📄 Assignment Statement

USB Power Delivery (USB PD) Specification Parsing and Structuring System

Title: Intelligent Parsing & Structuring of USB Power Delivery (USB PD) Specification Documents

Context:

USB Power Delivery (USB PD) specifications are complex technical documents containing a mix of structured and unstructured content including hierarchical sections (like chapters and subsections), figures, tables, and protocol/state machine descriptions.

To build an AI-powered domain assistant that understands and responds to questions on USB PD, we must convert these raw PDF specs into structured, machine-readable formats (like JSONL), preserving their logical hierarchy and metadata.

Objective:

You are tasked with building a prototype system that parses a USB PD specification PDF file and produces structured JSONL output representing the Table of Contents (ToC) hierarchy with associated page numbers and metadata.

This structured output will help downstream applications such as document search, validation, and knowledge graph creation.

Deliverables:

- Python script(s) that:
    - Extract the Table of Contents from the PDF
    - Parse each ToC line into: section_id, title, page number, and inferred hierarchy level
    - Generate a JSONL output file with one object per section
- A sample JSONL file (named: usb_pd_spec.jsonl) containing the parsed output
- README (or code comments) explaining how your script works

🗒️ Sample Output Format (JSONL):

Each line in the file should look like this:

{"doc_title": "USB PD Specification Rev X", "section_id": "2.1.2", "title": "Power Delivery Contract Negotiation", "page": 53, "level": 3, "parent_id": "2.1", "full_path": "2.1.2 Power Delivery Contract Negotiation"}

Steps to Follow:

1. Read this assignment fully.
2. Refer to the shared USB PD Specification PDF file.
3. Identify the Table of Contents section and extract its entries (can use PyMuPDF, pdfplumber, or any other PDF lib).
4. Parse each line of ToC to extract:
    - section_id (e.g. 2.1.2)
    - title (e.g. Power Delivery Contract Negotiation)
    - page number (e.g. 53)
    - level (inferred from the number of dots in section_id)
    - parent_id (e.g. 2.1 is parent of 2.1.2)
5. Generate the output as JSONL (each line = one section)
6. Name the final output file usb_pd_spec.jsonl
7. Submit:
    - Python script(s)
    - Output JSONL file
    - Short notes or comments if needed

📦 Output Files Required:

- usb_pd_parser.py or notebook
- usb_pd_spec.jsonl (output file)

- README.txt or comment block at top of code

Support:

- If you have technical questions, reach out over email for a quick discussion.
- You can also email questions if asynchronous is preferred.
- Bonus points for creating reusable functions and robust error handling.

Timeline:

Estimated effort: 2–5 working days

Prompt Chain Architecture

Below is a full prompt chain architecture (multi-step) for building a system that:

- Parses USB PD PDF specs
- Applies logical (not fixed-length) chunking
- Extracts and splits the Table of Contents (ToC)
- Cross-validates ToC with the actual parsed structure
- Builds a knowledge graph (optional)
- Generates insights + validation reports

We'll break this into modular prompt chains that can be executed in sequence or integrated into a larger pipeline.

📦 Prompt Chain Overview

Step Name	Purpose

Chain 1: Extract Table of Contents (ToC)	Extract hierarchical ToC from front matter of PDF
Chain 2: Logical Chunking	Break main document into semantically coherent sections
Chain 3: ToC-Driven Validation	Match chunked sections to ToC entries; detect mismatches
Chain 4: Content Metadata Extraction	Extract stats: #sections, figures, tables, headings
Chain 5: Graph Knowledge Construction	Build subject→section→relationship graphs
Chain 6: Validation Report Generation	Generate discrepancy report and insights dashboard

Each chain is outlined below with a full prompt or prompt template:

🔗 Chain 1: Extract Table of Contents (ToC)

Prompt:

You are an expert document parser. Given the raw extracted front matter text from a technical specification (usually the first 3–10 pages), extract the Table of Contents in structured JSONL format.

Each entry must include:

- level (chapter/section/subsection)
- title
- page_start (if available)
- full_path (e.g., "4.3.2 Device Role Swap Behavior")

Input:

{front_matter_text}

Output format:

```
[
 {
```

"level": "chapter",
"title": "4 USB PD Protocol Architecture",
"full_path": "4 USB PD Protocol Architecture",
"page_start": 32
},
{
"level": "subsection",
"title": "4.3.2 Device Role Swap Behavior",
"full_path": "4.3.2 Device Role Swap Behavior",
"page_start": 47
}
...
]

🧩 Chain 2: Logical Chunking

Prompt:

You are an AI parser for structured technical documents. Given a raw sequence of paragraphs and headings from a USB PD spec, chunk this document into semantically meaningful blocks.

Rules:

- Do not chunk by fixed length.
- Use changes in heading level, diagrams, and tables to detect logical boundaries.
- Preserve the full heading path and inferred section ID.
- Group related diagrams, tables, and explanatory text together.

Input:

{parsed_document_sections}

Output format:

[
{
"section_path": "4.3.2 Device Role Swap Behavior",
"start_heading": "4.3.2 Device Role Swap Behavior",
"content": "...",
"tables": ["Table 4-12"],
"figures": ["Figure 4-8"],
"page_range": [47, 49]

},
...
]

🔎 Chain 3: ToC-Driven Validation

Prompt:

You are a document structure validator. Given a parsed Table of Contents and a set of chunked document sections, match each ToC entry to the corresponding content chunk.

Check for the following:

- Missing sections from ToC (in parsed data)
- Extra sections in chunks (not listed in ToC)
- Duplicate or misaligned section headings
- Out-of-order section progression

Input:

{
"toc_entries": [...],
"chunked_sections": [...]
}

Output:

{
"toc_section_count": 94,
"parsed_section_count": 92,
"missing_sections": [...],
"extra_sections": [...],
"out_of_order_sections": [...],
"matched_sections": [...]
}

📊 Chain 4: Metadata & Content Insight Extraction

Prompt:

You are a metadata extractor for parsed specifications. Given a list of chunked sections, extract and count the following:

- Total chapters, sections, subsections
- Total figures and figure titles
- Total tables and table titles
- Sections with missing diagrams or tables
- Average length of section content (in tokens or words)

Input:

{chunked_sections}

Output:

```
{
 "total_chapters": 8,
 "total_sections": 94,
 "total_figures": 36,
 "total_tables": 28,
 "avg_tokens_per_section": 420,
 "sections_without_diagrams": [...],
 "sections_without_tables": [...]
 }
```

🧠 Chain 5: Graph Knowledge Builder (Optional)

Prompt:

You are a knowledge graph builder for a hardware protocol specification. Given the chunked document, extract subject-relation-object triples from each section.

Focus on relations between:

- Devices (e.g., Sink, Source, DRP)
- States (e.g., Role Swap, Contract Negotiation)
- Parameters (e.g., Vbus voltage, Rp resistor)
- Test Conditions or Capabilities

Output format:

[
{
"subject": "Sink device",
"relation": "requires",
"object": "Rp resistor between CC and GND"
},
...
]

📋 Chain 6: Final Report Generator

Prompt:

You are a report generator for document parsing QA. Given the ToC validation results and metadata stats, generate a concise summary report with:

- Overview of data quality (e.g., % matched vs. expected)
- Summary table of chapters/sections/tables/figures
- List of discrepancies
- Suggested fixable issues (e.g., merge short chunks, recheck OCR)

Output:

{
"summary": "Parsed 94 of 96 ToC sections (97.9% match)...",
"metrics": {
"toc_sections": 96,
"parsed_sections": 94,
"figures": 36,
"tables": 28,
"missing_sections": ["4.3.4...", "5.2.1..."]
},
"discrepancies": [
"Section 5.1.2 found in chunks but not in ToC",
"Table 5-6 missing from extracted content"
],
"recommendations": [
"Re-parse pages 73–75 for OCR quality",
"Merge 4.1.3.2 and 4.1.3.3 into one chunk"
]
}

💡 Tips for Implementation

- Use LangChain or LlamaIndex to compose the chains
- Store intermediate outputs (ToC, chunks, stats) in structured files (JSONL)
- Use a validation harness with golden ToC and document structure
- Visualize the ToC vs actual parse tree using Graphviz or D3

Implementation Steps

To implement logical parsing, chunking, and ToC validation for USB Power Delivery specifications using Python, we can follow a modular, library-driven development plan. Below is a breakdown of:

- 📦 Python libraries to use
- 🛠️ Implementation steps
- 🧩 Sample YAML schema (for ToC & chunked output)

📌 Goal: Build a pipeline that processes USB PD specs (PDF), extracts and logically chunks content, parses Table of Contents (ToC), and performs validation with structured outputs.

—

🧰 Recommended Python Libraries

Category	Library	Purpose
PDF parsing	pdfplumber, PyMuPDF (fitz), pdfminer.six	Extract raw text and layout

Chunking	regex, nltk, spacy	Tokenization, sentence/heading detection
Semantic validation	difflib, Levenshtein	Matching ToC ↔ parsed sections
Data modeling	pydantic, ruamel.yaml	Define and validate schemas
Embedding/search	sentence-transformers, FAISS	Optional for retrieval
Graph building	networkx, rdflib, graphviz	Optional for knowledge graphs
Multi-step pipelines	langchain, llama-index	Optional for orchestration

—

🛠️ Implementation Steps

Step 0: Prepare Test PDFs

- Get a real USB PD spec or redacted example
- Validate page count and ToC range (usually p. 1–10)

Step 1: Parse and extract Table of Contents (ToC)

- Use pdfplumber or PyMuPDF to extract front matter pages
- Apply regex patterns to parse heading lines (e.g. r'^(\d+(.\d+)*).+.{3,}\s+(\d+)$')
- Normalize into hierarchical format: chapter → section → subsection

Example regex for parsing ToC:

r"^(\d+(.\d+)*)(.?).{3,}\s+(\d+)$"

Step 2: Parse full text and chunk logically

- Use pdfplumber to extract all page text
- Detect headings using numbered patterns: 2, 2.1, 2.1.1
- Create a heading stack to nest content (tree-like chunking)
- Group paragraphs, tables, and figures under nearest heading

Step 3: Align chunks with ToC entries

- Compare chunked headings to parsed ToC using:
    - String match (normalized)
    - Fuzzy match (Levenshtein ratio > 0.9)
- Identify missing or extra sections

Step 4: Generate validation report

- Count & report:
    - Total sections in ToC vs parsed
    - Mismatches, order errors, gaps
- Output in structured JSONL format

Step 5: Build knowledge graph

- Add networkx or rdflib and create a hierarchical knowledge graph

Optional Steps:

- Use langchain if integrating with LLM-based tools

JSONL schema and a sample file

Refer below structure for a USB PD specification document — based on your provided Table of Contents.

This schema is optimized for:

- Retrieval, search, and hierarchical understanding
- Easy ingestion into vector stores or LLM-based document agents
- Consistency across all chapters, sections, and subsections

✅ JSONL Format Overview (Each line = one entry/section)

Each line (object) includes:

Field	Type	Description
section_id	string	Hierarchical section identifier (e.g., "2.1.2")
title	string	Section title (without numbering)
page	integer	Starting page number of the section
level	integer	Depth level (chapter = 1, section = 2, etc.)
parent_id	string/null	Immediate parent section (null for top level)
full_path	string	Concatenation of section_id and title (e.g., "2.1.2 Power Delivery Contract Negotiation")
doc_title	string	Document name or version for reference
tags	list	Optional: semantic labels (e.g., ["negotiation", "contracts"])

—

📄 Sample JSONL File (usb_pd_spec.jsonl)

{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2", "title": "Overview", "full_path": "2 Overview", "page": 53, "level": 1, "parent_id": null, "tags": []}

{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.1", "title": "Introduction", "full_path": "2.1 Introduction", "page": 53, "level": 2, "parent_id": "2", "tags": []}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.1.1", "title": "Power Delivery Source Operational Contracts", "full_path": "2.1.1 Power Delivery Source Operational Contracts", "page": 53, "level": 3, "parent_id": "2.1", "tags": ["contracts", "source"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.1.2", "title": "Power Delivery Contract Negotiation", "full_path": "2.1.2 Power Delivery Contract Negotiation", "page": 53, "level": 3, "parent_id": "2.1", "tags": ["contracts", "negotiation"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.1.3", "title": "Other Uses for Power Delivery", "full_path": "2.1.3 Other Uses for Power Delivery", "page": 54, "level": 3, "parent_id": "2.1", "tags": ["applications"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.2", "title": "Compatibility with Revision 2.0", "full_path": "2.2 Compatibility with Revision 2.0", "page": 54, "level": 2, "parent_id": "2", "tags": ["revision"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.3", "title": "USB Power Delivery Capable Devices", "full_path": "2.3 USB Power Delivery Capable Devices", "page": 55, "level": 2, "parent_id": "2", "tags": ["devices"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.4", "title": "SOP* Communication", "full_path": "2.4 SOP* Communication", "page": 57, "level": 2, "parent_id": "2", "tags": ["communication"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.4.1", "title": "Introduction", "full_path": "2.4.1 Introduction", "page": 57, "level": 3, "parent_id": "2.4", "tags": []}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.4.2", "title": "SOP* Collision Avoidance", "full_path": "2.4.2 SOP* Collision Avoidance", "page": 57, "level": 3, "parent_id": "2.4", "tags": ["collision", "avoidance"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.4.3", "title": "SOP Communication", "full_path": "2.4.3 SOP Communication", "page": 57, "level": 3, "parent_id": "2.4", "tags": ["communication"]}
{"doc_title": "USB Power Delivery Specification Rev X", "section_id": "2.4.4", "title": "SOP'/SOP" Communication with Cable Plugs", "full_path": "2.4.4 SOP'/SOP" Communication with Cable Plugs", "page": 57, "level": 3, "parent_id": "2.4", "tags": ["cable", "SOP'"]}

—

✅ Tips for Building This from PDF

- Extract text via pdfplumber or fitz
- Use regex to extract section IDs and titles:

Example regex:
^(\d+(.\d+)*)(\s+)([^\n.]+)(.+)\s+(\d+)$

- Use number of dots in section_id to infer level
- Track parent_id via section_id truncation logic:
    - 2.1.2 → parent = 2.1
    - 2 → parent = null