Implement a PDF parsing pipeline and search system architecture

Overview

In this paid task, you will implement a robust PDF parsing system that can handle heterogeneous financial documents from stock exchanges, and design a search system architecture that enables natural language querying of the parsed data.

This task demonstrates your ability to:

1. Build flexible data processing pipelines that handle varied input formats, and;
2. Design scalable search architectures for unstructured financial data.

This task involves PDF processing, data schema design, and system architecture.

It should take about 8-12 hours to complete.

Context

Stock exchanges worldwide publish various types of financial reports as PDF documents. These include bulk/block deal reports, board meeting announcements, shareholding patterns, regulatory circulars, and other compliance documents. At our organisation, we collect these documents to provide insights for financial analysis and regulatory monitoring.

The challenge lies in the heterogeneous nature of these documents — whereas data feeds have standardised formats, the structure of such PDFs varies considerably. For example:

- **Tabular reports** such as bulk deal summaries with structured data in rows and columns
- **Semi-structured announcements** like board meeting outcomes with mixed text and metadata
- **Hybrid documents** containing both narrative text and embedded tables
- **Multi-language content** including documents in Chinese and other languages

Our goal is to extract meaningful, structured data from these varied formats and make it searchable through natural language queries.

Definition: Document Types

For this challenge, we focus on three primary document types commonly found in stock exchange publications:

- **bulk_deal**: Reports containing details of large-volume transactions, typically presented in tabular format with fields such as security name, client details, transaction type, quantity, and value.
- **board_meeting**: Announcements regarding board meetings, resolutions, and corporate actions, usually presented as semi-structured text with key-value pairs embedded in narrative format.
- **shareholding_pattern**: Reports showing ownership distribution and changes in shareholding, which may contain both summary statistics and detailed tabular data.

Each document type requires different parsing strategies and output schemas, but all should be processable through a unified pipeline.

Objective

The objective of this challenge is to build a comprehensive PDF processing and search system that demonstrates your ability to:

1. **Implement a flexible PDF parsing pipeline** that can handle multiple document types and formats with appropriate error handling and extensibility.
2. **Design robust data schemas** that can accommodate heterogeneous document structures while maintaining data integrity and queryability.
3. **Create a scalable system architecture** for natural language search over parsed financial documents.
4. **Handle real-world challenges** including multi-language content, varying layouts, and missing or malformed data.

Note: This task emphasises both practical implementation and system design thinking. Success means building a working parser that handles the provided documents whilst designing a search architecture that could scale to production requirements.

Getting Started

Prerequisites

Before starting the task, please ensure you have the following installed:

- Python 3.8 or higher
- pip or poetry for package management
- A PDF processing library of your choice (e.g., PyMuPDF, pdfplumber, camelot-py)
- Any additional dependencies your solution requires

Dev environment Setup

We have provided you with:

Sample Documents

- A CSV file (`documents.csv`) containing URLs to various PDF documents from Chinese stock exchanges
- These documents represent real-world examples of the heterogeneous formats you'll encounter

Expected Output Format

- JSON schema examples showing the expected structure for different document types
- Guidelines for handling edge cases and missing data

Sample Data Structure

Your input will be a CSV file with the following structure:

```
url
<https://example.com/sse_bulk_deals.pdf>
<https://example.com/sse_board_meeting.pdf>
<https://example.com/shareholding_pattern.pdf>
```

Important Note: The provided URLs point to documents in Chinese. Your parser should handle multi-language content gracefully, though full translation is not required.

Technical specs

Part 1: PDF parsing pipeline

Your parsing pipeline should implement the following workflow:

```
graph TD
   LoadURLs[Load Document URLs]
   --> DownloadPDFs[Download PDFs]
   --> DetectType[Detect Document Type]
   --> ParseContent[Parse Content]
   --> ValidateData[Validate & Clean Data]
   --> OutputJSON[Generate JSON Output]
```

Document download

- Download PDFs from the provided URLs
- Implement appropriate timeout and retry logic
- Handle network errors gracefully
- Store downloaded files with meaningful names

Document type detection

Your system should automatically identify document types based on content patterns. This might involve:

- Analysing document structure (table presence, text patterns)
- Examining metadata and headers
- Using keyword matching or simple heuristics
- Defaulting to a generic type when classification is uncertain

Content extraction

Implement specialised parsers for each document type:

For Tabular Documents (Bulk Deals):

- Extract table structures preserving column relationships
- Handle merged cells and complex layouts
- Validate numerical data and format consistency
- Map columns to standardised field names

For Semi-Structured Text (Board Meetings):

- Extract key-value pairs from narrative text

- Identify dates, company names, and meeting purposes
- Handle formatted lists and bullet points
- Preserve important textual context

For Hybrid Documents:

- Combine table extraction with text parsing
- Maintain relationships between different data sections
- Handle documents with multiple tables or text blocks

Data Validation and Cleaning

- Implement field-level validation (e.g., date formats, numerical ranges)
- Handle missing or malformed data gracefully
- Log data quality issues for review
- Apply consistent formatting and normalisation

Part 2: Output Schema Design

Design a flexible JSON schema that can accommodate all document types whilst remaining queryable and extensible.

Example Output Structures

Bulk Deal Report:

```
{
  "doc_type": "bulk_deal",
  "source_url": "<https://example.com/sse_bulk_deals.pdf>",
  "extraction_timestamp": "2025-01-15T10:30:00Z",
  "metadata": {
    "report_date": "2025-09-20",
    "exchange": "SSE",
    "currency": "CNY",
    "language": "zh-CN"
  },
  "records": [
    {
      "security_name": "华发股份",
      "security_code": "600325",
      "client_name": "ABC Capital Management",
      "transaction_type": "BUY",
      "quantity": 1000000,
      "price": 25.50,
      "value": 25500000.0,
      "transaction_date": "2025-09-20"
    }
  ],
  "data_quality": {
    "completeness_score": 0.95,
    "validation_warnings": []
  }
}
```

Board Meeting Announcement:

```
{
  "doc_type": "board_meeting",
  "source_url": "<https://example.com/board_meeting.pdf>",
  "extraction_timestamp": "2025-01-15T10:30:00Z",
  "metadata": {
    "company": "Tellus Holdings",
    "company_code": "002345",
    "announcement_date": "2025-09-20",
    "exchange": "SZSE",
    "language": "zh-CN"
  },
  "details": {
    "meeting_date": "2025-10-05",
    "meeting_type": "Board Meeting",
    "purpose": "Consideration of quarterly results and dividend declaration",
    "resolutions": [
      {
        "resolution_number": "2025-001",
        "description": "Approval of Q3 financial statements",
        "status": "Approved"
      }
    ]
  },
  "data_quality": {
    "completeness_score": 0.88,
    "validation_warnings": ["Meeting venue not specified"]
  }
}
```

Part 3: Search System Architecture (Design Only)

Design a comprehensive search system that enables users to query the parsed financial data using natural language. **Note: Implementation is not required—provide a detailed architecture diagram and explanation.**

Architecture Deliverable

Provide a detailed diagram (using tools like draw.io, Lucidchart, or similar) showing:

- System components and their interactions
- Data flow from ingestion to query results
- Technology stack recommendations
- Scaling and deployment strategies

Include a written explanation (500-1000 words) covering your design decisions, trade-offs, and rationale.

Implementation Requirements

Core Functionality

Your implementation must include:

Command Line Interface

```
python parse_pdfs.py --urls documents.csv --output parsed_results.json
```

1.
2. **Document Processing**
    - Download PDFs from provided URLs
    - Detect document types automatically
    - Extract structured data using appropriate methods
    - Generate JSON output in the specified format
3. **Error Handling**
    - Graceful handling of network failures during download

    - Robust parsing that continues even if some fields are missing
    - Comprehensive logging of errors and warnings
    - Validation of extracted data with quality scores

4. **Code Structure**

    - Modular design with clear separation of concerns
    - Abstract base classes for different parser types
    - Extensible architecture for adding new document types
    - Configuration management for parser settings

Implementation requirements

---

1. **Code Quality**
    - Follow Python best practices and PEP 8 style guidelines
    - Include comprehensive docstrings and type hints
    - Implement proper exception handling throughout
    - Write clean, maintainable code with meaningful variable names
2. **Testing**
    - Unit tests for core parsing functions
    - Integration tests with sample PDF files
    - Error condition testing (malformed PDFs, network issues)
    - Data validation testing for output schemas
3. **Documentation**
    - README with setup and usage instructions
    - API documentation for public functions
    - Architecture overview explaining design decisions
    - Known limitations and future improvement suggestions
4. **Performance Considerations**
    - Efficient memory usage when processing large PDFs
    - Reasonable processing time for typical documents
    - Appropriate use of concurrency for downloading multiple files
    - Caching strategies to avoid re-processing unchanged documents

Extensions (Optional)

These are bonus features that will earn additional credit:

1. **Database Integration**
    - Store parsed results in PostgreSQL or SQLite
    - Design appropriate table schemas for different document types
    - Implement upsert logic to handle document updates
2. **API Service**

    - Build a REST API with endpoints like `POST /parse` and `GET /documents`
    - Include Swagger/OpenAPI documentation
    - Implement authentication and rate limiting
3. **Advanced Processing**
    - OCR support for scanned documents
    - Table detection and extraction from complex layouts
    - Support for additional document types beyond the core three
4. **Monitoring and Observability**
    - Structured logging with appropriate levels
    - Metrics collection for processing times and success rates
    - Health check endpoints for service monitoring

Your deliverables

Please provide the following deliverables:

1. Source Code

- Complete implementation of the PDF parsing pipeline
- Command-line interface as specified
- All supporting modules and configuration files
- Requirements file with dependencies

2. Architecture Documentation

- Detailed search system architecture diagram
- Written explanation of design decisions and trade-offs
- Technology stack recommendations with justifications
- Scaling strategy and deployment considerations

3. Testing and Quality Assurance

- Unit tests covering core functionality
- Integration tests with sample documents
- Test data and expected outputs for validation
- Code coverage report (if available)

4. Documentation

- README with setup and usage instructions
- API documentation for public functions

- Known limitations and assumptions
- Future improvement roadmap

5. Sample Outputs

- Processed JSON files from the provided document URLs
- Log files demonstrating error handling
- Performance metrics (processing times, success rates)

Our evaluation criteria

Your solution will be evaluated on the following criteria:

1. Functionality

- Does the parser successfully extract data from the provided PDF documents?
- Are the JSON outputs correctly formatted and complete?
- Does the CLI work as specified?

2. Code Quality

- Is the code well-structured, readable, and maintainable?
- Are best practices followed for Python development?
- Is the code properly documented with clear explanations?

3. Architecture and Design

- Is the parser architecture extensible for new document types?
- Are appropriate abstractions used to handle different PDF formats?
- Is the search system architecture well-designed and scalable?

4. Error Handling and Robustness

- Does the system handle edge cases and errors gracefully?
- Are appropriate fallback strategies implemented?
- Is logging comprehensive enough for debugging issues?

5. System Thinking

- Does the search architecture demonstrate understanding of large-scale systems?
- Are scalability and performance considerations properly addressed?

- Are the technology choices well-justified?

Submission Protocol

Submit your code in a GitHub repository. All text documentation should be in markdown format and included in the repo. A complete solution will include at least the following:

1. All source code files.
2. Docker configuration files.
3. Documentation files.
4. Sample output files.

We believe the above instructions are reasonably clear. However, there may remain some ambiguities. If so, it is your responsibility to resolve these yourself, without further input from us. One exception: Please contact us immediately if you are unable to access the code or data necessary to begin the task, and we will fix this promptly.

</aside>

We wish you the best of luck with this challenge!