book_to_slide_BY_sections_V2

June 26, 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 = "ICT312 Digital Forensic_Final.docx" # epub
# UNIT_OUTLINE_FILENAME = "ICT311 Applied Cryptography.docx" # pdf


# 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"
```

```
# --- 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 = "qwen3:8b" # "qwen3:8b", #"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: EPUB
Unit ID: ICT312
Unit Outline Path:
/home/sebas_dev_linux/projects/course_generator/data/UO/ICT312 Digital
Forensic_Final.docx
Book Path: /home/sebas_dev_linux/projects/course_generator/data/books/Bill
Nelson, Amelia Phillips, Christopher Steuart - Guide to Computer Forensics and
Investigations_ Processing Digital Evidence-Cengage Learning (2018).epub
Parsed UO Output Path:
/home/sebas_dev_linux/projects/course_generator/Parse_data/Parse_UO/ICT312
Digital Forensic_Final_parsed.json
Parsed ToC Output Path: /home/sebas_dev_linux/projects/course_generator/Parse_da
ta/Parse_TOC_books/ICT312_epub_table_of_contents.json
Vector DB Path: /home/sebas_dev_linux/projects/course_generator/data/DataBase_Ch
roma/chroma_db_toc_guided_chunks_epub_v2
Vector DB Collection: book_toc_guided_chunks_epub_v2
```

```
--- SETUP COMPLETE ---
```

2 System Prompt

[2]:
```
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]:
```python
# Place this in a new cell after your imports, or within Cell 3 before the␣
 ↪functions.
# This code is based on the schema from your screenshot on page 4.

from pydantic import BaseModel, Field, ValidationError
from typing import List, Optional
import time

# Define Pydantic models that match your JSON schema
class UnitInformation(BaseModel):
    unitCode: Optional[str] = None
    unitName: Optional[str] = None
    creditPoints: Optional[int] = None
    unitRationale: Optional[str] = None
    prerequisites: Optional[str] = None

class Assessment(BaseModel):
    taskName: str
    description: str
    dueWeek: Optional[str] = None
```

```
    weightingPercent: Optional[int] = None
    learningOutcomesAssessed: Optional[str] = None

class WeeklyScheduleItem(BaseModel):
    week: str
    contentTopic: str
    requiredReading: Optional[str] = None

class ParsedUnitOutline(BaseModel):
    unitInformation: UnitInformation
    learningOutcomes: List[str]
    assessments: List[Assessment]
    weeklySchedule: List[WeeklyScheduleItem]
    requiredReadings: List[str]
    recommendedReadings: List[str]
```

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

[4]:
```
# 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
```

```python
@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_robust(
    input_filepath: str,
    output_filepath: str,
    prompt_template: str,
    max_retries: int = 3
):
    logger.info(f"Starting to robustly 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
    except Exception as e:
        logger.error(f"Failed to extract text from file: {e}", exc_info=True)
        return

    client = ollama.Client(host=OLLAMA_HOST)
    current_prompt = prompt_template.format(outline_text=outline_text)

    for attempt in range(max_retries):
        logger.info(f"Attempt {attempt + 1}/{max_retries} to parse outline.")

        try:
            # Call the LLM
            llm_output_str = call_ollama_with_retry(client, current_prompt)
```

```
            # Find the JSON blob in the response
            json_blob = parse_llm_json_output(llm_output_str) # Your existing␣
↪helper
            if not json_blob:
                raise ValueError("LLM did not return a parsable JSON object.")

            # *** THE KEY VALIDATION STEP ***
            # Try to parse the dictionary into your Pydantic model.
            # This will raise a `ValidationError` if keys are wrong, types are␣
↪wrong, or fields are missing.
            parsed_data = ParsedUnitOutline.model_validate(json_blob)

            # If successful, save the validated data and exit the loop
            logger.info("Successfully validated JSON structure against Pydantic␣
↪model.")
            os.makedirs(os.path.dirname(output_filepath), exist_ok=True)
            with open(output_filepath, 'w', encoding='utf-8') as f:
                # Use .model_dump_json() for clean, validated output
                f.write(parsed_data.model_dump_json(indent=2))

            logger.info(f"Successfully parsed and saved Unit Outline to:␣
↪{output_filepath}")
            return # Exit function on success

        except ValidationError as e:
            logger.warning(f"Validation failed on attempt {attempt + 1}. Error:␣
↪{e}")
            # Formulate a new prompt with the error message for self-correction
            error_feedback = (
                f"\n\nYour previous attempt failed. You MUST correct the␣
↪following errors:\n"
                f"{e}\n\n"
                f"Please regenerate the entire JSON object, ensuring it␣
↪strictly adheres to the schema "
                f"and corrects these specific errors. Do not change any key␣
↪names."
            )
            current_prompt = current_prompt + error_feedback # Append the error␣
↪to the prompt

        except Exception as e:
            # Catch other errors like network issues from call_ollama_with_retry
            logger.error(f"An unexpected error occurred on attempt {attempt +␣
↪1}: {e}", exc_info=True)
            # You might want to wait before retrying for non-validation errors
            time.sleep(5)
```

```
    logger.error(f"Failed to get valid structured data from the LLM after␣
 ↪{max_retries} attempts.")


# --- In your execution block, call the new function ---
# parse_and_save_outline(...) becomes:
parse_and_save_outline_robust(
    input_filepath=FULL_PATH_UNIT_OUTLINE,
    output_filepath=PARSED_UO_JSON_PATH,
    prompt_template=UNIT_OUTLINE_SYSTEM_PROMPT_TEMPLATE
)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[4], line 115
    110     logger.error(f"Failed to get valid structured data from the LLM␣
 ↪after {max_retries} attempts.")
    113 # --- In your execution block, call the new function ---
    114 # parse_and_save_outline(…) becomes:
--> 115 parse_and_save_outline_robust(
    116     input_filepath=FULL_PATH_UNIT_OUTLINE,
    117     output_filepath=PARSED_UO_JSON_PATH,
    118     prompt_template=UNIT_OUTLINE_SYSTEM_PROMPT_TEMPLATE
    119 )

Cell In[4], line 71, in parse_and_save_outline_robust(input_filepath,␣
 ↪output_filepath, prompt_template, max_retries)
     67 logger.info(f"Attempt {attempt + 1}/{max_retries} to parse outline.")
     69 try:
     70     # Call the LLM
---> 71     llm_output_str = call_ollama_with_retry(client, current_prompt)
     73     # Find the JSON blob in the response
     74     json_blob = parse_llm_json_output(llm_output_str) # Your existing␣
 ↪helper

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/tenacity/__init__.py:
 ↪336, in BaseRetrying.wraps.<locals>.wrapped_f(*args, **kw)
    334 copy = self.copy()
    335 wrapped_f.statistics = copy.statistics  # type: ignore[attr-defined]
--> 336 return copy(f, *args, **kw)

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/tenacity/__init__.py:
 ↪475, in Retrying.__call__(self, fn, *args, **kwargs)
    473 retry_state = RetryCallState(retry_object=self, fn=fn, args=args,␣
 ↪kwargs=kwargs)
    474 while True:
```

```
--> 475     do = self.iter(retry_state=retry_state)
    476     if isinstance(do, DoAttempt):
    477         try:

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/tenacity/__init__.py:
 ↪376, in BaseRetrying.iter(self, retry_state)
    374 result = None
    375 for action in self.iter_state.actions:
--> 376     result = action(retry_state)
    377 return result

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/tenacity/__init__.py:
 ↪398, in BaseRetrying._post_retry_check_actions.<locals>.<lambda>(rs)
    396 def _post_retry_check_actions(self, retry_state: "RetryCallState") ->␣
 ↪None:
    397     if not (self.iter_state.is_explicit_retry or self.iter_state.
 ↪retry_run_result):
--> 398         self._add_action_func(lambda rs: rs.outcome.result())
    399         return
    401     if self.after is not None:

File ~/miniconda3/envs/tensor2/lib/python3.10/concurrent/futures/_base.py:451,␣
 ↪in Future.result(self, timeout)
    449     raise CancelledError()
    450 elif self._state == FINISHED:
--> 451     return self.__get_result()
    453 self._condition.wait(timeout)
    455 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

File ~/miniconda3/envs/tensor2/lib/python3.10/concurrent/futures/_base.py:403,␣
 ↪in Future.__get_result(self)
    401 if self._exception:
    402     try:
--> 403         raise self._exception
    404     finally:
    405         # Break a reference cycle with the exception in self._exception
    406         self = None

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/tenacity/__init__.py:
 ↪478, in Retrying.__call__(self, fn, *args, **kwargs)
    476 if isinstance(do, DoAttempt):
    477     try:
--> 478         result = fn(*args, **kwargs)
    479     except BaseException:  # noqa: B902
    480         retry_state.set_exception(sys.exc_info())  # type:␣
 ↪ignore[arg-type]

Cell In[4], line 31, in call_ollama_with_retry(client, prompt)
```

```
    28 @retry(stop=stop_after_attempt(3), wait=wait_exponential(min=2, max=10))
    29 def call_ollama_with_retry(client, prompt):
    30     logger.info(f"Calling Ollama model '{OLLAMA_MODEL}'…")
---> 31     response = client.chat(
    32         model=OLLAMA_MODEL,
    33         messages=[{"role": "user", "content": prompt}],
    34         format="json",
    35         options={"temperature": 0.0}
    36     )
    37     if not response or 'message' not in response or not␣
 ↪response['message'].get('content'):
    38         raise ValueError("Ollama returned an empty or invalid response. )

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/ollama/_client.py:
 ↪342, in Client.chat(self, model, messages, tools, stream, think, format,␣
 ↪options, keep_alive)
   297 def chat(
   298   self,
   299   model: str = '',
  (…)
   307   keep_alive: Optional[Union[float, str]] = None,
   308 ) -> Union[ChatResponse, Iterator[ChatResponse]]:
   309   """
   310   Create a chat response using the requested model.
   311
  (…)
   340   Returns `ChatResponse` if `stream` is `False`, otherwise returns a␣
 ↪`ChatResponse` generator.
   341   """
--> 342   return self._request(
   343     ChatResponse,
   344     'POST',
   345     '/api/chat',
   346     json=ChatRequest(
   347       model=model,
   348       messages=list(_copy_messages(messages)),
   349       tools=list(_copy_tools(tools)),
   350       stream=stream,
   351       think=think,
   352       format=format,
   353       options=options,
   354       keep_alive=keep_alive,
   355     ).model_dump(exclude_none=True),
   356     stream=stream,
   357   )

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/ollama/_client.py:
 ↪180, in Client._request(self, cls, stream, *args, **kwargs)
```

```
    176         yield cls(**part)
    178   return inner()
--> 180 return cls(**self._request_raw(*args, **kwargs).json())

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/ollama/_client.py:
 ↪120, in Client._request_raw(self, *args, **kwargs)
    118 def _request_raw(self, *args, **kwargs):
    119   try:
--> 120     r = self._client.request(*args, **kwargs)
    121     r.raise_for_status()
    122     return r

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpx/_client.py:
 ↪827, in Client.request(self, method, url, content, data, files, json, params,␣
 ↪headers, cookies, auth, follow_redirects, timeout, extensions)
    812     warnings.warn(message, DeprecationWarning)
    814 request = self.build_request(
    815     method=method,
    816     url=url,
   (…)
    825     extensions=extensions,
    826 )
--> 827 return self.send(request, auth=auth, follow_redirects=follow_redirects)

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpx/_client.py:
 ↪914, in Client.send(self, request, stream, auth, follow_redirects)
    906 follow_redirects = (
    907     self.follow_redirects
    908     if isinstance(follow_redirects, UseClientDefault)
    909     else follow_redirects
    910 )
    912 auth = self._build_request_auth(request, auth)
--> 914 response = self._send_handling_auth(
    915     request,
    916     auth=auth,
    917     follow_redirects=follow_redirects,
    918     history=[],
    919 )
    920 try:
    921     if not stream:

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpx/_client.py:
 ↪942, in Client._send_handling_auth(self, request, auth, follow_redirects,␣
 ↪history)
    939 request = next(auth_flow)
    941 while True:
--> 942     response = self._send_handling_redirects(
    943         request,
```

```
    944         follow_redirects=follow_redirects,
    945         history=history,
    946     )
    947     try:
    948         try:

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpx/_client.py:
 ↪979, in Client._send_handling_redirects(self, request, follow_redirects,␣
 ↪history)
    976 for hook in self._event_hooks["request"]:
    977     hook(request)
--> 979 response = self._send_single_request(request)
    980 try:
    981     for hook in self._event_hooks["response"]:

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpx/_client.py:
 ↪1015, in Client._send_single_request(self, request)
   1010     raise RuntimeError(
   1011         "Attempted to send an async request with a sync Client instance "
   1012     )
   1014 with request_context(request=request):
-> 1015     response = transport.handle_request(request)
   1017 assert isinstance(response.stream, SyncByteStream)
   1019 response.request = request

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpx/_transports/
 ↪default.py:233, in HTTPTransport.handle_request(self, request)
    220 req = httpcore.Request(
    221     method=request.method,
    222     url=httpcore.URL(
   (…)
    230     extensions=request.extensions,
    231 )
    232 with map_httpcore_exceptions():
--> 233     resp = self._pool.handle_request(req)
    235 assert isinstance(resp.stream, typing.Iterable)
    237 return Response(
    238     status_code=resp.status,
    239     headers=resp.headers,
    240     stream=ResponseStream(resp.stream),
    241     extensions=resp.extensions,
    242 )

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪connection_pool.py:256, in ConnectionPool.handle_request(self, request)
    253         closing = self._assign_requests_to_connections()
    255     self._close_connections(closing)
--> 256     raise exc from None
```

```
    258 # Return the response. Note that in this case we still have to manage
    259 # the point at which the response is closed.
    260 assert isinstance(response.stream, typing.Iterable)

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪connection_pool.py:236, in ConnectionPool.handle_request(self, request)
    232 connection = pool_request.wait_for_connection(timeout=timeout)
    234 try:
    235     # Send the request on the assigned connection.
--> 236     response = connection.handle_request(
    237         pool_request.request
    238     )
    239 except ConnectionNotAvailable:
    240     # In some cases a connection may initially be available to
    241     # handle a request, but then become unavailable.
    242     #
    243     # In this case we clear the connection and try again.
    244     pool_request.clear_connection()

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪connection.py:103, in HTTPConnection.handle_request(self, request)
    100     self._connect_failed = True
    101     raise exc
--> 103 return self._connection.handle_request(request)

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪http11.py:136, in HTTP11Connection.handle_request(self, request)
    134     with Trace("response_closed", logger, request) as trace:
    135         self._response_closed()
--> 136 raise exc

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪http11.py:106, in HTTP11Connection.handle_request(self, request)
     95     pass
     97 with Trace(
     98     "receive_response_headers", logger, request, kwargs
     99 ) as trace:
    100     (
    101         http_version,
    102         status,
    103         reason_phrase,
    104         headers,
    105         trailing_data,
--> 106     ) = self._receive_response_headers(**kwargs)
    107     trace.return_value = (
    108         http_version,
    109         status,
    110         reason_phrase,
```

```
    111         headers,
    112     )
    114 network_stream = self._network_stream

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪http11.py:177, in HTTP11Connection._receive_response_headers(self, request)
    174 timeout = timeouts.get("read", None)
    176 while True:
--> 177     event = self._receive_event(timeout=timeout)
    178     if isinstance(event, h11.Response):
    179         break

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_sync/
 ↪http11.py:217, in HTTP11Connection._receive_event(self, timeout)
    214     event = self._h11_state.next_event()
    216 if event is h11.NEED_DATA:
--> 217     data = self._network_stream.read(
    218         self.READ_NUM_BYTES, timeout=timeout
    219     )
    221     # If we feed this case through h11 we'll raise an exception like:
    222     #
    223     #     httpcore.RemoteProtocolError: can't handle event type
   (…)
    227     # perspective. Instead we handle this case distinctly and treat
    228     # it as a ConnectError.
    229     if data == b"" and self._h11_state.their_state == h11.SEND_RESPONSE

File ~/miniconda3/envs/tensor2/lib/python3.10/site-packages/httpcore/_backends/
 ↪sync.py:128, in SyncStream.read(self, max_bytes, timeout)
    126 with map_exceptions(exc_map):
    127     self._sock.settimeout(timeout)
--> 128     return self._sock.recv(max_bytes)

KeyboardInterrupt:
```

4 Extract TOC from epub or epub

[4]:
```
# 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()
    content_tag = navpoint.content
    href = content_tag['src'] if content_tag else None

    # Add filtering logic here if needed
    node = {"level": level, "title": title, "href": href, "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)
        href = a_tag.get('href', None)
        # Add filtering logic here if needed
        node = {"level": level, "title": title, "href": href, "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, level=0)
                    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, level=0)
                        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):
    """
    Builds a hierarchical structure from a flat ToC list from PyMuPDF.
    MODIFIED: Normalizes levels to start at 0 for consistency with EPUB.
    """
    root = []
    # The parent_stack keys are now level-based, starting from -1 for the␣
 ↪root's parent.
    parent_stack = {-1: {"children": root}}

    for level, title, page in toc_list:
        # --- FIX: NORMALIZE LEVEL TO START AT 0 ---
        # fitz/PyMuPDF ToC levels start at 1, so we subtract 1.
        normalized_level = level - 1

        node = {
            "level": normalized_level,
            "title": title.strip(),
            "page": page,
            "children": []
        }

        # Find the correct parent in the stack. The parent's level is one less␣
 ↪than the current node's.
        # This logic correctly places the node under its parent in the␣
 ↪hierarchy.
        parent_node = parent_stack[normalized_level - 1]
        parent_node["children"].append(node)

        # Add the current node to the stack so it can be a parent for␣
 ↪subsequent nodes.
```

```
        parent_stack[normalized_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 EPUB ToC for:
/home/sebas_dev_linux/projects/course_generator/data/books/Bill Nelson, Amelia
Phillips, Christopher Steuart - Guide to Computer Forensics and Investigations_
Processing Digital Evidence-Cengage Learning (2018).epub
INFO: Found EPUB 2 (NCX) Table of Contents.
  Successfully wrote EPUB ToC to: /home/sebas_dev_linux/projects/course_generato
r/Parse_data/Parse_TOC_books/ICT312_epub_table_of_contents.json
```

5 Hirachical DB base on TOC

5.1 Process Book

[5]:
```
# Cell 5: Create Hierarchical Vector Database (Definitive Final Version)

import os
import json
import shutil
```

```
import logging
import re
from typing import List, Dict, Any, Tuple
from langchain_core.documents import Document
from langchain_community.document_loaders import UnstructuredEPubLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_ollama.embeddings import OllamaEmbeddings
from langchain_chroma import Chroma

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# --- HELPER FUNCTIONS ---
def normalize_text(text: str) -> str:
    """Converts text to a canonical form for matching: lowercase, no punctuation, single spaces."""
    if not text: return ""
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

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 PROCESSING FUNCTION ---
# In Cell 5, replace the entire function with this one.

from rapidfuzz import process, fuzz

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)}'...")
    try:
        with open(extracted_toc_json_path, 'r', encoding='utf-8') as f:
            hierarchical_toc = json.load(f)
    except Exception as e:
        logger.error(f"FATAL: Error loading ToC JSON: {e}"); return ([], [])

    # --- [PATCH A - PART 1] ---
```

```
    # Build the href -> nav_path mapping from the full ToC
    href_to_navpath = {}
    def _walk(nodes, trail):
        for n in nodes:
            # Add the current node's title to the trail for its children
            current_trail = trail + [n.get("title", "")]
            href = n.get("href")
            if href:
                # Store the full trail using the href as the key
                href_to_navpath[href.split("#")[0]] = current_trail

            if n.get("children"):
                _walk(n["children"], current_trail)

    _walk(hierarchical_toc, [])
    # --- [END PATCH A - PART 1] ---

    loader = UnstructuredEPubLoader(book_path, mode="elements", strategy="fast")
    all_raw_book_docs = loader.load()

    logger.info("Enriching documents with robust hierarchical metadata...")
    # (The following section combines the original heading-based enrichment␣
↪with the new full_path enrichment)
    flat_toc_entries = []
    def _flatten_toc_recursive(nodes: List[Dict[str, Any]], path: List[str]):
        for node in nodes:
            if title := node.get("title", "").strip():
                new_path = path + [title]
                flat_toc_entries.append({"full_title_for_matching": title,␣
↪"titles_path": new_path})
                if node.get("children"):
                    _flatten_toc_recursive(node["children"], new_path)
    _flatten_toc_recursive(hierarchical_toc, [])

    toc_title_set = {normalize_text(entry["full_title_for_matching"]) for entry␣
↪in flat_toc_entries}
    normalized_title_to_path_map =␣
↪{normalize_text(entry["full_title_for_matching"]): entry["titles_path"] for␣
↪entry in flat_toc_entries}

    final_documents_with_metadata: List[Document] = []
    current_hierarchy = {}
    for doc in all_raw_book_docs:
        normalized_text = normalize_text(doc.page_content)
        if not normalized_text: continue
```

```
        match, score, _ = process.extractOne(normalized_text, toc_title_set,␣
↪scorer=fuzz.token_set_ratio)
        if score > 95:
            current_hierarchy = {}
            path = normalized_title_to_path_map[match]
            for i, title in enumerate(path):
                current_hierarchy[f"level_{i}_title"] = title

        if not current_hierarchy: continue

        new_metadata = doc.metadata.copy()
        new_metadata.update(current_hierarchy)
        final_documents_with_metadata.append(Document(page_content=doc.
↪page_content, metadata=new_metadata))

    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size,␣
↪chunk_overlap=chunk_overlap)
    final_chunks = text_splitter.split_documents(final_documents_with_metadata)

    logger.info(f"Split documents into {len(final_chunks)} final chunks.")
    logger.info("Assigning sequence IDs and canonical paths...")

    for i, chunk in enumerate(final_chunks):
        chunk.metadata['global_chunk_sequence_id'] = i
        path_parts = [chunk.metadata.get(f"level_{j}_title", "") for j in␣
↪range(6)]
        raw_path = " > ".join(filter(None, path_parts))
        chunk.metadata['toc_path'] = raw_path
        chunk.metadata['toc_path_norm'] = normalize_text(raw_path)

        # --- [PATCH A - PART 2] ---
        # Attach the canonical navMap path to every chunk.
        source_file = chunk.metadata.get("source", "").split("#")[0]
        nav_path = href_to_navpath.get(source_file, [])
        full_nav = " > ".join(filter(None, nav_path))

        chunk.metadata["toc_path_full"] = full_nav
        chunk.metadata["toc_path_full_norm"] = normalize_text(full_nav)
        # --- [END PATCH A - PART 2] ---

    return final_chunks, hierarchical_toc

# --- Main Execution Block ---
if not os.path.exists(PRE_EXTRACTED_TOC_JSON_PATH):
    logger.error("CRITICAL: Pre-extracted ToC file not found. Run Cell 4 first.
↪")
else:
```

```
    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=500, chunk_overlap=50
    )
    if final_chunks_for_db:
        logger.info("Sanitizing all chunk metadata for ChromaDB compatibility...
↪")
        for chunk in final_chunks_for_db:
            chunk.metadata = {k: clean_metadata_for_chroma(v) for k, v in chunk.
↪metadata.items()}

        if os.path.exists(CHROMA_PERSIST_DIR):
            logger.warning(f"Deleting existing ChromaDB directory:␣
↪{CHROMA_PERSIST_DIR}")
            shutil.rmtree(CHROMA_PERSIST_DIR)

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

```
2025-06-26 05:10:12,833 - INFO - Processing book 'Bill Nelson, Amelia Phillips,
Christopher Steuart - Guide to Computer Forensics and Investigations_ Processing
Digital Evidence-Cengage Learning (2018).epub'…
2025-06-26 05:10:14,768 - INFO - Note: NumExpr detected 32 cores but
"NUMEXPR_MAX_THREADS" not set, so enforcing safe limit of 16.
2025-06-26 05:10:14,768 - INFO - NumExpr defaulting to 16 threads.
[WARNING] Could not load translations for en-US
  data file translations/en.yaml not found
2025-06-26 05:10:21,588 - WARNING - Could not load translations for en-US
  data file translations/en.yaml not found
[WARNING] The term Abstract has no translation defined.

2025-06-26 05:10:21,589 - WARNING - The term Abstract has no translation
defined.

2025-06-26 05:10:24,902 - INFO - Enriching documents with robust hierarchical
```

```
metadata…
2025-06-26 05:10:29,337 - INFO - Split documents into 12498 final chunks.
2025-06-26 05:10:29,337 - INFO - Assigning sequence IDs and canonical paths…
2025-06-26 05:10:29,415 - INFO - Sanitizing all chunk metadata for ChromaDB
compatibility…
2025-06-26 05:10:29,447 - INFO - Initializing embedding model 'nomic-embed-text'
and creating new vector database…
2025-06-26 05:10:29,497 - INFO - Anonymized telemetry enabled. See
https://docs.trychroma.com/telemetry for more information.
2025-06-26 05:11:40,679 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-26 05:12:54,167 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-26 05:13:19,488 - INFO - HTTP Request: POST
http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
2025-06-26 05:13:20,617 - INFO -   Vector DB created successfully. Collection
contains 12498 documents.

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

5.1.1 Smoke test

[6]:
```
# # Cell 5.1: Smoke Test & Sanity Check

# print("Smoke Test & Sanity Check")

# if 'final_chunks_for_db' in locals() and final_chunks_for_db:
#     logger.info("  `final_chunks_for_db` object exists. Running sanity checks.
 ↪..")

#     # 1. Peek at one sample chunk's metadata
#     print("\n PEEKING at metadata of the first chunk:")
#     sample_metadata = final_chunks_for_db[0].metadata
#     print(json.dumps(sample_metadata, indent=2))
#     assert "toc_path" in sample_metadata, "FATAL: 'toc_path' field is missing!
 ↪"
#     assert "toc_path_norm" in sample_metadata, "FATAL: 'toc_path_norm' field␣
 ↪is missing!"
#     logger.info(" Sample chunk contains the required 'toc_path' and␣
 ↪'toc_path_norm' fields.")

#     # 2. Confirm every chunk has the new field
#     all_chunks_have_norm_path = all("toc_path_norm" in c.metadata for c in␣
 ↪final_chunks_for_db)
#     assert all_chunks_have_norm_path, "FATAL: Not all chunks have the␣
 ↪'toc_path_norm' metadata field!"
```

```
#     logger.info(f" Verified that all {len(final_chunks_for_db)} chunks have␣
 ↪the 'toc_path_norm' field.")

#     # 3. Sanity-count distinct paths
#     unique_paths = {c.metadata["toc_path_norm"] for c in final_chunks_for_db␣
 ↪if "toc_path_norm" in c.metadata}
#     logger.info(f" Found {len(unique_paths)} unique normalized paths in the␣
 ↪dataset.")

#     print("\n" + "*"*80)
#     print("SMOKE TEST PASSED. The data is correctly structured. You can now␣
 ↪proceed to build the database.".center(80))
#     print("*"*80)

# else:
#     logger.error(" `final_chunks_for_db` not found or is empty. Run the main␣
 ↪part of Cell 5 first.")
```

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.

[7]:
```
# # Cell 6: Verify Vector Database (Definitive Final Suite with Diagnostics)

# import os
# import json
# import re
# import random
# import logging
# from typing import List, Dict, Any, Tuple, Optional
# import pandas as pd

# try:
#     from langchain_chroma import Chroma
#     from langchain_core.documents import Document
#     from langchain_ollama.embeddings import OllamaEmbeddings
#     langchain_available = True
# except ImportError:
#     langchain_available = False

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

# # --- HELPER FUNCTIONS ---
# def normalize_text(text: str) -> str:
#     """Converts text to a canonical form for matching: lowercase, no␣
 ↪punctuation, single spaces."""
#     if not text: return ""
#     text = text.lower()
#     text = re.sub(r'[^\w\s]', '', text)
#     text = re.sub(r'\s+', ' ', text).strip()
#     return text

# def print_header(text: str, char: str = "="):
#     """Prints a centered header to the console."""
#     print("\n" + char * 80)
#     print(text.center(80))
#     print(char * 80)

# def print_results(operation_name: str, results: list, where_filter:␣
 ↪Optional[Dict] = None):
#     """Prints the results of a vector store operation in a clear, readable␣
 ↪format."""
#     print("\n" + "-"*50)
#     print(f" Operation: '{operation_name}'")
#     if where_filter:
#         print(f" Filter: {json.dumps(where_filter, indent=2)}")
```

```
#     if not results:
#         print("\n RESULTS: No documents found for this operation.")
#         print("-" * 50)
#         return

#     print(f"\n RESULTS: Found {len(results)} documents. Displaying details␣
 ↪for top 3:")
#     for i, doc in enumerate(results[:3]):
#         print(f"\n--- Result {i+1} ---")
#         print(f"Content: '{doc.page_content.replace('', ' ').strip()[:150]}...
 ↪'")
#         print(f"Metadata: {json.dumps(doc.metadata, indent=2)}")
#     print("-" * 50)

# def find_leaf_section(nodes: List[Dict]) -> Optional[List[str]]:
#     """Finds a random, deep, leaf-node section from the ToC.json."""
#     leaf_paths = []
#     def _traverse(sub_nodes, current_path):
#         for node in sub_nodes:
#             new_path = current_path + [node.get('title', 'Untitled')]
#             if not node.get('children'):
#                 if len(new_path) > 2: # Ensure it's at least level 2 deep for␣
 ↪a meaningful test
#                     leaf_paths.append(new_path)
#             else:
#                 _traverse(node['children'], new_path)
#     _traverse(nodes, [])
#     return random.choice(leaf_paths) if leaf_paths else None

# # --- TEST CASE FUNCTIONS ---
# def run_test(name: str, goal: str, func, *args):
#     """A wrapper to run each test and print its final status."""
#     print_header(name, char="-")
#     logger.info(f" GOAL: {goal}")
#     status = " FAILED"
#     try:
#         if func(*args):
#             status = " PASSED"
#             return True
#         return False
#     except Exception as e:
#         logger.error(f"ERROR: {e}", exc_info=False)
#         return False
#     finally:
#         print(f"\n--> {name} Status: {status}")

# def _health_and_hierarchy_report(db):
```

```
#     """Provides a high-level diagnostic overview of the database's structure.
 ↪"""
#     print_header("Database Health & Hierarchy Report", char="*")
#     total_docs = db._collection.count()
#     logger.info(f"Retrieving metadata for all {total_docs} chunks...")
#     retrieved_data = db.get(limit=total_docs, include=["metadatas"])
#     all_metadatas = retrieved_data['metadatas']

#     level_0_counts = {}
#     for meta in all_metadatas:
#         level_0_title = meta.get("level_0_title")
#         if level_0_title:
#             level_0_counts[level_0_title] = level_0_counts.get(level_0_title,␣
 ↪0) + 1

#     assert level_0_counts, "CRITICAL: No 'level_0_title' metadata found in␣
 ↪any chunks!"

#     print("\n Found the following top-level sections and their chunk counts:
 ↪")
#     df = pd.DataFrame(list(level_0_counts.items()), columns=['Top-Level␣
 ↪Section (level_0_title)', 'Chunk Count'])
#     df = df.sort_values(by='Chunk Count', ascending=False).
 ↪reset_index(drop=True)
#     print(df.to_string())
#     return True

# def _deep_hierarchy_test(db, toc):
#     """Verifies a deep leaf section can be retrieved via a similarity search␣
 ↪with a strict filter."""
#     path = find_leaf_section(toc)
#     assert path, "Could not find a leaf section to test."
#     section_title = path[-1]

#     # Use the normalized path for the filter, as created in Cell 5
#     full_path_norm = normalize_text(' > '.join(path))
#     w_filter = {"toc_path_norm": {"$eq": full_path_norm}}

#     # Use similarity search on the raw title for a realistic test
#     results = db.similarity_search(section_title, k=1, filter=w_filter)
#     print_results(f"Deep hierarchy check for: '{section_title}'", results,␣
 ↪w_filter)
#     assert len(results) > 0, "Deeply filtered similarity search returned no␣
 ↪results."
#     return True
```

```
# def _narrative_flow_test(db, toc):
#     """Verifies chunks for a specific leaf section are sequentially ordered.
 ↪"""
#     path = find_leaf_section(toc)
#     assert path, "Could not find a leaf section in ToC to test."

#     full_path_norm = normalize_text(' > '.join(path))
#     w_filter = {"toc_path_norm": {"$eq": full_path_norm}}

#     operation_name = f"Narrative flow check for: {' > '.join(path)}"
#     print_results(operation_name, [], w_filter) # Announce the operation

#     retrieved_data = db.get(where=w_filter, limit=200)

#     docs = []
#     if retrieved_data and retrieved_data['ids']:
#         docs = [Document(page_content=retrieved_data['documents'][i],␣
 ↪metadata=retrieved_data['metadatas'][i]) for i in␣
 ↪range(len(retrieved_data['ids']))]

#     print_results(f"Results for '{path[-1]}'", docs) # Show what was found
#     assert len(docs) > 1, "Not enough chunks retrieved to test sequence."

#     docs.sort(key=lambda x: x.metadata.get('global_chunk_sequence_id', -1))
#     sequence_numbers = [doc.metadata['global_chunk_sequence_id'] for doc in␣
 ↪docs]

#     print(f"\nANALYSIS: Retrieved and sorted global sequence numbers:␣
 ↪{sequence_numbers}")
#     assert all(sequence_numbers[i] < sequence_numbers[i+1] for i in␣
 ↪range(len(sequence_numbers)-1)), "Global sequence is not strictly increasing.
 ↪"
#     return True

# # --- MAIN VERIFICATION EXECUTION ---
# def run_verification():
#     print_header("Database Verification Process")
#     if not langchain_available: logger.error("LangChain libraries not found.
 ↪"); return
#     required_paths = [CHROMA_PERSIST_DIR, PRE_EXTRACTED_TOC_JSON_PATH,␣
 ↪PARSED_UO_JSON_PATH]
#     if not all(os.path.exists(p) for p in required_paths): logger.
 ↪error("Missing file/dir. Run Cells 4 & 5."); return
#     with open(PRE_EXTRACTED_TOC_JSON_PATH, 'r', encoding='utf-8') as f:␣
 ↪toc_data = json.load(f)
```

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

#     tests = [
#         ("Diagnostic: Health & Hierarchy Report", "Provides a high-level␣
 ↪overview of the DB's structure.", _health_and_hierarchy_report,␣
 ↪(vector_store,)),
#         ("Test 1: Deep Hierarchy & Filter", "Checks if a deep section can be␣
 ↪retrieved with a strict filter.", _deep_hierarchy_test, (vector_store,␣
 ↪toc_data)),
#         ("Test 2: Narrative Flow", "Checks if chunks within a specific leaf␣
 ↪sub-section are correctly ordered.", _narrative_flow_test, (vector_store,␣
 ↪toc_data))
#     ]
#     results_summary = [run_test(name, goal, func, *args) for name, goal,␣
 ↪func, args in tests]

#     passed_count = sum(filter(None, results_summary))
#     failed_count = len(results_summary) - passed_count
#     print_header("Final Verification Summary")
#     print(f"Total Tests Run: {len(results_summary)} |  Passed:␣
 ↪{passed_count} |  Failed: {failed_count}")
#     print_header("Verification Complete", char="=")

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

[11]:
```
# Cell 6: Verify Vector Database (Definitive Final Version)

import os
import json
import re
import random
import logging
from typing import List, Dict, Any, Tuple, Optional
import pandas as pd

try:
    from langchain_chroma import Chroma
    from langchain_core.documents import Document
    from langchain_ollama.embeddings import OllamaEmbeddings
    langchain_available = True
except ImportError:
```

```
    langchain_available = False

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# --- HELPER FUNCTIONS ---
def normalize_text(text: str) -> str:
    """Converts text to a canonical form for matching: lowercase, no punctuation, single spaces."""
    if not text: return ""
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def print_header(text: str, char: str = "="):
    """Prints a centered header to the console."""
    print("\n" + char * 80)
    print(text.center(80))
    print(char * 80)

def print_results(operation_name: str, results: list, where_filter: Optional[Dict] = None):
    """Prints the results of a vector store operation in a clear, readable format."""
    print("\n" + "-"*50)
    print(f" Operation: '{operation_name}'")
    if where_filter:
        print(f" Filter: {json.dumps(where_filter, indent=2)}")

    if not results:
        print("\n RESULTS: No documents found for this operation.")
        print("-" * 50)
        return

    print(f"\n RESULTS: Found {len(results)} documents. Displaying details for top 3:")
    for i, doc in enumerate(results[:3]):
        print(f"\n--- Result {i+1} ---")
        print(f"Content: '{doc.page_content.replace('', ' ').strip()[:150]}...'")
        print(f"Metadata: {json.dumps(doc.metadata, indent=2)}")
    print("-" * 50)

def find_leaf_section(nodes: List[Dict]) -> Optional[List[str]]:
```

```
    """Finds a random, deep, leaf-node section from the ToC.json."""
    leaf_paths = []
    def _traverse(sub_nodes, current_path):
        for node in sub_nodes:
            new_path = current_path + [node.get('title', 'Untitled')]
            if not node.get('children'):
                if len(new_path) > 2:
                    leaf_paths.append(new_path)
            else:
                _traverse(node['children'], new_path)
    _traverse(nodes, [])
    return random.choice(leaf_paths) if leaf_paths else None

# --- TEST CASE FUNCTIONS ---
def run_test(name: str, goal: str, func, *args):
    """A wrapper to run each test and print its final status."""
    print_header(name, char="-")
    logger.info(f"  GOAL: {goal}")
    status = "  FAILED"
    try:
        if func(*args):
            status = "  PASSED"
            return True
        return False
    except Exception as e:
        logger.error(f"ERROR: {e}", exc_info=False)
        return False
    finally:
        print(f"\n--> {name} Status: {status}")

def _health_and_hierarchy_report(db):
    """Provides a high-level diagnostic overview of the database's structure."""
    total_docs = db._collection.count()
    logger.info(f"Retrieving metadata for all {total_docs} chunks...")
    all_metadatas = db.get(limit=total_docs, include=["metadatas"])['metadatas']

    level_0_counts = {}
    for meta in all_metadatas:
        level_0_title = meta.get("level_0_title")
        if level_0_title:
            level_0_counts[level_0_title] = level_0_counts.get(level_0_title,␣
 ↪0) + 1

    assert level_0_counts, "CRITICAL: No 'level_0_title' metadata found!"

    print("\n  Found the following top-level sections and their chunk counts:")
```

```
    df = pd.DataFrame(list(level_0_counts.items()), columns=['Top-Level␣
↪Section', 'Chunk Count'])
    print(df.sort_values(by='Chunk Count', ascending=False).
↪reset_index(drop=True).to_string())
    return True

def _deep_hierarchy_and_flow_test(db, toc):
    """The definitive test. It verifies both hierarchy and sequence."""
    path = find_leaf_section(toc)
    assert path, "Could not find a suitable leaf section in ToC to test."

    # --- [PATCH B] ---
    # Use a tolerant '$contains' filter on the new authoritative metadata field.
    # This finds the correct chunks even if wrapper levels are missing from␣
↪headings.
    assert len(path) >= 2, f"Found path '{path}' is not deep enough for this␣
↪test."
    leaf_norm   = normalize_text(path[-1])
    parent_norm = normalize_text(path[-2])

    w_filter = {
        "$and": [
            {"toc_path_full_norm": {"$contains": parent_norm}},
            {"toc_path_full_norm": {"$contains": leaf_norm}}
        ]
    }
    # --- [END OF PATCH B] ---

    operation_name = f"Deep Hierarchy & Narrative Flow check for: {' > '.
↪join(path)}"
    print_results(operation_name, [], w_filter) # Announce the operation and␣
↪show the filter

    retrieved_data = db.get(where=w_filter, limit=200)

    docs = []
    if retrieved_data and retrieved_data['ids']:
        docs = [Document(page_content=retrieved_data['documents'][i],
                         metadata=retrieved_data['metadatas'][i]) for i in
                range(len(retrieved_data['ids']))]

    print_results(f"Results for '{path[-1]}'", docs)
    assert len(docs) > 0, "Deep hierarchy search with '$contains' returned no␣
↪results."

    if len(docs) > 1:
```

```
        docs.sort(key=lambda x: x.metadata.get('global_chunk_sequence_id', -1))
        sequence_numbers = [doc.metadata['global_chunk_sequence_id'] for doc in␣
↪docs]

        print(f"\nANALYSIS: Retrieved and sorted global sequence numbers:
↪\n{sequence_numbers}")
        assert all(sequence_numbers[i] < sequence_numbers[i+1] for i in
                   range(len(sequence_numbers)-1)), "Global sequence is not␣
↪strictly increasing."
    else:
        logger.warning("Only one chunk was retrieved; cannot test narrative␣
↪flow sequence.")

    return True

# --- MAIN VERIFICATION EXECUTION ---
def run_verification():
    print_header("Database Verification Process")
    if not langchain_available: logger.error("LangChain libraries not found.");␣
↪return
    required_paths = [CHROMA_PERSIST_DIR, PRE_EXTRACTED_TOC_JSON_PATH]
    if not all(os.path.exists(p) for p in required_paths): logger.
↪error("Missing file/dir. Run Cells 4 & 5."); return
    with open(PRE_EXTRACTED_TOC_JSON_PATH, 'r', encoding='utf-8') as f:␣
↪toc_data = json.load(f)

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

    tests = [
        ("Diagnostic: Health & Hierarchy Report", "Provides a high-level␣
↪overview of the DB's structure.", _health_and_hierarchy_report,␣
↪(vector_store,)),
        ("Test: Deep Hierarchy & Narrative Flow", "Checks if chunks within a␣
↪leaf section are found and correctly ordered.",␣
↪_deep_hierarchy_and_flow_test, (vector_store, toc_data))
    ]
    results_summary = [run_test(name, goal, func, *args) for name, goal, func,␣
↪args in tests]

    passed_count = sum(filter(None, results_summary))
    failed_count = len(results_summary) - passed_count
    print_header("Final Verification Summary")
```

```
    print(f"Total Tests Run: {len(results_summary)} |  Passed: {passed_count}␣
 ↪|  Failed: {failed_count}")
    print_header("Verification Complete", char="=")

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

```
2025-06-26 05:14:53,958 - INFO - Initializing embedding model 'nomic-embed-text'
to connect to DB…
2025-06-26 05:14:53,969 - INFO -  GOAL: Provides a high-level overview of the
DB's structure.
2025-06-26 05:14:53,971 - INFO - Retrieving metadata for all 12498 chunks…

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

--------------------------------------------------------------------------------
                      Diagnostic: Health & Hierarchy Report
--------------------------------------------------------------------------------

2025-06-26 05:14:54,793 - INFO -  GOAL: Checks if chunks within a leaf section
are found and correctly ordered.
2025-06-26 05:14:54,793 - ERROR - ERROR: Expected where operator to be one of
$gt, $gte, $lt, $lte, $ne, $eq, $in, $nin, got $contains in get.


  Found the following top-level sections and their chunk counts:
                                                                    Top-Level
Section  Chunk Count
0                     Lab Manual for Guide to Computer Forensics and
Investigations         5444
1                                       Chapter 6. Current Digital Forensics
Tools         1440
2                                    Chapter 5. Working with Windows and CLI
Systems         1310
3   Chapter 10. Virtual Machine Forensics, Live Acquisitions, and Network
Forensics          699
4                                        Chapter 16. Ethics for the Expert
Witness          593
5                                                  Chapter 13. Cloud
Forensics          575
6      Chapter 1. Understanding the Digital Forensics Profession and
Investigations          365
7                              Chapter 2. The Investigator’s Office and
Laboratory          300
8                           Chapter 15. Expert Testimony in Digital
Investigations          286
```

--> Diagnostic: Health & Hierarchy Report Status: PASSED

Test: Deep Hierarchy & Narrative Flow

Operation: 'Deep Hierarchy & Narrative Flow check for: Chapter 2. The Investigator’s Office and Laboratory > Building a Business Case for Developing a Forensics Lab > Preparing a Business Case for a Digital Forensics Lab > Approval and Acquisition'

```
 Filter: {
  "$and": [
    {
      "toc_path_full_norm": {
        "$contains": "preparing a business case for a digital forensics lab"
      }
    },
    {
      "toc_path_full_norm": {
        "$contains": "approval and acquisition"
      }
    }
  ]
}

 RESULTS: No documents found for this operation.
------------------------------------------------

--> Test: Deep Hierarchy & Narrative Flow Status:   FAILED

================================================================================
                          Final Verification Summary
================================================================================
Total Tests Run: 2 |   Passed: 1 |   Failed: 1

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

6 Full Database Health & Hierarchy Diagnostic Report

[9]:
```python
# Cell 6: Full Database Health & Hierarchy Diagnostic Report

import os
import json
import logging
from typing import List, Dict, Any, Optional
import pandas as pd

try:
    from langchain_chroma import Chroma
    from langchain_ollama.embeddings import OllamaEmbeddings
    langchain_available = True
except ImportError:
    langchain_available = False
```

```
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -␣
 ↪%(message)s')
logger = logging.getLogger(__name__)

# --- HELPER FUNCTIONS ---
def print_header(text: str, char: str = "="):
    """Prints a centered header to the console."""
    print("\n" + char * 80)
    print(text.center(80))
    print(char * 80)

def count_total_chunks(node: Dict) -> int:
    """Recursively counts all chunks in a node and its children."""
    total = node.get('_chunks', 0)
    for child_node in node.get('_children', {}).values():
        total += count_total_chunks(child_node)
    return total

def print_hierarchy_report(node: Dict, indent_level: int = 0):
    """Recursively prints the reconstructed hierarchy with chunk counts."""
    # Sort children by their total chunk count for a more organized report
    sorted_children = sorted(
        node.get('_children', {}).items(),
        key=lambda item: count_total_chunks(item[1]),
        reverse=True
    )

    for title, child_node in sorted_children:
        prefix = "    " * indent_level + "|__ "

        # Total chunks includes the node itself and all descendants
        total_chunks_in_branch = count_total_chunks(child_node)

        # Chunks directly assigned to this node (should be low if it has␣
 ↪children)
        direct_chunks = child_node.get('_chunks', 0)

        print(f"{prefix}{title}  (Total Chunks: {total_chunks_in_branch},␣
 ↪Direct: {direct_chunks})")

        # Recursive call for the next level
        print_hierarchy_report(child_node, indent_level + 1)

# --- MAIN DIAGNOSTIC FUNCTION ---
def run_full_diagnostics():
    print_header("Full Database Health & Hierarchy Diagnostic Report")
```

```
    # 1. Connect to the Database
    logger.info("Connecting to the vector database...")
    if not os.path.exists(CHROMA_PERSIST_DIR):
        logger.error(f"FATAL: Chroma DB directory not found at␣
↪{CHROMA_PERSIST_DIR}. Please run Cell 5 first."); return

    embeddings = OllamaEmbeddings(model=EMBEDDING_MODEL_OLLAMA)
    vector_store = Chroma(persist_directory=CHROMA_PERSIST_DIR,␣
↪embedding_function=embeddings, collection_name=CHROMA_COLLECTION_NAME)

    # 2. Retrieve ALL Metadata
    total_docs = vector_store._collection.count()
    logger.info(f"Retrieving metadata for all {total_docs} chunks...")
    retrieved_data = vector_store.get(limit=total_docs, include=["metadatas"])
    all_metadatas = retrieved_data['metadatas']
    logger.info("Successfully retrieved all metadata.")

    # 3. Reconstruct the Hierarchy Tree from Metadata
    logger.info("Reconstructing hierarchy from chunk metadata...")
    hierarchy_tree = {'_children': {}} # A root node to hold everything
    chunks_without_path = 0

    for meta in all_metadatas:
        path = meta.get("toc_path")
        if not path:
            chunks_without_path += 1
            continue

        path_parts = path.split(" > ")
        current_node = hierarchy_tree

        for part in path_parts:
            # Navigate or create the path in our tree
            current_node = current_node['_children'].setdefault(part,␣
↪{'_chunks': 0, '_children': {}})

        # Increment the count for the leaf node
        current_node['_chunks'] += 1

    logger.info("Hierarchy reconstruction complete.")

    # 4. Print the Report
    print_header("Reconstructed Hierarchy Report", char="-")
    print("This report shows the full hierarchical structure discovered from␣
↪the chunk metadata.")
    print("'(Total Chunks: X, Direct: Y)' means X chunks in the whole branch, Y␣
↪directly tagged to this heading.\n")
```

```
    print_hierarchy_report(hierarchy_tree)

    print_header("Diagnostic Summary", char="-")
    top_level_sections = len(hierarchy_tree.get('_children', {}))
    print(f"  Found {top_level_sections} distinct top-level sections.")
    if chunks_without_path > 0:
        logger.warning(f"  Found {chunks_without_path} chunks MISSING the 'toc_path' metadata. This indicates an error in the enrichment pipeline.")
    else:
        print("  All chunks contain 'toc_path' metadata.")

    print_header("Diagnostic Complete")

# --- Execute Diagnostics ---
run_full_diagnostics()
```

```
2025-06-26 05:14:02,922 - INFO - Connecting to the vector database…
2025-06-26 05:14:02,937 - INFO - Retrieving metadata for all 12498 chunks…


================================================================================
                Full Database Health & Hierarchy Diagnostic Report
================================================================================

2025-06-26 05:14:03,841 - INFO - Successfully retrieved all metadata.
2025-06-26 05:14:03,842 - INFO - Reconstructing hierarchy from chunk metadata…
2025-06-26 05:14:03,851 - INFO - Hierarchy reconstruction complete.


--------------------------------------------------------------------------------
                          Reconstructed Hierarchy Report
--------------------------------------------------------------------------------
This report shows the full hierarchical structure discovered from the chunk
metadata.
'(Total Chunks: X, Direct: Y)' means X chunks in the whole branch, Y directly
tagged to this heading.

|__ Lab Manual for Guide to Computer Forensics and Investigations  (Total
Chunks: 5444, Direct: 3)
    |__ Chapter 16. Ethics for the Expert Witness  (Total Chunks: 1698, Direct:
118)
        |__ Lab 16.1. Rebuilding an MFT Record from a Corrupt Image  (Total
Chunks: 1568, Direct: 5)
            |__ Review Questions  (Total Chunks: 788, Direct: 788)
            |__ Objectives  (Total Chunks: 425, Direct: 298)
                |__ Materials Required  (Total Chunks: 127, Direct: 127)
            |__ Activity  (Total Chunks: 350, Direct: 337)
                |__ Copying the Corrected MFT Record  (Total Chunks: 4, Direct:
```

```
4)
                |__ Creating a Duplicate Forensic Image  (Total Chunks: 4,
Direct: 4)
                |__ Determining the Offset Byte Address of the Corrupt MFT
Record  (Total Chunks: 3, Direct: 3)
                |__ Extracting Additional Evidence  (Total Chunks: 2, Direct: 2)
        |__ Chapter Introduction  (Total Chunks: 12, Direct: 12)
    |__ Chapter 1. Understanding the Digital Forensics Profession and
Investigations  (Total Chunks: 933, Direct: 469)
        |__ Lab 1.1. Installing Autopsy for Windows  (Total Chunks: 262, Direct:
262)
        |__ Lab 1.2. Downloading FTK Imager Lite  (Total Chunks: 100, Direct:
100)
        |__ Lab 1.4. Using Autopsy for Windows  (Total Chunks: 99, Direct: 99)
        |__ Lab 1.3. Downloading WinHex  (Total Chunks: 3, Direct: 3)
    |__ Chapter 3. Data Acquisition  (Total Chunks: 537, Direct: 508)
        |__ Lab 3.1. Creating a DEFT Zero Forensic Boot CD and USB Drive  (Total
Chunks: 16, Direct: 4)
            |__ Activity  (Total Chunks: 12, Direct: 0)
                |__ Learning DEFT Zero Features  (Total Chunks: 6, Direct: 6)
                |__ Creating a DEFT Zero Boot CD  (Total Chunks: 4, Direct: 4)
                |__ Creating a Bootable USB DEFT Zero Drive  (Total Chunks: 2,
Direct: 2)
        |__ Lab 3.4. Examining an HFS+ Image  (Total Chunks: 7, Direct: 7)
        |__ Lab 3.2. Examining a FAT Image  (Total Chunks: 3, Direct: 3)
        |__ Lab 3.3. Examining an NTFS Image  (Total Chunks: 3, Direct: 3)
    |__ Chapter 4. Processing Crime and Incident Scenes  (Total Chunks: 497,
Direct: 462)
        |__ Lab 4.2. Using Mini-WinFE to Boot and Image a Windows Computer
(Total Chunks: 20, Direct: 20)
        |__ Lab 4.1. Creating a Mini-WinFE Boot CD  (Total Chunks: 7, Direct: 3)
            |__ Activity  (Total Chunks: 4, Direct: 0)
                |__ Creating a Mini-WinFE ISO Image  (Total Chunks: 3, Direct:
3)
                |__ Setting Up Mini-WinFE  (Total Chunks: 1, Direct: 1)
        |__ Lab 4.4. Creating an Image with Guymager  (Total Chunks: 5, Direct:
5)
        |__ Lab 4.3. Testing the Mini-WinFE Write-Protection Feature  (Total
Chunks: 3, Direct: 3)
    |__ Chapter 5. Working with Windows and CLI Systems  (Total Chunks: 492,
Direct: 438)
        |__ Lab 5.4. Examining the ntuser.dat Registry File  (Total Chunks: 31,
Direct: 31)
        |__ Lab 5.1. Using DART to Export Windows Registry Files  (Total Chunks:
15, Direct: 15)
        |__ Lab 5.2. Examining the SAM Hive  (Total Chunks: 4, Direct: 4)
        |__ Lab 5.3. Examining the SYSTEM Hive  (Total Chunks: 4, Direct: 4)
    |__ Chapter 6. Current Digital Forensics Tools  (Total Chunks: 448, Direct:
```

```
349)
        |__ Lab 6.3. Examining a Corrupt Image File with FTK Imager Lite,
Autopsy, and WinHex  (Total Chunks: 76, Direct: 8)
            |__ Activity  (Total Chunks: 68, Direct: 0)
                |__ Testing an Image File in Autopsy 4.3.0  (Total Chunks: 62,
Direct: 62)
                |__ Examining Image Files in WinHex  (Total Chunks: 6, Direct:
6)
        |__ Lab 6.1. Using Autopsy 4.7.0 to Search an Image File  (Total Chunks:
19, Direct: 3)
            |__ Activity  (Total Chunks: 16, Direct: 0)
                |__ Installing Autopsy 4.7.0  (Total Chunks: 14, Direct: 14)
                |__ Searching E-mail in Autopsy 4.7.0  (Total Chunks: 2, Direct:
2)
        |__ Lab 6.2. Using OSForensics to Search an Image of a Hard Drive
(Total Chunks: 4, Direct: 4)
    |__ Chapter 9. Digital Forensics Analysis and Validation  (Total Chunks:
280, Direct: 231)
        |__ Lab 9.2. Validating File Hash Values with FTK Imager Lite  (Total
Chunks: 41, Direct: 41)
        |__ Lab 9.3. Validating File Hash Values with WinHex  (Total Chunks: 5,
Direct: 5)
        |__ Lab 9.1. Using Autopsy to Search for Keywords in an Image  (Total
Chunks: 3, Direct: 3)
    |__ Chapter 15. Expert Testimony in Digital Investigations  (Total Chunks:
200, Direct: 81)
        |__ Lab 15.3. Recovering a Password from Password-Protected Files
(Total Chunks: 113, Direct: 7)
            |__ Activity  (Total Chunks: 106, Direct: 0)
                |__ Verifying the Existence of a Warning Banner  (Total Chunks:
105, Direct: 105)
                |__ Recovering a Password from Password-Protected Files  (Total
Chunks: 1, Direct: 1)
        |__ Lab 15.1. Conducting a Preliminary Investigation  (Total Chunks: 3,
Direct: 3)
        |__ Lab 15.2. Investigating an Arsonist  (Total Chunks: 3, Direct: 3)
    |__ Chapter 13. Cloud Forensics  (Total Chunks: 120, Direct: 112)
        |__ Lab 13.3. Examining OneDrive Cloud Storage  (Total Chunks: 6,
Direct: 6)
        |__ Lab 13.1. Examining Dropbox Cloud Storage  (Total Chunks: 1, Direct:
1)
        |__ Lab 13.2. Examining Google Drive Cloud Storage  (Total Chunks: 1,
Direct: 1)
    |__ Chapter 2. The Investigator's Office and Laboratory  (Total Chunks: 60,
Direct: 6)
        |__ Lab 2.5. Viewing Images in FTK Imager Lite  (Total Chunks: 23,
Direct: 23)
        |__ Lab 2.1. Wiping a USB Drive Securely  (Total Chunks: 9, Direct: 9)
```

```
        |__ Lab 2.2. Using Directory Snoop to Image a USB Drive  (Total Chunks:
8, Direct: 8)
        |__ Lab 2.4. Imaging Evidence with FTK Imager Lite  (Total Chunks: 8,
Direct: 8)
        |__ Lab 2.3. Converting a Raw Image to an .E01 Image  (Total Chunks: 6,
Direct: 6)
    |__ Chapter 10. Virtual Machine Forensics, Live Acquisitions, and Network
Forensics  (Total Chunks: 35, Direct: 7)
        |__ Lab 10.3. Using Kali Linux for Network Forensics  (Total Chunks: 12,
Direct: 1)
            |__ Activity  (Total Chunks: 11, Direct: 0)
                |__ Mounting Drives in Kali Linux  (Total Chunks: 9, Direct: 9)
                |__ Identifying Open Ports and Making a Screen Capture  (Total
Chunks: 2, Direct: 2)
        |__ Lab 10.1. Analyzing a Forensic Image Hosting a Virtual Machine
(Total Chunks: 11, Direct: 3)
            |__ Activity  (Total Chunks: 8, Direct: 0)
                |__ Installing MD5 Hashes in Autopsy  (Total Chunks: 6, Direct:
6)
                |__ Analyzing a Windows Image Containing a Virtual Machine
(Total Chunks: 2, Direct: 2)
        |__ Lab 10.2. Conducting a Live Acquisition  (Total Chunks: 5, Direct:
1)
            |__ Activity  (Total Chunks: 4, Direct: 0)
                |__ Installing Tools for Live Acquisitions  (Total Chunks: 2,
Direct: 2)
                |__ Exploring Tools for Live Acquisitions  (Total Chunks: 2,
Direct: 2)
    |__ Chapter 8. Recovering Graphics Files  (Total Chunks: 33, Direct: 13)
        |__ Lab 8.3. Using WinHex to Analyze Multimedia Files  (Total Chunks:
11, Direct: 11)
        |__ Lab 8.2. Using OSForensics to Analyze Multimedia Files  (Total
Chunks: 6, Direct: 6)
        |__ Lab 8.1. Using Autopsy to Analyze Multimedia Files  (Total Chunks:
3, Direct: 3)
    |__ Chapter 12. Mobile Device Forensics  (Total Chunks: 33, Direct: 17)
        |__ Lab 12.2. Using FTK Imager Lite to View Text Messages, Phone
Numbers, and Photos  (Total Chunks: 8, Direct: 8)
        |__ Lab 12.3. Using Autopsy to Search Cloud Backups of Mobile Devices
(Total Chunks: 5, Direct: 5)
        |__ Lab 12.1. Examining Cell Phone Storage Devices  (Total Chunks: 3,
Direct: 3)
    |__ Chapter 11. E-mail and Social Media Investigations  (Total Chunks: 32,
Direct: 4)
        |__ Lab 11.3. Finding Google Searches and Multiple E-mail Accounts
(Total Chunks: 22, Direct: 22)
        |__ Lab 11.1. Using OSForensics to Search for E-mails and Mailboxes
(Total Chunks: 3, Direct: 3)
```

```
        |__ Lab 11.2. Using Autopsy to Search for E-mails and Mailboxes  (Total
Chunks: 3, Direct: 3)
    |__ Chapter 7. Linux and Macintosh File Systems  (Total Chunks: 17, Direct:
6)
        |__ Lab 7.3. Using Autopsy to Process a Linux Image  (Total Chunks: 5,
Direct: 5)
        |__ Lab 7.1. Using Autopsy to Process a Mac OS X Image  (Total Chunks:
3, Direct: 3)
        |__ Lab 7.2. Using Autopsy to Process a Mac OS 9 Image  (Total Chunks:
3, Direct: 3)
    |__ Chapter 14. Report Writing for High-Tech Investigations  (Total Chunks:
16, Direct: 5)
        |__ Lab 14.3. Preparing a Report  (Total Chunks: 5, Direct: 5)
        |__ Lab 14.1. Investigating Corporate Espionage  (Total Chunks: 3,
Direct: 3)
        |__ Lab 14.2. Adding Evidence to a Case  (Total Chunks: 3, Direct: 3)
    |__ Introduction  (Total Chunks: 10, Direct: 10)
|__ Chapter 6. Current Digital Forensics Tools  (Total Chunks: 1440, Direct: 0)
    |__ Evaluating Digital Forensics Tool Needs  (Total Chunks: 1090, Direct: 1)
        |__ Tasks Performed by Digital Forensics Tools  (Total Chunks: 743,
Direct: 1)
            |__ Acquisition  (Total Chunks: 637, Direct: 637)
            |__ Reporting  (Total Chunks: 38, Direct: 38)
            |__ Extraction  (Total Chunks: 33, Direct: 33)
            |__ Reconstruction  (Total Chunks: 29, Direct: 29)
            |__ Validation and Verification  (Total Chunks: 5, Direct: 5)
        |__ Types of Digital Forensics Tools  (Total Chunks: 341, Direct: 9)
            |__ Software Forensics Tools  (Total Chunks: 308, Direct: 308)
            |__ Hardware Forensics Tools  (Total Chunks: 24, Direct: 24)
        |__ Tool Comparisons  (Total Chunks: 4, Direct: 4)
        |__ Other Considerations for Tools  (Total Chunks: 1, Direct: 1)
    |__ Digital Forensics Software Tools  (Total Chunks: 197, Direct: 4)
        |__ Linux Forensics Tools  (Total Chunks: 166, Direct: 27)
            |__ Kali Linux  (Total Chunks: 66, Direct: 66)
            |__ Smart  (Total Chunks: 51, Direct: 51)
            |__ Autopsy and Sleuth Kit  (Total Chunks: 17, Direct: 17)
            |__ Forcepoint Threat Protection  (Total Chunks: 3, Direct: 3)
            |__ Helix 3  (Total Chunks: 2, Direct: 2)
        |__ Command-Line Forensics Tools  (Total Chunks: 22, Direct: 22)
        |__ Other GUI Forensics Tools  (Total Chunks: 5, Direct: 5)
    |__ Digital Forensics Hardware Tools  (Total Chunks: 133, Direct: 18)
        |__ Forensic Workstations  (Total Chunks: 100, Direct: 97)
            |__ Building Your Own Workstation  (Total Chunks: 3, Direct: 3)
        |__ Recommendations for a Forensic Workstation  (Total Chunks: 8,
Direct: 8)
        |__ Using a Write-Blocker  (Total Chunks: 7, Direct: 7)
    |__ Validating and Testing Forensics Software  (Total Chunks: 20, Direct:
10)
```

```
        |__ Using Validation Protocols  (Total Chunks: 9, Direct: 1)
            |__ Digital Forensics Examination Protocol  (Total Chunks: 4,
Direct: 4)
            |__ Digital Forensics Tool Upgrade Protocol  (Total Chunks: 4,
Direct: 4)
        |__ Using National Institute of Standards and Technology Tools  (Total
Chunks: 1, Direct: 1)
|__ Chapter 5. Working with Windows and CLI Systems  (Total Chunks: 1310,
Direct: 0)
    |__ Understanding Microsoft Startup Tasks  (Total Chunks: 785, Direct: 3)
        |__ Startup in Windows 7, Windows 8, and Windows 10  (Total Chunks: 745,
Direct: 745)
        |__ Startup in Windows NT and Later  (Total Chunks: 37, Direct: 14)
            |__ Windows XP System Files  (Total Chunks: 14, Direct: 14)
            |__ Startup Files for Windows Vista  (Total Chunks: 6, Direct: 6)
            |__ Startup Files for Windows XP  (Total Chunks: 2, Direct: 2)
            |__ Contamination Concerns with Windows XP  (Total Chunks: 1,
Direct: 1)
    |__ Examining NTFS Disks  (Total Chunks: 258, Direct: 21)
        |__ MFT Structures for File Data  (Total Chunks: 70, Direct: 2)
            |__ Attribute 0x30: File Name  (Total Chunks: 18, Direct: 18)
            |__ Attribute 0x80: Data for a Nonresident File  (Total Chunks: 16,
Direct: 16)
            |__ MFT Header Fields  (Total Chunks: 14, Direct: 14)
            |__ Attribute 0x40: Object_ID  (Total Chunks: 10, Direct: 10)
            |__ Attribute 0x80: Data for a Resident File  (Total Chunks: 8,
Direct: 8)
            |__ Attribute 0x10: Standard Information  (Total Chunks: 1, Direct:
1)
            |__ Interpreting a Data Run  (Total Chunks: 1, Direct: 1)
        |__ MFT and File Attributes  (Total Chunks: 46, Direct: 46)
        |__ Deleting NTFS Files  (Total Chunks: 31, Direct: 31)
        |__ NTFS Alternate Data Streams  (Total Chunks: 29, Direct: 29)
        |__ Resilient File System  (Total Chunks: 28, Direct: 28)
        |__ NTFS System Files  (Total Chunks: 19, Direct: 19)
        |__ EFS Recovery Key Agent  (Total Chunks: 8, Direct: 8)
        |__ NTFS Compressed Files  (Total Chunks: 4, Direct: 4)
        |__ NTFS Encrypting File System  (Total Chunks: 2, Direct: 2)
    |__ Exploring Microsoft File Structures  (Total Chunks: 160, Direct: 1)
        |__ Disk Partitions  (Total Chunks: 148, Direct: 148)
        |__ Examining FAT Disks  (Total Chunks: 11, Direct: 5)
            |__ Deleting FAT Files  (Total Chunks: 6, Direct: 6)
    |__ Understanding File Systems  (Total Chunks: 58, Direct: 21)
        |__ Understanding Disk Drives  (Total Chunks: 16, Direct: 16)
        |__ Solid-State Storage Devices  (Total Chunks: 11, Direct: 11)
        |__ Understanding the Boot Sequence  (Total Chunks: 10, Direct: 10)
    |__ Understanding the Windows Registry  (Total Chunks: 22, Direct: 10)
        |__ Exploring the Organization of the Windows Registry  (Total Chunks:
```

```
11, Direct: 11)
        |__ Examining the Windows Registry  (Total Chunks: 1, Direct: 1)
    |__ Understanding Whole Disk Encryption  (Total Chunks: 21, Direct: 12)
        |__ Examining Third-Party Disk Encryption Tools  (Total Chunks: 8,
Direct: 8)
        |__ Examining Microsoft BitLocker  (Total Chunks: 1, Direct: 1)
    |__ Understanding Virtual Machines  (Total Chunks: 6, Direct: 2)
        |__ Creating a Virtual Machine  (Total Chunks: 4, Direct: 4)
|__ Chapter 10. Virtual Machine Forensics, Live Acquisitions, and Network
Forensics  (Total Chunks: 699, Direct: 0)
    |__ An Overview of Virtual Machine Forensics  (Total Chunks: 618, Direct: 2)
        |__ Type 2 Hypervisors  (Total Chunks: 608, Direct: 502)
            |__ VirtualBox  (Total Chunks: 73, Direct: 73)
            |__ VMware Workstation and Workstation Player  (Total Chunks: 13,
Direct: 13)
            |__ Microsoft Hyper-V  (Total Chunks: 12, Direct: 12)
            |__ Parallels Desktop  (Total Chunks: 7, Direct: 7)
            |__ KVM  (Total Chunks: 1, Direct: 1)
        |__ Working with Type 1 Hypervisors  (Total Chunks: 6, Direct: 6)
        |__ Conducting an Investigation with Type 2 Hypervisors  (Total Chunks:
2, Direct: 0)
            |__ Other VM Examination Methods  (Total Chunks: 1, Direct: 1)
            |__ Using VMs as Forensics Tools  (Total Chunks: 1, Direct: 1)
    |__ Network Forensics Overview  (Total Chunks: 76, Direct: 3)
        |__ Developing Procedures for Network Forensics  (Total Chunks: 55,
Direct: 7)
            |__ Using Network Tools  (Total Chunks: 29, Direct: 29)
            |__ Using Packet Analyzers  (Total Chunks: 10, Direct: 10)
            |__ Reviewing Network Logs  (Total Chunks: 9, Direct: 9)
        |__ Investigating Virtual Networks  (Total Chunks: 10, Direct: 10)
        |__ Securing a Network  (Total Chunks: 5, Direct: 5)
        |__ The Need for Established Procedures  (Total Chunks: 2, Direct: 2)
        |__ Examining the Honeynet Project  (Total Chunks: 1, Direct: 1)
    |__ Performing Live Acquisitions  (Total Chunks: 5, Direct: 1)
        |__ Performing a Live Acquisition in Windows  (Total Chunks: 4, Direct:
4)
|__ Chapter 16. Ethics for the Expert Witness  (Total Chunks: 593, Direct: 0)
    |__ Chapter Review  (Total Chunks: 400, Direct: 9)
        |__ Key Terms  (Total Chunks: 125, Direct: 125)
        |__ Case Projects  (Total Chunks: 110, Direct: 110)
        |__ Hands-On Projects  (Total Chunks: 80, Direct: 80)
        |__ Chapter Summary  (Total Chunks: 76, Direct: 76)
    |__ Organizations with Codes of Ethics  (Total Chunks: 110, Direct: 3)
        |__ International High Technology Crime Investigation Association
(Total Chunks: 43, Direct: 43)
        |__ International Association of Computer Investigative Specialists
(Total Chunks: 24, Direct: 24)
        |__ International Society of Forensic Computer Examiners  (Total Chunks:
```

```
20, Direct: 20)
        |__ American Bar Association  (Total Chunks: 17, Direct: 17)
        |__ American Psychological Association  (Total Chunks: 3, Direct: 3)
    |__ Applying Ethics and Codes to Expert Witnesses  (Total Chunks: 43,
Direct: 11)
        |__ Traps for Unwary Experts  (Total Chunks: 17, Direct: 17)
        |__ Determining Admissibility of Evidence  (Total Chunks: 7, Direct: 7)
        |__ Forensics Examiners' Roles in Testifying  (Total Chunks: 7, Direct:
7)
        |__ Considerations in Disqualification  (Total Chunks: 1, Direct: 1)
    |__ An Ethics Exercise  (Total Chunks: 26, Direct: 4)
        |__ Interpreting Attribute 0x80 Data Runs  (Total Chunks: 14, Direct: 1)
            |__ Calculating Data Runs  (Total Chunks: 7, Direct: 7)
            |__ Finding Attribute 0x80 an MFT Record  (Total Chunks: 4, Direct:
4)
            |__ Configuring Data Interpreter Options in WinHex  (Total Chunks:
2, Direct: 2)
        |__ Carving Data Run Clusters Manually  (Total Chunks: 4, Direct: 4)
        |__ Performing the Exam  (Total Chunks: 2, Direct: 1)
            |__ Preparing for an Examination  (Total Chunks: 1, Direct: 1)
        |__ Performing a Cursory Exam of a Forensic Image  (Total Chunks: 1,
Direct: 1)
        |__ Performing a Detailed Exam of a Forensic Image  (Total Chunks: 1,
Direct: 1)
    |__ Ethical Difficulties in Expert Testimony  (Total Chunks: 14, Direct: 3)
        |__ Ethical Responsibilities Owed to You  (Total Chunks: 6, Direct: 6)
        |__ Standard Forensics Tools and Tools You Create  (Total Chunks: 5,
Direct: 5)
|__ Chapter 13. Cloud Forensics  (Total Chunks: 575, Direct: 0)
    |__ Conducting a Cloud Investigation  (Total Chunks: 201, Direct: 2)
        |__ Examining Stored Cloud Data on a PC  (Total Chunks: 182, Direct: 1)
            |__ Google Drive  (Total Chunks: 80, Direct: 80)
            |__ OneDrive  (Total Chunks: 54, Direct: 54)
            |__ Dropbox  (Total Chunks: 47, Direct: 47)
        |__ Investigating CSPs  (Total Chunks: 11, Direct: 11)
        |__ Understanding Prefetch Files  (Total Chunks: 3, Direct: 3)
        |__ Windows Prefetch Artifacts  (Total Chunks: 2, Direct: 2)
        |__ Investigating Cloud Customers  (Total Chunks: 1, Direct: 1)
    |__ Technical Challenges in Cloud Forensics  (Total Chunks: 168, Direct: 9)
        |__ Architecture  (Total Chunks: 95, Direct: 95)
        |__ Standards and Training  (Total Chunks: 31, Direct: 31)
        |__ Anti-Forensics  (Total Chunks: 15, Direct: 15)
        |__ Role Management  (Total Chunks: 8, Direct: 8)
        |__ Incident First Responders  (Total Chunks: 6, Direct: 6)
        |__ Analysis of Cloud Forensic Data  (Total Chunks: 4, Direct: 4)
    |__ Legal Challenges in Cloud Forensics  (Total Chunks: 132, Direct: 2)
        |__ Accessing Evidence in the Cloud  (Total Chunks: 84, Direct: 5)
            |__ Search Warrants  (Total Chunks: 52, Direct: 52)
```

```
            |__ Subpoenas and Court Orders  (Total Chunks: 27, Direct: 27)
        |__ Jurisdiction Issues  (Total Chunks: 23, Direct: 23)
        |__ Service Level Agreements  (Total Chunks: 23, Direct: 15)
            |__ Policies, Standards, and Guidelines for CSPs  (Total Chunks: 6,
Direct: 6)
            |__ CSP Processes and Procedures  (Total Chunks: 2, Direct: 2)
    |__ An Overview of Cloud Computing  (Total Chunks: 38, Direct: 1)
        |__ Cloud Vendors  (Total Chunks: 18, Direct: 18)
        |__ Cloud Service Levels and Deployment Methods  (Total Chunks: 12,
Direct: 12)
        |__ History of the Cloud  (Total Chunks: 4, Direct: 4)
        |__ Basic Concepts of Cloud Forensics  (Total Chunks: 3, Direct: 3)
    |__ Tools for Cloud Forensics  (Total Chunks: 23, Direct: 19)
        |__ Magnet AXIOM Cloud  (Total Chunks: 2, Direct: 2)
        |__ Forensic Open-Stack Tools  (Total Chunks: 1, Direct: 1)
        |__ F-Response for the Cloud  (Total Chunks: 1, Direct: 1)
    |__ Acquisitions in the Cloud  (Total Chunks: 13, Direct: 9)
        |__ Encryption in the Cloud  (Total Chunks: 4, Direct: 4)
|__ Chapter 1. Understanding the Digital Forensics Profession and Investigations
(Total Chunks: 365, Direct: 0)
    |__ Conducting an Investigation  (Total Chunks: 120, Direct: 28)
        |__ Gathering the Evidence  (Total Chunks: 40, Direct: 40)
        |__ Completing the Case  (Total Chunks: 20, Direct: 16)
            |__ Autopsy's Report Generator  (Total Chunks: 4, Direct: 4)
        |__ Analyzing Your Digital Evidence  (Total Chunks: 19, Direct: 7)
            |__ Some Additional Features of Autopsy  (Total Chunks: 12, Direct:
12)
        |__ Critiquing the Case  (Total Chunks: 9, Direct: 9)
        |__ Understanding Bit-stream Copies  (Total Chunks: 4, Direct: 3)
            |__ Acquiring an Image of Evidence Media  (Total Chunks: 1, Direct:
1)
    |__ Preparing a Digital Forensics Investigation  (Total Chunks: 79, Direct:
5)
        |__ Taking a Systematic Approach  (Total Chunks: 67, Direct: 19)
            |__ Securing Your Evidence  (Total Chunks: 35, Direct: 35)
            |__ Assessing the Case  (Total Chunks: 10, Direct: 10)
            |__ Planning Your Investigation  (Total Chunks: 3, Direct: 3)
        |__ An Overview of a Company Policy Violation  (Total Chunks: 4, Direct:
4)
        |__ An Overview of a Computer Crime  (Total Chunks: 3, Direct: 3)
    |__ Procedures for Private-Sector High-Tech Investigations  (Total Chunks:
76, Direct: 2)
        |__ Industrial Espionage Investigations  (Total Chunks: 28, Direct: 14)
            |__ Interviews and Interrogations in High-Tech Investigations
(Total Chunks: 14, Direct: 14)
        |__ Internet Abuse Investigations  (Total Chunks: 26, Direct: 26)
        |__ E-mail Abuse Investigations  (Total Chunks: 9, Direct: 9)
        |__ Attorney-Client Privilege Investigations  (Total Chunks: 8, Direct:
```

```
            |__ Other Training and Certifications  (Total Chunks: 5, Direct: 5)
            |__ AccessData Certified Examiner  (Total Chunks: 2, Direct: 2)
            |__ EnCase Certified Examiner Certification  (Total Chunks: 1,
Direct: 1)
        |__ Lab Budget Planning  (Total Chunks: 6, Direct: 6)
        |__ Identifying Duties of the Lab Manager and Staff  (Total Chunks: 1,
Direct: 1)
    |__ Determining the Physical Requirements for a Digital Forensics Lab
(Total Chunks: 45, Direct: 6)
        |__ Auditing a Digital Forensics Lab  (Total Chunks: 11, Direct: 11)
        |__ Identifying Lab Security Needs  (Total Chunks: 8, Direct: 8)
        |__ Considering Physical Security Needs  (Total Chunks: 8, Direct: 8)
        |__ Overseeing Facility Maintenance  (Total Chunks: 5, Direct: 5)
        |__ Conducting High-Risk Investigations  (Total Chunks: 4, Direct: 4)
        |__ Using Evidence Containers  (Total Chunks: 2, Direct: 2)
        |__ Determining Floor Plans for Digital Forensics Labs  (Total Chunks:
1, Direct: 1)
    |__ Selecting a Basic Forensic Workstation  (Total Chunks: 40, Direct: 5)
        |__ Stocking Hardware Peripherals  (Total Chunks: 14, Direct: 14)
        |__ Using a Disaster Recovery Plan  (Total Chunks: 10, Direct: 10)
        |__ Selecting Workstations for Private-Sector Labs  (Total Chunks: 5,
Direct: 5)
        |__ Planning for Equipment Upgrades  (Total Chunks: 3, Direct: 3)
        |__ Maintaining Operating Systems and Software Inventories  (Total
Chunks: 2, Direct: 2)
        |__ Selecting Workstations for a Lab  (Total Chunks: 1, Direct: 1)
|__ Chapter 15. Expert Testimony in Digital Investigations  (Total Chunks: 286,
Direct: 0)
    |__ Testifying in Court  (Total Chunks: 146, Direct: 36)
        |__ General Guidelines on Testifying  (Total Chunks: 49, Direct: 13)
            |__ Using Graphics During Testimony  (Total Chunks: 26, Direct: 26)
            |__ Avoiding Testimony Problems  (Total Chunks: 7, Direct: 7)
            |__ Understanding Prosecutorial Misconduct  (Total Chunks: 3,
Direct: 3)
        |__ Testifying During Cross-Examination  (Total Chunks: 31, Direct: 31)
        |__ Understanding the Trial Process  (Total Chunks: 11, Direct: 11)
        |__ Testifying During Direct Examination  (Total Chunks: 11, Direct: 11)
        |__ Providing Qualifications for Your Testimony  (Total Chunks: 8,
Direct: 8)
    |__ Preparing for Testimony  (Total Chunks: 50, Direct: 1)
        |__ Reviewing Your Role as a Consulting Expert or an Expert Witness
(Total Chunks: 14, Direct: 14)
        |__ Creating and Maintaining Your CV  (Total Chunks: 10, Direct: 10)
        |__ Preparing Technical Definitions  (Total Chunks: 10, Direct: 10)
        |__ Documenting and Preparing Evidence  (Total Chunks: 8, Direct: 8)
        |__ Preparing to Deal with the News Media  (Total Chunks: 7, Direct: 7)
    |__ Preparing Forensics Evidence for Testimony  (Total Chunks: 48, Direct:
17)
```

```
        |__ Preparing a Defense of Your Evidence-Collection Methods  (Total
Chunks: 31, Direct: 31)
    |__ Preparing for a Deposition or Hearing  (Total Chunks: 42, Direct: 8)
        |__ Guidelines for Testifying at Depositions  (Total Chunks: 21, Direct:
11)
            |__ Recognizing Deposition Problems  (Total Chunks: 10, Direct: 10)
        |__ Guidelines for Testifying at Hearings  (Total Chunks: 13, Direct:
13)
|__ Chapter 4. Processing Crime and Incident Scenes  (Total Chunks: 252, Direct:
0)
    |__ Reviewing a Case  (Total Chunks: 58, Direct: 4)
        |__ An Example of a Criminal Investigation  (Total Chunks: 23, Direct:
23)
        |__ Conducting the Investigation: Acquiring Evidence with OSForensics
(Total Chunks: 16, Direct: 16)
        |__ Planning the Investigation  (Total Chunks: 14, Direct: 14)
        |__ Sample Civil Investigation  (Total Chunks: 1, Direct: 1)
    |__ Storing Digital Evidence  (Total Chunks: 47, Direct: 23)
        |__ Documenting Evidence  (Total Chunks: 18, Direct: 18)
        |__ Evidence Retention and Media Storage Needs  (Total Chunks: 6,
Direct: 6)
    |__ Seizing Digital Evidence at the Scene  (Total Chunks: 37, Direct: 2)
        |__ Processing Incident or Crime Scenes  (Total Chunks: 19, Direct: 19)
        |__ Using a Technical Advisor  (Total Chunks: 10, Direct: 10)
        |__ Processing and Handling Digital Evidence  (Total Chunks: 4, Direct:
4)
        |__ Preparing to Acquire Digital Evidence  (Total Chunks: 1, Direct: 1)
        |__ Processing Data Centers with RAID Systems  (Total Chunks: 1, Direct:
1)
    |__ Preparing for a Search  (Total Chunks: 34, Direct: 3)
        |__ Determining the Tools You Need  (Total Chunks: 12, Direct: 12)
        |__ Getting a Detailed Description of the Location  (Total Chunks: 5,
Direct: 5)
        |__ Identifying the Type of OS or Digital Device  (Total Chunks: 4,
Direct: 4)
        |__ Determining Who Is in Charge  (Total Chunks: 3, Direct: 3)
        |__ Using Additional Technical Expertise  (Total Chunks: 3, Direct: 3)
        |__ Identifying the Nature of the Case  (Total Chunks: 2, Direct: 2)
        |__ Determining Whether You Can Seize Computers and Digital Devices
(Total Chunks: 1, Direct: 1)
        |__ Preparing the Investigation Team  (Total Chunks: 1, Direct: 1)
    |__ Processing Law Enforcement Crime Scenes  (Total Chunks: 32, Direct: 12)
        |__ Understanding Concepts and Terms Used in Warrants  (Total Chunks:
20, Direct: 20)
    |__ Identifying Digital Evidence  (Total Chunks: 19, Direct: 14)
        |__ Understanding Rules of Evidence  (Total Chunks: 5, Direct: 5)
    |__ Obtaining a Digital Hash  (Total Chunks: 9, Direct: 9)
    |__ Collecting Evidence in Private-Sector Incident Scenes  (Total Chunks: 8,
```

```
Direct: 8)
    |__ Securing a Digital Incident or Crime Scene  (Total Chunks: 8, Direct: 8)
|__ Chapter 3. Data Acquisition  (Total Chunks: 231, Direct: 0)
    |__ Understanding Storage Formats for Digital Evidence  (Total Chunks: 94,
Direct: 1)
        |__ Raw Format  (Total Chunks: 46, Direct: 46)
        |__ Proprietary Formats  (Total Chunks: 39, Direct: 39)
        |__ Advanced Forensic Format  (Total Chunks: 8, Direct: 8)
    |__ Validating Data Acquisitions  (Total Chunks: 42, Direct: 5)
        |__ Linux Validation Methods  (Total Chunks: 36, Direct: 1)
            |__ Validating dcfldd-Acquired Data  (Total Chunks: 31, Direct: 31)
            |__ Validating dd-Acquired Data  (Total Chunks: 4, Direct: 4)
        |__ Windows Validation Methods  (Total Chunks: 1, Direct: 1)
    |__ Using Acquisition Tools  (Total Chunks: 39, Direct: 0)
        |__ Acquiring Data with a Linux Boot CD  (Total Chunks: 34, Direct: 1)
            |__ Acquiring Data with dcfldd in Linux  (Total Chunks: 14, Direct:
14)
            |__ Using Linux Live CD Distributions  (Total Chunks: 11, Direct:
11)
            |__ Acquiring Data with dd in Linux  (Total Chunks: 8, Direct: 8)
        |__ Capturing an Image with AccessData FTK Imager Lite  (Total Chunks:
4, Direct: 4)
        |__ Mini-WinFE Boot CDs and USB Drives  (Total Chunks: 1, Direct: 1)
    |__ Performing RAID Data Acquisitions  (Total Chunks: 27, Direct: 6)
        |__ Understanding RAID  (Total Chunks: 16, Direct: 16)
        |__ Acquiring RAID Disks  (Total Chunks: 5, Direct: 5)
    |__ Using Other Forensics Acquisition Tools  (Total Chunks: 20, Direct: 0)
        |__ ASR Data SMART  (Total Chunks: 8, Direct: 8)
        |__ Runtime Software  (Total Chunks: 6, Direct: 6)
        |__ SourceForge  (Total Chunks: 3, Direct: 3)
        |__ ILookIX IXImager  (Total Chunks: 2, Direct: 2)
        |__ PassMark Software ImageUSB  (Total Chunks: 1, Direct: 1)
    |__ Using Remote Network Acquisition Tools  (Total Chunks: 8, Direct: 5)
        |__ Remote Acquisition with F-Response  (Total Chunks: 3, Direct: 3)
    |__ Contingency Planning for Image Acquisitions  (Total Chunks: 1, Direct:
1)
|__ Chapter 8. Recovering Graphics Files  (Total Chunks: 221, Direct: 0)
    |__ Understanding Data Compression  (Total Chunks: 107, Direct: 1)
        |__ Searching for and Carving Data from Unallocated Space  (Total
Chunks: 56, Direct: 52)
            |__ Planning Your Examination  (Total Chunks: 3, Direct: 3)
            |__ Searching for and Recovering Digital Photograph Evidence  (Total
Chunks: 1, Direct: 1)
        |__ Lossless and Lossy Compression  (Total Chunks: 26, Direct: 26)
        |__ Rebuilding File Headers  (Total Chunks: 7, Direct: 7)
        |__ Reconstructing File Fragments  (Total Chunks: 6, Direct: 6)
        |__ Repairing Damaged Headers  (Total Chunks: 6, Direct: 6)
        |__ Locating and Recovering Graphics Files  (Total Chunks: 4, Direct: 4)
```

```
        |__ Identifying Graphics File Fragments  (Total Chunks: 1, Direct: 1)
    |__ Recognizing a Graphics File  (Total Chunks: 69, Direct: 14)
        |__ Understanding Digital Photograph File Formats  (Total Chunks: 29,
Direct: 1)
            |__ Examining the Exchangeable Image File Format  (Total Chunks: 19,
Direct: 19)
            |__ Examining the Raw File Format  (Total Chunks: 9, Direct: 9)
        |__ Understanding Bitmap and Raster Images  (Total Chunks: 16, Direct:
16)
        |__ Understanding Graphics File Formats  (Total Chunks: 4, Direct: 4)
        |__ Understanding Vector Graphics  (Total Chunks: 3, Direct: 3)
        |__ Understanding Metafile Graphics  (Total Chunks: 3, Direct: 3)
    |__ Identifying Unknown File Formats  (Total Chunks: 44, Direct: 4)
        |__ Understanding Steganography in Graphics Files  (Total Chunks: 19,
Direct: 19)
        |__ Analyzing Graphics File Headers  (Total Chunks: 11, Direct: 11)
        |__ Using Steganalysis Tools  (Total Chunks: 8, Direct: 8)
        |__ Tools for Viewing Images  (Total Chunks: 2, Direct: 2)
    |__ Understanding Copyright Issues with Graphics  (Total Chunks: 1, Direct:
1)
|__ Chapter 11. E-mail and Social Media Investigations  (Total Chunks: 176,
Direct: 0)
    |__ Investigating E-mail Crimes and Violations  (Total Chunks: 90, Direct:
23)
        |__ Using Network E-mail Logs  (Total Chunks: 17, Direct: 17)
        |__ Tracing an E-mail Message  (Total Chunks: 14, Direct: 14)
        |__ Examining E-mail Messages  (Total Chunks: 12, Direct: 10)
            |__ Copying an E-mail Message  (Total Chunks: 2, Direct: 2)
        |__ Understanding Forensic Linguistics  (Total Chunks: 9, Direct: 9)
        |__ Examining E-mail Headers  (Total Chunks: 6, Direct: 6)
        |__ Viewing E-mail Headers  (Total Chunks: 5, Direct: 5)
        |__ Examining Additional E-mail Files  (Total Chunks: 4, Direct: 4)
    |__ Using Specialized E-mail Forensics Tools  (Total Chunks: 37, Direct: 22)
        |__ Using a Hex Editor to Carve E-mail Messages  (Total Chunks: 8,
Direct: 8)
        |__ Recovering Outlook Files  (Total Chunks: 4, Direct: 4)
        |__ Using Magnet AXIOM to Recover E-mail  (Total Chunks: 2, Direct: 2)
        |__ E-mail Case Studies  (Total Chunks: 1, Direct: 1)
    |__ Understanding E-mail Servers  (Total Chunks: 24, Direct: 2)
        |__ Examining UNIX E-mail Server Logs  (Total Chunks: 14, Direct: 14)
        |__ Examining Microsoft E-mail Server Logs  (Total Chunks: 8, Direct: 8)
    |__ Applying Digital Forensics Methods to Social Media Communications
(Total Chunks: 20, Direct: 19)
        |__ Forensics Tools for Social Media Investigations  (Total Chunks: 1,
Direct: 1)
    |__ Exploring the Role of E-mail in Investigations  (Total Chunks: 4,
Direct: 4)
    |__ Exploring the Roles of the Client and Server in E-mail  (Total Chunks:
```

```
1, Direct: 1)
|__ Chapter 7. Linux and Macintosh File Systems  (Total Chunks: 144, Direct: 0)
    |__ Examining Linux File Structures  (Total Chunks: 104, Direct: 17)
        |__ File Structures in Ext4  (Total Chunks: 87, Direct: 7)
            |__ Inodes  (Total Chunks: 63, Direct: 63)
            |__ Hard Links and Symbolic Links  (Total Chunks: 17, Direct: 17)
    |__ Understanding Macintosh File Structures  (Total Chunks: 30, Direct: 1)
        |__ An Overview of Mac File Structures  (Total Chunks: 28, Direct: 28)
        |__ Forensics Procedures in Mac  (Total Chunks: 1, Direct: 1)
    |__ Using Linux Forensics Tools  (Total Chunks: 10, Direct: 8)
        |__ Examining a Case with Sleuth Kit and Autopsy  (Total Chunks: 2,
Direct: 2)
|__ Chapter 9. Digital Forensics Analysis and Validation  (Total Chunks: 144,
Direct: 0)
    |__ Addressing Data-Hiding Techniques  (Total Chunks: 98, Direct: 1)
        |__ Recovering Passwords  (Total Chunks: 27, Direct: 27)
        |__ Examining Encrypted Files  (Total Chunks: 22, Direct: 22)
        |__ Bit-Shifting  (Total Chunks: 20, Direct: 20)
        |__ Understanding Steganalysis Methods  (Total Chunks: 12, Direct: 12)
        |__ Marking Bad Clusters  (Total Chunks: 8, Direct: 8)
        |__ Hiding Partitions  (Total Chunks: 4, Direct: 4)
        |__ Hiding Files by Using the OS  (Total Chunks: 4, Direct: 4)
    |__ Validating Forensic Data  (Total Chunks: 23, Direct: 9)
        |__ Validating with Digital Forensics Tools  (Total Chunks: 10, Direct:
10)
        |__ Validating with Hexadecimal Editors  (Total Chunks: 4, Direct: 3)
            |__ Using Hash Values to Discriminate Data  (Total Chunks: 1,
Direct: 1)
    |__ Determining What Data to Collect and Analyze  (Total Chunks: 23, Direct:
8)
        |__ Using Autopsy to Validate Data  (Total Chunks: 11, Direct: 8)
            |__ Installing NSRL Hashes in Autopsy  (Total Chunks: 3, Direct: 3)
        |__ Approaching Digital Forensics Cases  (Total Chunks: 3, Direct: 2)
            |__ Refining and Modifying the Investigation Plan  (Total Chunks: 1,
Direct: 1)
        |__ Collecting Hash Values in Autopsy  (Total Chunks: 1, Direct: 1)
|__ Chapter 14. Report Writing for High-Tech Investigations  (Total Chunks: 140,
Direct: 0)
    |__ Guidelines for Writing Reports  (Total Chunks: 113, Direct: 25)
        |__ Designing the Layout and Presentation of Reports  (Total Chunks: 52,
Direct: 5)
            |__ Providing References  (Total Chunks: 23, Direct: 23)
            |__ Including Appendixes  (Total Chunks: 10, Direct: 10)
            |__ Explaining Examination and Data Collection Methods  (Total
Chunks: 4, Direct: 4)
            |__ Providing Supporting Material  (Total Chunks: 3, Direct: 3)
            |__ Providing for Uncertainty and Error Analysis  (Total Chunks: 3,
Direct: 3)
```

```
            |__ Formatting Consistently  (Total Chunks: 2, Direct: 2)
            |__ Including Calculations  (Total Chunks: 1, Direct: 1)
            |__ Explaining Results and Conclusions  (Total Chunks: 1, Direct: 1)
        |__ Writing Reports Clearly  (Total Chunks: 16, Direct: 9)
            |__ Including Signposts  (Total Chunks: 4, Direct: 4)
            |__ Considering Writing Style  (Total Chunks: 3, Direct: 3)
        |__ What to Include in Written Preliminary Reports  (Total Chunks: 11,
Direct: 11)
        |__ Report Structure  (Total Chunks: 9, Direct: 9)
    |__ Understanding the Importance of Reports  (Total Chunks: 25, Direct: 1)
        |__ Types of Reports  (Total Chunks: 23, Direct: 23)
        |__ Limiting a Report to Specifics  (Total Chunks: 1, Direct: 1)
    |__ Generating Report Findings with Forensics Software Tools  (Total Chunks:
2, Direct: 0)
        |__ Using Autopsy to Generate Reports  (Total Chunks: 2, Direct: 2)
|__ Chapter 12. Mobile Device Forensics and the Internet of Anything  (Total
Chunks: 93, Direct: 0)
    |__ Understanding Mobile Device Forensics  (Total Chunks: 64, Direct: 4)
        |__ Inside Mobile Devices  (Total Chunks: 32, Direct: 9)
            |__ SIM Cards  (Total Chunks: 23, Direct: 23)
        |__ Mobile Phone Basics  (Total Chunks: 28, Direct: 28)
    |__ Understanding Acquisition Procedures for Mobile Devices  (Total Chunks:
28, Direct: 0)
        |__ Mobile Forensics Equipment  (Total Chunks: 24, Direct: 2)
            |__ SIM Card Readers  (Total Chunks: 21, Direct: 21)
            |__ Mobile Phone Forensics Tools and Methods  (Total Chunks: 1,
Direct: 1)
        |__ Using Mobile Forensics Tools  (Total Chunks: 4, Direct: 4)
    |__ Understanding Forensics in the Internet of Anything  (Total Chunks: 1,
Direct: 1)
|__ About the Authors  (Total Chunks: 18, Direct: 18)
|__ Copyright Page  (Total Chunks: 16, Direct: 16)
|__ Preface  (Total Chunks: 15, Direct: 15)
|__ Acknowledgments  (Total Chunks: 10, Direct: 10)
|__ Title Page  (Total Chunks: 7, Direct: 7)
|__ Appendix B. Digital Forensics References  (Total Chunks: 6, Direct: 6)
|__ Appendix D. Legacy File System and Forensics Tools  (Total Chunks: 6,
Direct: 6)
|__ Appendix C. Digital Forensics Lab Considerations  (Total Chunks: 5, Direct:
5)
|__ Cover Page  (Total Chunks: 1, Direct: 1)
|__ Appendix A. Certification Test References  (Total Chunks: 1, Direct: 1)

--------------------------------------------------------------------------------
                              Diagnostic Summary
--------------------------------------------------------------------------------
  Found 27 distinct top-level sections.
  All chunks contain 'toc_path' metadata.
```

```
================================================================================
                                Diagnostic Complete
================================================================================
```

[10]:
```python
# Cell 6: Database Health & Hierarchy Diagnostic Report

import os
import json
import logging
from typing import List, Dict
import pandas as pd # You might need to run: pip install pandas
from langchain_chroma import Chroma
from langchain_ollama.embeddings import OllamaEmbeddings

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

def print_header(text: str, char: str = "="):
    """Prints a centered header to the console."""
    print("\n" + char * 80)
    print(text.center(80))
    print(char * 80)

# --- Main Diagnostic Function ---
def run_diagnostics():
    print_header("Database Health & Hierarchy Diagnostic Report")

    # --- 1. Connect to the Database ---
    logger.info("Connecting to the vector database...")
    if not os.path.exists(CHROMA_PERSIST_DIR):
        logger.error(f"FATAL: Chroma DB directory not found at {CHROMA_PERSIST_DIR}. Please run Cell 5 first."); return

    # This assumes your global variables (CHROMA_PERSIST_DIR, etc.) are available
    embeddings = OllamaEmbeddings(model=EMBEDDING_MODEL_OLLAMA)
    vector_store = Chroma(persist_directory=CHROMA_PERSIST_DIR, embedding_function=embeddings, collection_name=CHROMA_COLLECTION_NAME)
    logger.info("Successfully connected to the database.")

    # --- 2. Retrieve ALL Metadata ---
    logger.info("Retrieving all metadata from the database. This may take a moment...")
    try:
        total_docs = vector_store._collection.count()
```

```
        if total_docs == 0:
            logger.error("Database is empty. Cannot run diagnostics."); return
        retrieved_data = vector_store.get(limit=total_docs,␣
↪include=["metadatas"])
        all_metadatas = retrieved_data['metadatas']
        logger.info(f"Successfully retrieved metadata for all␣
↪{len(all_metadatas)} chunks.")
    except Exception as e:
        logger.error(f"Failed to retrieve data from ChromaDB: {e}"); return

    # --- 3. Analyze Hierarchy Distribution ---
    print_header("Hierarchy Distribution Analysis", char="-")

    level_0_counts = {}
    chunks_without_level_0 = 0
    for meta in all_metadatas:
        level_0_title = meta.get("level_0_title")
        if level_0_title:
            level_0_counts[level_0_title] = level_0_counts.get(level_0_title,␣
↪0) + 1
        else:
            chunks_without_level_0 += 1

    if level_0_counts:
        print("\n Found the following top-level sections (level_0_title) and␣
↪their chunk counts:")
        # Use pandas to create a nicely formatted table
        df = pd.DataFrame(list(level_0_counts.items()), columns=['Top-Level␣
↪Section (level_0_title)', 'Chunk Count'])
        df = df.sort_values(by='Chunk Count', ascending=False).
↪reset_index(drop=True)
        print(df.to_string())
    else:
        logger.error(" CRITICAL ERROR: No chunks with 'level_0_title' metadata␣
↪were found!")

    if chunks_without_level_0 > 0:
        logger.warning(f"\n Found {chunks_without_level_0} chunks that are␣
↪MISSING the 'level_0_title' metadata entirely. This is a major sign of a␣
↪data processing error.")

    print_header("Diagnostic Complete")

# --- Execute Diagnostics ---
run_diagnostics()
```

```
2025-06-26 03:58:57,716 - INFO - Connecting to the vector database…
```

2025-06-26 03:58:57,731 - INFO - Successfully connected to the database.
2025-06-26 03:58:57,733 - INFO - Retrieving all metadata from the database. This may take a moment…

==
Database Health & Hierarchy Diagnostic Report
==

2025-06-26 03:58:58,380 - INFO - Successfully retrieved metadata for all 12498 chunks.

Hierarchy Distribution Analysis

Found the following top-level sections (level_0_title) and their chunk counts:

	Top-Level Section (level_0_title)	Chunk Count
0	Lab Manual for Guide to Computer Forensics and Investigations	4178
1	Chapter 16. Ethics for the Expert Witness	2550
2	Chapter 1. Understanding the Digital Forensics Profession and Investigations	594
3	Chapter 5. Working with Windows and CLI Systems	483
4	Chapter 6. Current Digital Forensics Tools	465
5	Chapter 4. Processing Crime and Incident Scenes	450
6	Chapter 3. Data Acquisition	410
7	Chapter 13. Cloud Forensics	365
8	Chapter 15. Expert Testimony in Digital Investigations	340
9	Chapter 2. The Investigator’s Office and Laboratory	339
10	Chapter 10. Virtual Machine Forensics, Live Acquisitions, and Network Forensics	330
11	Chapter 11. E-mail and Social Media Investigations	319
12	Chapter 8. Recovering Graphics Files	289
13	Chapter 7. Linux and Macintosh File Systems	287
14	Chapter 14. Report Writing for High-Tech	

Investigations 267
15 Chapter 9. Digital Forensics Analysis and
Validation 263
16 Chapter 12. Mobile Device Forensics and the Internet of
Anything 233
17 Appendix B. Digital Forensics
References 85
18 Appendix D. Legacy File System and Forensics
Tools 73
19 Appendix C. Digital Forensics Lab
Considerations 66
20 Appendix A. Certification Test
References 62
21
Acknowledgments 28
22
Preface 14
23 About the
Authors 8

==
Diagnostic Complete
==