath)
        full_text = [p.text for p in doc.paragraphs]
        for table in doc.tables:
            for row in table.rows:
                full_text.append(" | ".join(cell.text for cell in row.cells))
        return '\n'.join(full_text)
    elif ext == '.pdf':
        with pdfplumber.open(filepath) as pdf:
            return "\n".join(page.extract_text() for page in pdf.pages if page.
 ↪extract_text())
    else:
```

```
        raise TypeError(f"Unsupported file type: {ext}")

def parse_llm_json_output(content: str) -> dict:
    try:
        match = re.search(r'\{.*\}', content, re.DOTALL)
        if not match: return None
        return json.loads(match.group(0))
    except (json.JSONDecodeError, TypeError):
        return None

@retry(stop=stop_after_attempt(3), wait=wait_exponential(min=2, max=10))
def call_ollama_with_retry(client, prompt):
    logger.info(f"Calling Ollama model '{OLLAMA_MODEL}'...")
    response = client.chat(
        model=OLLAMA_MODEL,
        messages=[{"role": "user", "content": prompt}],
        format="json",
        options={"temperature": 0.0}
    )
    if not response or 'message' not in response or not response['message'].
 ↪get('content'):
        raise ValueError("Ollama returned an empty or invalid response.")
    return response['message']['content']

# --- Main Orchestration Function for this Cell ---
def parse_and_save_outline(input_filepath: str, output_filepath: str,␣
 ↪prompt_template: str):
    logger.info(f"Starting to process Unit Outline: {input_filepath}")

    if not os.path.exists(input_filepath):
        logger.error(f"Input file not found: {input_filepath}")
        return

    try:
        outline_text = extract_text_from_file(input_filepath)
        if not outline_text.strip():
            logger.error("Extracted text is empty. Aborting.")
            return

        prompt = prompt_template.format(outline_text=outline_text)
        client = ollama.Client(host=OLLAMA_HOST)
        llm_output = call_ollama_with_retry(client, prompt)
        parsed_data = parse_llm_json_output(llm_output)

        if parsed_data:
            os.makedirs(os.path.dirname(output_filepath), exist_ok=True)
            with open(output_filepath, 'w', encoding='utf-8') as f:
```

```
                json.dump(parsed_data, f, indent=2, ensure_ascii=False)
            logger.info(f"  Successfully parsed and saved Unit Outline to:␣
 ↪{output_filepath}")
        else:
            logger.error("  Failed to get valid structured data from the LLM.")

    except Exception as e:
        logger.error(f"An error occurred during parsing: {e}", exc_info=True)

# --- Execute Parsing ---
# Uses variables from Cell 1
parse_and_save_outline(
    input_filepath=FULL_PATH_UNIT_OUTLINE,
    output_filepath=PARSED_UO_JSON_PATH,
    prompt_template=UNIT_OUTLINE_SYSTEM_PROMPT_TEMPLATE
)
```

```
2025-06-19 17:27:59,004 - INFO - Starting to process Unit Outline:
/home/sebas_dev_linux/projects/course_generator/data/UO/ICT311 Applied
Cryptography.docx
2025-06-19 17:27:59,026 - INFO - Calling Ollama model 'mistral:latest'…
2025-06-19 17:28:48,071 - INFO - HTTP Request: POST
http://localhost:11434/api/chat "HTTP/1.1 200 OK"
2025-06-19 17:28:48,072 - INFO -   Successfully parsed and saved Unit Outline
to: /home/sebas_dev_linux/projects/course_generator/Parse_data/Parse_UO/ICT311
Applied Cryptography_parsed.json
```

4 Extract TOC from epub or epub

[21]:
```
# Cell 4: Extract Book Table of Contents (ToC)
# This cell extracts the ToC from the specified book (EPUB or PDF)
# and saves it to the path defined in Cell 1.

from ebooklib import epub, ITEM_NAVIGATION
from bs4 import BeautifulSoup
import fitz  # PyMuPDF
import json

# --- EPUB Extraction Logic ---
def parse_navpoint(navpoint, level=0):
    # (Your existing parse_navpoint function)
    title = navpoint.navLabel.text.strip()
    # Add filtering logic here if needed
    node = {"level": level, "title": title, "children": []}
    for child_navpoint in navpoint.find_all('navPoint', recursive=False):
        child_node = parse_navpoint(child_navpoint, level + 1)
        if child_node: node["children"].append(child_node)
```

```
    return node

def parse_li(li_element, level=0):
    # (Your existing parse_li function)
    a_tag = li_element.find('a')
    if a_tag:
        title = a_tag.get_text(strip=True)
        # Add filtering logic here if needed
        node = {"level": level, "title": title, "children": []}
        nested_ol = li_element.find('ol')
        if nested_ol:
            for sub_li in nested_ol.find_all('li', recursive=False):
                child_node = parse_li(sub_li, level + 1)
                if child_node: node["children"].append(child_node)
        return node
    return None

def extract_epub_toc(epub_path, output_json_path):
    print(f"Processing EPUB ToC for: {epub_path}")
    toc_data = []
    book = epub.read_epub(epub_path)
    for nav_item in book.get_items_of_type(ITEM_NAVIGATION):
        soup = BeautifulSoup(nav_item.get_content(), 'xml')
        if nav_item.get_name().endswith('.ncx'):
            print("INFO: Found EPUB 2 (NCX) Table of Contents.")
            navmap = soup.find('navMap')
            if navmap:
                for navpoint in navmap.find_all('navPoint', recursive=False):
                    node = parse_navpoint(navpoint)
                    if node: toc_data.append(node)
        else:
            print("INFO: Found EPUB 3 (XHTML) Table of Contents.")
            toc_nav = soup.select_one('nav[epub|type="toc"]')
            if toc_nav:
                top_ol = toc_nav.find('ol')
                if top_ol:
                    for li in top_ol.find_all('li', recursive=False):
                        node = parse_li(li)
                        if node: toc_data.append(node)
        if toc_data: break

    if toc_data:
        os.makedirs(os.path.dirname(output_json_path), exist_ok=True)
        with open(output_json_path, 'w', encoding='utf-8') as f:
            json.dump(toc_data, f, indent=2, ensure_ascii=False)
        print(f"  Successfully wrote EPUB ToC to: {output_json_path}")
    else:
```

```
        print("  WARNING: No ToC data extracted from EPUB.")

# --- PDF Extraction Logic ---
def build_pdf_hierarchy(toc_list):
    # (Your existing build_hierarchy function)
    root = []
    parent_stack = {0: {"children": root}}
    for level, title, page in toc_list:
        node = {"level": level, "title": title.strip(), "page": page,␣
 ↪"children": []}
        parent_node = parent_stack[level - 1]
        parent_node["children"].append(node)
        parent_stack[level] = node
    return root

def extract_pdf_toc(pdf_path, output_json_path):
    print(f"Processing PDF ToC for: {pdf_path}")
    try:
        doc = fitz.open(pdf_path)
        toc = doc.get_toc()
        if not toc:
            print("  WARNING: This PDF has no embedded bookmarks (ToC).")
            hierarchical_toc = []
        else:
            print(f"INFO: Found {len(toc)} bookmark entries.")
            hierarchical_toc = build_pdf_hierarchy(toc)

        os.makedirs(os.path.dirname(output_json_path), exist_ok=True)
        with open(output_json_path, 'w', encoding='utf-8') as f:
            json.dump(hierarchical_toc, f, indent=2, ensure_ascii=False)
        print(f"  Successfully wrote PDF ToC to: {output_json_path}")

    except Exception as e:
        print(f"An error occurred during PDF ToC extraction: {e}")

# --- Execute ToC Extraction ---
if PROCESS_EPUB:
    extract_epub_toc(BOOK_PATH, PRE_EXTRACTED_TOC_JSON_PATH)
else:
    extract_pdf_toc(BOOK_PATH, PRE_EXTRACTED_TOC_JSON_PATH)
```

```
Processing PDF ToC for:
/home/sebas_dev_linux/projects/course_generator/data/books/(Chapman & Hall_CRC
Cryptography and Network Security Series) Jonathan Katz, Yehuda Lindell -
Introduction to Modern Cryptography-CRC Press (2020).pdf
INFO: Found 290 bookmark entries.
  Successfully wrote PDF ToC to: /home/sebas_dev_linux/projects/course_generator
/Parse_data/Parse_TOC_books/ICT311_pdf_table_of_contents.json
```

5 Hirachical DB base on TOC

5.1 Process Book

[23]:
```python
# Cell 5: Create Hierarchical Vector Database
# This cell processes the book, enriches it with the hierarchical ToC,
# chunks it, and creates the final vector database.

import os
import json
import shutil
import logging
from typing import List, Dict, Any, Tuple
from langchain_core.documents import Document
from langchain_community.document_loaders import PyPDFLoader,␣
 ↪UnstructuredEPubLoader
from langchain_ollama.embeddings import OllamaEmbeddings
from langchain_chroma import Chroma
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Setup Logger for this cell
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -␣
 ↪%(message)s')
logger = logging.getLogger(__name__)

# --- Helper: Clean metadata values for ChromaDB ---
def clean_metadata_for_chroma(value: Any) -> Any:
    """Sanitizes metadata values to be compatible with ChromaDB."""
    if isinstance(value, list): return ", ".join(map(str, value))
    if isinstance(value, dict): return json.dumps(value)
    if isinstance(value, (str, int, float, bool)) or value is None: return value
    return str(value)

# --- Core Function to Process Book with Pre-extracted ToC ---
def process_book_with_extracted_toc(
    book_path: str,
    extracted_toc_json_path: str,
    chunk_size: int,
    chunk_overlap: int
) -> Tuple[List[Document], List[Dict[str, Any]]]:

    logger.info(f"Processing book '{os.path.basename(book_path)}' using ToC␣
 ↪from '{os.path.basename(extracted_toc_json_path)}'.")

    # 1. Load the pre-extracted hierarchical ToC
    try:
        with open(extracted_toc_json_path, 'r', encoding='utf-8') as f:
            hierarchical_toc = json.load(f)
```

```
        if not hierarchical_toc:
            logger.error(f"Pre-extracted ToC at '{extracted_toc_json_path}' is␣
↪empty or invalid.")
            return [], []
        logger.info(f"Successfully loaded pre-extracted ToC with␣
↪{len(hierarchical_toc)} top-level entries.")
    except Exception as e:
        logger.error(f"Error loading pre-extracted ToC JSON: {e}",␣
↪exc_info=True)
        return [], []

    # 2. Load all text elements/pages from the book
    all_raw_book_docs: List[Document] = []
    _, file_extension = os.path.splitext(book_path.lower())

    if file_extension == ".epub":
        loader = UnstructuredEPubLoader(book_path, mode="elements",␣
↪strategy="fast")
        try:
            all_raw_book_docs = loader.load()
            logger.info(f"Loaded {len(all_raw_book_docs)} text elements from␣
↪EPUB.")
        except Exception as e:
            logger.error(f"Error loading EPUB content: {e}", exc_info=True)
            return [], hierarchical_toc
    elif file_extension == ".pdf":
        loader = PyPDFLoader(book_path)
        try:
            all_raw_book_docs = loader.load()
            logger.info(f"Loaded {len(all_raw_book_docs)} pages from PDF.")
        except Exception as e:
            logger.error(f"Error loading PDF content: {e}", exc_info=True)
            return [], hierarchical_toc
    else:
        logger.error(f"Unsupported book file format: {file_extension}")
        return [], hierarchical_toc

    if not all_raw_book_docs:
        logger.error("No text elements/pages loaded from the book.")
        return [], hierarchical_toc

    # 3. Create enriched LangChain Documents by matching ToC to content
    final_documents_with_metadata: List[Document] = []

    # Flatten the ToC for easier iteration and path tracking
    flat_toc_entries: List[Dict[str, Any]] = []
```

```
  def _flatten_toc_recursive(nodes: List[Dict[str,Any]], current_titles_path:␣
↪List[str]):
      for node in nodes:
          title = node.get("title","").strip()
          if not title: continue

          new_titles_path = current_titles_path + [title]
          entry = {
              "titles_path": new_titles_path,
              "level": node.get("level"),
              "full_title_for_matching": title
          }
          if "page" in node: entry["page"] = node["page"]
          flat_toc_entries.append(entry)

          if node.get("children"):
              _flatten_toc_recursive(node.get("children", []),␣
↪new_titles_path)

  _flatten_toc_recursive(hierarchical_toc, [])
  logger.info(f"Flattened ToC into {len(flat_toc_entries)} entries for␣
↪matching.")

  # Logic for PDF metadata assignment
  if file_extension == ".pdf" and any("page" in entry for entry in␣
↪flat_toc_entries):
      logger.info("Assigning metadata to PDF pages based on ToC page numbers..
↪.")
      flat_toc_entries.sort(key=lambda x: x.get("page", -1) if x.get("page")␣
↪is not None else -1)

      for page_doc in all_raw_book_docs:
          page_num_0_indexed = page_doc.metadata.get("page", -1)
          page_num_1_indexed = page_num_0_indexed + 1

          assigned_metadata = {"source": os.path.basename(book_path),␣
↪"page_number": page_num_1_indexed}
          best_match_toc_entry = None

          for toc_entry in flat_toc_entries:
              toc_page = toc_entry.get("page")
              if toc_page is not None and toc_page <= page_num_1_indexed:
                  if best_match_toc_entry is None or toc_page >␣
↪best_match_toc_entry.get("page", -1):
                      best_match_toc_entry = toc_entry
              elif toc_page is not None and toc_page > page_num_1_indexed:
```

```
                    break

            if best_match_toc_entry:
                for i, title_in_path in␣
↪enumerate(best_match_toc_entry["titles_path"]):
                    assigned_metadata[f"level_{i+1}_title"] = title_in_path
            else:
                assigned_metadata["level_1_title"] = "Uncategorized PDF Page"

            cleaned_meta = {k: clean_metadata_for_chroma(v) for k,v in␣
↪assigned_metadata.items()}
            final_documents_with_metadata.append(Document(page_content=page_doc.
↪page_content, metadata=cleaned_meta))

    # Logic for EPUB metadata assignment
    elif file_extension == ".epub":
        logger.info("Assigning metadata to EPUB elements by matching ToC titles␣
↪in text...")
        toc_titles_for_search = [entry for entry in flat_toc_entries if entry.
↪get("full_title_for_matching")]
        current_hierarchy_metadata = {}

        for element_doc in all_raw_book_docs:
            element_text = element_doc.page_content.strip() if element_doc.
↪page_content else ""
            if not element_text: continue

            found_new_heading = False
            for toc_entry in toc_titles_for_search:
                if element_text == toc_entry["full_title_for_matching"]:
                    current_hierarchy_metadata = {"source": os.path.
↪basename(book_path)}
                    for i, title_in_path in enumerate(toc_entry["titles_path"]):
                        current_hierarchy_metadata[f"level_{i+1}_title"] =␣
↪title_in_path
                    if "page" in toc_entry:␣
↪current_hierarchy_metadata["epub_toc_page"] = toc_entry["page"]
                    found_new_heading = True
                    break

            doc_metadata_to_assign = current_hierarchy_metadata.copy() if␣
↪current_hierarchy_metadata else {"source": os.path.basename(book_path),␣
↪"level_1_title": "EPUB Preamble"}

            cleaned_meta = {k: clean_metadata_for_chroma(v) for k,v in␣
↪doc_metadata_to_assign.items()}
```

```
            final_documents_with_metadata.
 ↪append(Document(page_content=element_text, metadata=cleaned_meta))

    else: # Fallback for unsupported types or logic failure
        final_documents_with_metadata = all_raw_book_docs

    if not final_documents_with_metadata:
        logger.error("No documents were processed or enriched with hierarchical␣
 ↪metadata.")
        return [], hierarchical_toc

    logger.info(f"Total documents prepared for chunking:␣
 ↪{len(final_documents_with_metadata)}")

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap,
        length_function=len
    )
    final_chunks = text_splitter.split_documents(final_documents_with_metadata)
    logger.info(f"Split into {len(final_chunks)} final chunks, inheriting␣
 ↪hierarchical metadata.")

    return final_chunks, hierarchical_toc

# --- Main Execution Block for this Cell ---

# Use the global variables defined in Cell 1
if not os.path.exists(PRE_EXTRACTED_TOC_JSON_PATH):
    logger.error(f"CRITICAL: Pre-extracted ToC file not found at␣
 ↪'{PRE_EXTRACTED_TOC_JSON_PATH}'.")
    logger.error("Please run the 'Extract Book Table of Contents (ToC)' cell␣
 ↪(Cell 4) first.")
else:
    # Process the book to get the chunks
    final_chunks_for_db, toc_reloaded = process_book_with_extracted_toc(
        book_path=BOOK_PATH,
        extracted_toc_json_path=PRE_EXTRACTED_TOC_JSON_PATH,
        chunk_size=CHUNK_SIZE,
        chunk_overlap=CHUNK_OVERLAP
    )

    if final_chunks_for_db:
        # Delete old DB if it exists
        if os.path.exists(CHROMA_PERSIST_DIR):
```

```
            logger.warning(f"Deleting existing ChromaDB directory:␣
↪{CHROMA_PERSIST_DIR}")
            shutil.rmtree(CHROMA_PERSIST_DIR)

        # Create and persist the new vector database
        logger.info(f"Initializing embedding model '{EMBEDDING_MODEL_OLLAMA}'␣
↪and creating new vector database...")
        embedding_model = OllamaEmbeddings(model=EMBEDDING_MODEL_OLLAMA)

        vector_db = Chroma.from_documents(
            documents=final_chunks_for_db,
            embedding=embedding_model,
            persist_directory=CHROMA_PERSIST_DIR,
            collection_name=CHROMA_COLLECTION_NAME
        )

        # Verify creation
        reloaded_db = Chroma(persist_directory=CHROMA_PERSIST_DIR,␣
↪embedding_function=embedding_model, collection_name=CHROMA_COLLECTION_NAME)
        count = reloaded_db._collection.count()

        print("-" * 50)
        logger.info(f"  Vector DB created successfully at:␣
↪{CHROMA_PERSIST_DIR}")
        logger.info(f"  Collection '{CHROMA_COLLECTION_NAME}' contains {count}␣
↪documents.")
        print("-" * 50)
    else:
        logger.error("  Failed to generate chunks. Vector DB not created.")
```

```
2025-06-19 17:34:39,134 - INFO - Processing book '(Chapman & Hall_CRC
Cryptography and Network Security Series) Jonathan Katz, Yehuda Lindell -
Introduction to Modern Cryptography-CRC Press (2020).pdf' using ToC from
'ICT311_pdf_table_of_contents.json'.
2025-06-19 17:34:39,135 - INFO - Successfully loaded pre-extracted ToC with 16
top-level entries.

2025-06-19 17:34:50,405 - INFO - Loaded 649 pages from PDF.
2025-06-19 17:34:50,405 - INFO - Flattened ToC into 290 entries for matching.
2025-06-19 17:34:50,406 - INFO - Assigning metadata to PDF pages based on ToC
page numbers…
2025-06-19 17:34:50,415 - INFO - Total documents prepared for chunking: 649
2025-06-19 17:34:50,443 - INFO - Split into 2353 final chunks, inheriting
hierarchical metadata.
2025-06-19 17:34:50,448 - INFO - Initializing embedding model 'nomic-embed-text'
and creating new vector database…
2025-06-19 17:34:50,463 - INFO - Anonymized telemetry enabled. See
https://docs.trychroma.com/telemetry for more information.
```

```
2025-06-19 17:35:19,272 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-19 17:35:22,588 - INFO -   Vector DB created successfully at: /home/seba
s_dev_linux/projects/course_generator/data/DataBase_Chroma/chroma_db_toc_guided_
chunks_pdf_v2
2025-06-19 17:35:22,588 - INFO -   Collection 'book_toc_guided_chunks_pdf_v2'
contains 2353 documents.

--------------------------------------------------
--------------------------------------------------
```

5.2 Test Data Base for content development

5.2.1 Verification Test Strategy

The script automatically validates the vector database by performing four dynamic tests that increase in complexity, moving from a general health check to specific application-level requirements.

Basic Retrieval Test: - Goal: Confirm the database is live and its content is broadly relevant to the course subject. - Method: It performs a simple search using the course's unitName (e.g., "Digital Forensic") extracted from the unit outline. - Success means: The database is online, and the ingested content is thematically correct.

Deep Hierarchy Test: - Goal: Verify the structural integrity of the metadata, ensuring text is correctly tagged with its full, multi-level context (e.g., Part -> Chapter -> Section). - Method: It randomly picks a deeply nested sub-section from the Table of Contents and performs a search that is filtered to match that exact hierarchical path. - Success means: The data ingestion process is correctly assigning detailed, nested parentage to all text chunks.

Advanced Unit Outline Alignment Test: - Goal: Ensure the system can correctly map a weekly syllabus topic to the right chapter(s) in the book, adapting to different ToC structures (e.g., flat chapters vs. chapters inside "Parts"). - Method: It randomly selects a week, finds all required chapter numbers from the reading list, and dynamically determines the correct metadata level to check. It then verifies that a search for the weekly topic retrieves chunks belonging to the correct chapters. - Success means: The database is directly useful for its primary purpose: linking the course structure to the source textbook reliably.

Content Sequence Test (PDF-only): - Goal: Check if retrieved content can be re-ordered chronologically to form a coherent narrative. - Method: It retrieves multiple chunks for a random topic, sorts them using the page_number metadata, and verifies the page numbers are in ascending order. - Success means: The database contains the necessary metadata to reconstruct the original flow of the book's content, which is crucial for generating logical summaries or lecture material.

[2]:
```python
# Cell 6: Verify Vector Database (Final Version for All ToC Structures)

import os
import json
import re
import random
import logging
from typing import List, Dict, Any, Tuple, Optional
```

```
# Third-party imports
try:
    from langchain_chroma import Chroma
    from langchain_ollama.embeddings import OllamaEmbeddings
    langchain_available = True
except ImportError:
    langchain_available = False

# Setup Logger for this cell
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -␣
 ↪%(message)s')
logger = logging.getLogger(__name__)

# --- HELPER FUNCTIONS (UNCHANGED) ---

def print_header(text: str, char: str = "="):
    print("\n" + char * 80)
    print(text.center(80))
    print(char * 80)

def print_results(query_text: str, results: list, where_filter: Optional[Dict]␣
 ↪= None):
    print(f"\n Query: '{query_text}'")
    if where_filter:
        print(f" Filter: {json.dumps(where_filter, indent=2)}")
    if not results:
        print("\n No results found for this query and filter.")
        return
    print(f"\n Found {len(results)} results. Displaying top {min(len(results),␣
 ↪3)}:")
    for i, doc in enumerate(results[:3]):
        print(f"\n--- Result {i+1} ---")
        content_preview = doc.page_content.replace('\n', ' ').strip()
        print(f"Content: '{content_preview[:150]}...'")
        print(f"Metadata: {json.dumps(doc.metadata, indent=2)}")

def find_deep_entry(nodes: List[Dict], current_path: List[str] = []) ->␣
 ↪Optional[Tuple[Dict, List[str]]]:
    shuffled_nodes = random.sample(nodes, len(nodes))
    for node in shuffled_nodes:
        if node.get('level', 0) >= 2 and node.get('children'): return node,␣
 ↪current_path + [node['title']]
        if node.get('children'):
            path = current_path + [node['title']]
            deep_entry = find_deep_entry(node['children'], path)
            if deep_entry: return deep_entry
```

```
    return None

def find_chapter_title_by_number(toc_data: List[Dict], chap_num: int) ->␣
 ↪Optional[List[str]]:
    def search_nodes(nodes, num, current_path):
        for node in nodes:
            path = current_path + [node['title']]
            # This regex handles "Chapter 1", "1: Title", "1. Title" etc.
            if re.match(rf"(Chapter\s)?{num}[.:\s]", node.get('title', ''), re.
 ↪IGNORECASE):
                return path
            if node.get('children'):
                found_path = search_nodes(node['children'], num, path)
                if found_path: return found_path
        return None
    return search_nodes(toc_data, chap_num, [])

# --- TEST CASE FUNCTIONS ---

def advanced_alignment_test(db, outline, toc):
    print_header("Test 3: Advanced Unit Outline Alignment", char="-")
    logger.info("  Description: Checks if a weekly topic maps to its required␣
 ↪chapter(s).")
    try:
        week_to_test = random.choice(outline['weeklySchedule'])
        logger.info(f"Testing alignment for Week {week_to_test['week']}:␣
 ↪'{week_to_test['contentTopic']}'")

        reading = week_to_test.get('requiredReading', '')
        chap_nums = re.findall(r'\d+', reading)
        assert chap_nums, f"Could not find chapter numbers in reading:␣
 ↪'{reading}'"

        chapter_paths = [find_chapter_title_by_number(toc, int(n)) for n in␣
 ↪chap_nums]
        chapter_paths = [path for path in chapter_paths if path is not None]
        assert chapter_paths, f"Could not map any chapter numbers {chap_nums}␣
 ↪to a title in the ToC."

        # --- FIX IS HERE: This logic now handles both flat and nested ToCs ---

        # The chapter title is always the LAST element in the path.
        expected_chapter_titles = [path[-1] for path in chapter_paths]

        # The metadata level where the chapter title is stored depends on the␣
 ↪path's length.
```

```
        # e.g., if path is ['Chapter 3...'], length is 1, so it's in␣
↪'level_1_title'.
        # e.g., if path is ['Part I...', 'Chapter 3...'], length is 2, so it's␣
↪in 'level_2_title'.
        chapter_level = len(chapter_paths[0])
        chapter_metadata_key = f"level_{chapter_level}_title"

        # The filter should be based on the FIRST element of the path (the␣
↪top-level entry).
        level_1_titles = list(set([path[0] for path in chapter_paths]))

        query = week_to_test['contentTopic']
        or_filter = [{"level_1_title": {"$eq": title}} for title in␣
↪level_1_titles]
        w_filter = {"$or": or_filter} if len(or_filter) > 1 else or_filter[0]

        results = db.similarity_search(query, k=5, filter=w_filter)
        print_results(query, results, w_filter)
        assert len(results) > 0, "Alignment query returned no results for the␣
↪correct section/chapter."

        # Check if any of the retrieved documents have the correct Chapter␣
↪title at the correct level.
        is_match_found = any(
            doc.metadata.get(chapter_metadata_key) in expected_chapter_titles␣
↪for doc in results
        )
        assert is_match_found, f"None of the top results had the expected␣
↪chapter title in metadata field '{chapter_metadata_key}'."
        # --- END OF FIX ---

        logger.info("  TEST PASSED")
        return True
    except Exception as e:
        logger.error(f"  TEST FAILED: {e}", exc_info=True)
        return False

# (The other test functions remain the same)
def basic_retrieval_test(db, outline):

    print_header("Test 1: Basic Retrieval (Dynamic)", char="-")
    logger.info("  Description: Checks if a query using the Unit Name retrieves␣
↪any content.")
    try:
        query_text = outline.get("unitInformation", {}).get("unitName",␣
↪"introduction")
```

```
        logger.info(f"Using Unit Name for basic query: '{query_text}'")
        results = db.similarity_search(query_text, k=1)
        print_results(query_text, results)
        assert len(results) > 0, f"Basic retrieval for '{query_text}' should␣
 ↪return at least one chunk."
        logger.info("  TEST PASSED")
        return True
    except Exception as e:
        logger.error(f"  TEST FAILED: {e}", exc_info=True)
        return False

def deep_hierarchy_test(db, toc):

    print_header("Test 2: Deep Hierarchy Retrieval", char="-")
    logger.info("  Description: Checks if a deep sub-section is tagged with its␣
 ↪full path.")
    try:
        deep_entry_result = find_deep_entry(toc)
        assert deep_entry_result, "Could not find a deep entry (level >= 2) in␣
 ↪the ToC to test."
        node, path = deep_entry_result
        query = node['title']
        conditions = [{f"level_{i+1}_title": {"$eq": title}} for i, title in␣
 ↪enumerate(path)]
        w_filter = {"$and": conditions}
        results = db.similarity_search(query, k=1, filter=w_filter)
        print_results(query, results, w_filter)
        assert len(results) > 0, "Deeply filtered query returned no results."
        logger.info("  TEST PASSED")
        return True
    except Exception as e:
        logger.error(f"  TEST FAILED: {e}", exc_info=True)
        return False

def sequential_content_test(db, outline):

    print_header("Test 4: Content Sequence Verification (PDF-only)", char="-")
    logger.info("  Description: Checks if retrieved chunks can be ordered by␣
 ↪page number.")
    try:
        topic_query = random.choice(outline['weeklySchedule'])['contentTopic']
        logger.info(f"Testing content sequence for topic: '{topic_query}'")
        results = db.similarity_search(topic_query, k=10)
        docs_with_pages = [doc for doc in results if doc.metadata.
 ↪get("page_number") is not None]
```

```
        assert len(docs_with_pages) > 3, "Fewer than 4 retrieved chunks have␣
↪page numbers."
        docs_with_pages.sort(key=lambda x: x.metadata["page_number"])
        page_numbers = [doc.metadata["page_number"] for doc in docs_with_pages]
        print_results(topic_query, results)
        print(f"Retrieved and sorted page numbers: {page_numbers}")
        is_sorted = all(page_numbers[i] <= page_numbers[i+1] for i in␣
↪range(len(page_numbers)-1))
        assert is_sorted, "The retrieved chunks' page numbers are not in␣
↪ascending order."
        logger.info("  Page numbers are in correct ascending order.")
        return True
    except Exception as e:
        logger.error(f"  TEST FAILED: {e}", exc_info=True)
        return False

# --- MAIN VERIFICATION EXECUTION ---
def run_verification():
    print_header("Database Verification Process")
    if not langchain_available: logger.error("LangChain libraries not found.");␣
↪return
    if not all([os.path.exists(p) for p in [CHROMA_PERSIST_DIR,␣
↪PRE_EXTRACTED_TOC_JSON_PATH]]):
        logger.error("  Required file/directory is missing. Run Cells 4 and 5␣
↪first."); return
    with open(PRE_EXTRACTED_TOC_JSON_PATH, 'r', encoding='utf-8') as f:␣
↪toc_data = json.load(f)
    if not os.path.exists(PARSED_UO_JSON_PATH):
        logger.error(f"  Parsed Unit Outline not found. Cannot run all dynamic␣
↪tests."); return
    with open(PARSED_UO_JSON_PATH, 'r', encoding='utf-8') as f:␣
↪unit_outline_data = json.load(f)

    logger.info(f"Initializing embedding model '{EMBEDDING_MODEL_OLLAMA}'...")
    embeddings = OllamaEmbeddings(model=EMBEDDING_MODEL_OLLAMA)
    vector_store = Chroma(persist_directory=CHROMA_PERSIST_DIR,␣
↪embedding_function=embeddings, collection_name=CHROMA_COLLECTION_NAME)

    # Run tests and count results
    results_summary = []
    results_summary.append(basic_retrieval_test(vector_store,␣
↪unit_outline_data))
    results_summary.append(deep_hierarchy_test(vector_store, toc_data))
    results_summary.append(advanced_alignment_test(vector_store,␣
↪unit_outline_data, toc_data))
```

```
    if not PROCESS_EPUB: # Global variable from Cell 1
        results_summary.append(sequential_content_test(vector_store,␣
↪unit_outline_data))
    else:
        logger.info("\n  Skipping Content Sequence test (for PDF-based sources␣
↪only).")

    # Final Report
    passed_count = sum(results_summary)
    failed_count = len(results_summary) - passed_count
    print_header("Verification Summary")
    print(f"Total Tests Run: {len(results_summary)}")
    print(f"  Passed: {passed_count}")
    print(f"  Failed: {failed_count}")
    print_header("Verification Complete", char="=")

# --- Execute Verification ---
run_verification()
```

```
2025-06-19 17:37:06,572 - INFO - Initializing embedding model 'nomic-embed-
text'…
2025-06-19 17:37:06,586 - INFO - Anonymized telemetry enabled. See
https://docs.trychroma.com/telemetry for more information.
2025-06-19 17:37:06,642 - INFO -   Description: Checks if a query using the
Unit Name retrieves any content.
2025-06-19 17:37:06,643 - INFO - Using Unit Name for basic query: 'Applied
Cryptography'


================================================================================
                          Database Verification Process
================================================================================

--------------------------------------------------------------------------------
                       Test 1: Basic Retrieval (Dynamic)
--------------------------------------------------------------------------------

2025-06-19 17:37:06,788 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-19 17:37:06,807 - INFO -   TEST PASSED
2025-06-19 17:37:06,808 - INFO -   Description: Checks if a deep sub-section is
tagged with its full path.
2025-06-19 17:37:06,883 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-19 17:37:06,886 - INFO -   TEST PASSED
2025-06-19 17:37:06,887 - INFO -   Description: Checks if a weekly topic maps
to its required chapter(s).
2025-06-19 17:37:06,887 - INFO - Testing alignment for Week 6: 'Number Theory
```

```
and Cryptographic Hardness Assumptions'
2025-06-19 17:37:06,915 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-19 17:37:06,921 - INFO -   TEST PASSED
2025-06-19 17:37:06,921 - INFO -   Description: Checks if retrieved chunks can
be ordered by page number.
2025-06-19 17:37:06,921 - INFO - Testing content sequence for topic:
'Introduction and Classical Cryptography'
2025-06-19 17:37:06,937 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-19 17:37:06,940 - INFO -   Page numbers are in correct ascending order.


  Query: 'Applied Cryptography'

  Found 1 results. Displaying top 1:

--- Result 1 ---
Content: 'Introduction to Modern  Cryptography…'
Metadata: {
  "level_1_title": "Half Title",
  "page_number": 2,
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf"
}

--------------------------------------------------------------------------------
                        Test 2: Deep Hierarchy Retrieval
--------------------------------------------------------------------------------

  Query: '2: Perfectly Secret Encryption'
  Filter: {
   "$and": [
     {
       "level_1_title": {
         "$eq": "I: Introduction and Classical Cryptography"
       }
     },
     {
       "level_2_title": {
         "$eq": "2: Perfectly Secret Encryption"
       }
     }
   ]
}

  Found 1 results. Displaying top 1:
```

```
--- Result 1 ---
Content: 'Chapter 2 Perfectly Secret Encryption In the previous chapter we
presented some historical encryption schemes and showed that they can be broken
easil…'
Metadata: {
  "level_1_title": "I: Introduction and Classical Cryptography",
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf",
  "page_number": 46,
  "level_2_title": "2: Perfectly Secret Encryption"
}

--------------------------------------------------------------------------------
                    Test 3: Advanced Unit Outline Alignment
--------------------------------------------------------------------------------

  Query: 'Number Theory and Cryptographic Hardness Assumptions'
  Filter: {
   "level_1_title": {
     "$eq": "III: Public-Key (Asymmetric) Cryptography"
   }
}

  Found 5 results. Displaying top 3:

--- Result 1 ---
Content: 'Chapter 9 Number Theory and Cryptographic Hardness Assumptions Modern
cryptosystems are invariably based on an assumption that some problem is hard.
I…'
Metadata: {
  "page_number": 328,
  "level_1_title": "III: Public-Key (Asymmetric) Cryptography",
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf",
  "level_2_title": "9: Number Theory and Cryptographic Hardness Assumptions"
}

--- Result 2 ---
Content: 'Number Theory and Cryptographic Hardness Assumptions 359 there is a
collision; i.e., x = x  and Hs(x) = Hs(x ). Lemma 9.65 implies that whenever
ther…'
Metadata: {
  "level_3_title": "References and Additional Reading",
  "page_number": 382,
  "level_1_title": "III: Public-Key (Asymmetric) Cryptography",
```

```
  "level_2_title": "9: Number Theory and Cryptographic Hardness Assumptions",
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf"
}

--- Result 3 ---
Content: 'Number Theory and Cryptographic Hardness Assumptions 355 9.4.1 One-Way
Functions and Permutations One-way functions are the minimal cryptographic
prim…'
Metadata: {
  "level_2_title": "9: Number Theory and Cryptographic Hardness Assumptions",
  "level_1_title": "III: Public-Key (Asymmetric) Cryptography",
  "page_number": 378,
  "level_3_title": "9.4 *Cryptographic Applications",
  "level_4_title": "9.4.1 One-Way Functions and Permutations",
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf"
}

--------------------------------------------------------------------------------
                Test 4: Content Sequence Verification (PDF-only)
--------------------------------------------------------------------------------

  Query: 'Introduction and Classical Cryptography'

  Found 10 results. Displaying top 3:

--- Result 1 ---
Content: 'Part I Introduction and Classical Cryptography…'
Metadata: {
  "level_1_title": "I: Introduction and Classical Cryptography",
  "page_number": 22,
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf"
}

--- Result 2 ---
Content: 'Introduction to Modern  Cryptography…'
Metadata: {
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf",
  "level_1_title": "Half Title",
  "page_number": 2
}
```

```
--- Result 3 ---
Content: '(or lack thereof) and the mathematical assumptions underlying those
proofs. It is not our intention for readers to become experts-or to be able to
de-…'
Metadata: {
  "level_2_title": "1: Introduction",
  "level_3_title": "1.2 The Setting of Private-Key Encryption",
  "source": "(Chapman & Hall_CRC Cryptography and Network Security Series)
Jonathan Katz, Yehuda Lindell - Introduction to Modern Cryptography-CRC Press
(2020).pdf",
  "level_1_title": "I: Introduction and Classical Cryptography",
  "page_number": 25
}
Retrieved and sorted page numbers: [2, 8, 16, 16, 19, 22, 25, 25, 25, 641]

================================================================================
                              Verification Summary
================================================================================
Total Tests Run: 4
  Passed: 4
  Failed: 0

================================================================================
                             Verification Complete
================================================================================
```