Contents

Introduction

For this task, I leveraged both ChatGPT-4.1 and Claude Sonnet 4 as research and development tools to prototype code, conduct research, compare approaches, and generate supporting materials such as app.diagrams.net figures.

While AI models aided with brainstorming and evaluating technical alternatives, the final code and implementation reflect my own engineering practices and standards. I independently reviewed and, where time permitted, verified the accuracy and feasibility of AI recommendations before incorporating them.

For example, based on suggestions from both models, I explored several OCR solutions, including TrOCR and PaddleOCR. I determined that TrOCR was not immediately suitable for processing image formats extracted from PDFs, and PaddleOCR could not be fully evaluated within the project timeframe. Ultimately, I selected ocrmypdf, which offered the most robust and practical solution given the project's requirements and time constraints despite limitations in handwriting OCR support.

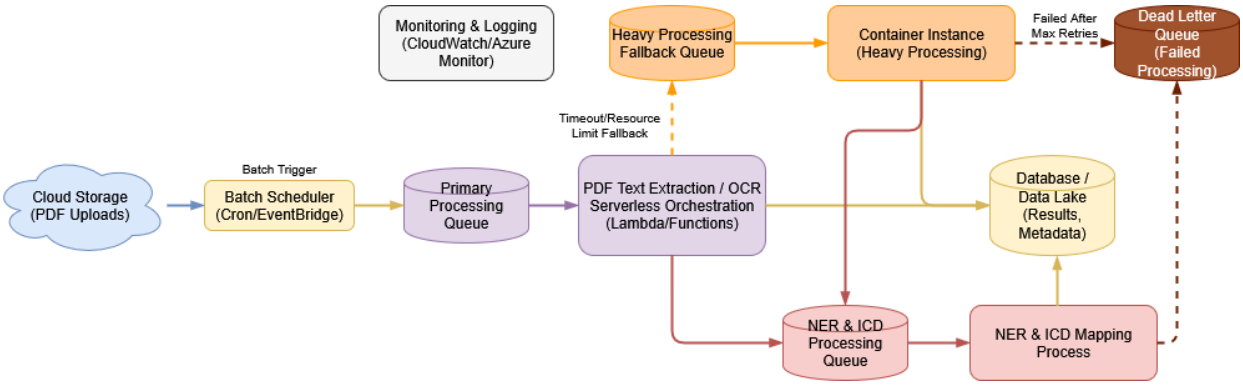
In addition to AI-based guidance, I consulted primary sources such as tool documentation, man pages, and the UMLS knowledge base to inform design, implementation, or to fetch resources. The code subfolder in this submission contains my initial implementation of the extraction and ICD coding workflow.

Development and testing were performed on a Windows laptop with 16 GB of RAM using Ubuntu via WSL2. I used VS Code with Copilot and other integrations as my IDE. The proof of concept was also successfully built and tested in a local Docker Desktop environment.

I appreciate the opportunity to tackle this challenge and look forward to any feedback on my approach.

Question 1: How will you design an automated workflow to achieve your goal?

To automate extraction and ICD coding of disease-related descriptions from clinical PDFs, I leveraged ChatGPT 4.1 and Claude Sonnet 4 to design a cloud-native, queue-driven processing pipeline that is robust, scalable, and modular. This architecture is compatible with both AWS and Azure and is designed to handle a wide range of workloads and operational scenarios.



Workflow Overview

1. PDF Ingestion and Batch Scheduling
    - PDF Upload: Documents are uploaded to secure cloud storage *(AWS: S3, Azure: Blob Storage)*
    - Batch/Event Trigger: New uploads are detected, and jobs are then scheduled *(AWS: EventBridge scheduled rule, Azure: Logic Apps recurrence or Functions Timer Trigger)*
    - Jobs are added to a primary processing queue *(AWS: SQS, Azure: Service Bus or Storage Queues)*
2. Queue-Driven Processing Workers
    - Stateless workers consume jobs from the primary queue and attempt to process each PDF using standard serverless compute *(AWS: Lambda, Azure: Functions)*
    - If a job fails due to resource constraints (e.g., timeouts, memory errors), it is automatically retried or redirected to a heavy processing queue for escalation
3. Text Extraction and OCR

- Standard PDFs: For most documents, workers extract selectable text and apply OCR (e.g., ocrmypdf, Tesseract) to image-based content
- Heavy/Complex PDFs: Containerized workers process jobs from the heavy processing queue, providing greater resources for large, complex, or problematic files
  (AWS: ECS/Fargate/EKS, Azure: Container Instances/AKS)
- Jobs that fail after multiple attempts are routed to a dead letter queue for error tracking and manual intervention
  (AWS: SQS DLQ, Azure: Service Bus DLQ)

4. NER and ICD Mapping Processing
   - After text extraction, jobs are placed in a dedicated NER & ICD Processing Queue, which decouples this compute intensive step from the primary serverless orchestration layer.
   - The queue triggers serverless functions or containers that use spaCy-based biomedical NER and dictionary-based matching to identify disease mentions and map them to ICD codes.
   - For this proof of concept, the dictionaries were derived from UMLS resources. ICD-10-CM datasets (available from the CDC FTP site) were discovered near the end of development; with more time, I would integrate these directly for improved accuracy and coverage.
   - These serverless functions or containers are designed as stateless components and can be deployed flexibly across AWS Lambda, ECS/Fargate, Azure Functions, Container Instances, or AKS, though they require substantial RAM for both serverless and container architectures.
5. Result Storage and Monitoring
   - Extracted disease mentions, ICD codes, and metadata are written to a managed database or data lake
     (AWS: RDS, DynamoDB, S3, Redshift; Azure: SQL DB, Cosmos DB, Synapse, Data Lake Storage Gen2)
   - Every stage is integrated with cloud-native monitoring and logging solutions
     (AWS: CloudWatch, Azure: Monitor/Log Analytics)

Key Advantages

- Scalability: Each pipeline stage can scale horizontally, processing many documents in parallel.
- Fault Tolerance: Queues, retries, and dead-letter queues provide robust error and failure isolation.
- Modularity: Each component (upload, extraction, mapping, storage) can be updated, replaced, or scaled independently.
- Compliance & Auditability: Full traceability, audit logs, and cloud-native security features ensure data integrity and regulatory compliance.

The modular, queue-driven design supports potential further decomposition into specialized microservices. As use cases grow, additional pipeline stages can be introduced. For example, text extraction and image extraction can be decoupled into distinct steps, OCR can be applied on a per-image basis, and NER/ICD mapping can be performed independently on individual document chunks. Each of these components can be implemented as separate, modular pipeline stages or microservices to optimize parallelism and maintainability. This approach enables targeted scaling, easier maintenance, and seamless integration of advanced processing techniques or models in the future.

Question 2: Any major modules/functions that are essential for your workflow?

The workflow has been implemented as a functional Python package (pdf2icd) with Poetry-based dependency management, type hints, unit tests, and a Dockerfile for containerization. A single pipeline covering text extraction, OCR, disease NER, disease matching, and ICD mapping is presented in workflow.py.

Core Processing Modules

1. poppler.py: Extracts embedded text from PDFs using pdftotext and identifies image-containing pages using pdfimages.
2. ocr.py: Runs OCR on image-based pages using ocrmypdf, returning clean sidecar text.
3. disease_ner.py: Uses a spaCy biomedical NER model to extract unique disease mentions from both raw and normalized text.
4. disease_matcher.py: Performs exact and fuzzy dictionary-based matching to resolve disease mentions to UMLS CUIs and ICD codes.
5. prepare_assets.py: Builds the term_to_cuis.json and cui_to_icd.json lookup assets from UMLS RRF files needed in disease_matcher.py.

6. utils.py: Provides text normalization, Unicode cleaning, and functions to load the prebuilt mappings.
7. workflow.py: Coordinates the entire pipeline: text extraction (embedded and OCR), NER, ICD mapping, and TSV output.

Cloud Integration Layer
The proof of concept is structured to support future extension with AWS or Azure SDKs (for example, boto3 or azure-storage-blob) for document storage, event triggers, queue processing, and secure result storage.

Justification, Alternatives Considered, and Limitations

- Text extraction: Poppler utilities were chosen for reliability and permissive licensing. Libraries like PyMuPDF were avoided due to more restrictive terms.
- OCR: ocrmypdf integrates cleanly with the pipeline. Alternatives such as TrOCR and PaddleOCR were explored but required more adaptation. Cloud OCR APIs (AWS Textract, Azure Computer Vision) remain future options for messy handwriting.
- NER and ICD mapping: The proof of concept uses a spaCy NER model (en_ner_bc5cdr_md) plus dictionary-based mapping to balance computational requirements as the spaCy UMLS linker was too resource intensive.
- Assets: The pipeline currently relies on UMLS-derived dictionaries. ICD-10-CM datasets (from the CDC FTP site) were identified late in development and could be integrated in a future iteration for improved coverage.
- Licensing: Tools were chosen for their permissive or widely accepted licenses.

Potential Limitations

- Reduced OCR accuracy on poor-quality scans or handwritten notes.
- Mapping quality depends on the completeness and correctness of the UMLS/ICD assets.
- Increased memory requirements when loading large models, making queue-driven asynchronous processing preferable for cloud deployments.

Question 3: Any challenges and specific considerations you need to pay attention to when designing the workflow?

Designing an automated workflow for disease extraction and ICD coding from clinical PDFs presents several technical and operational challenges. The following

considerations were identified as critical to ensure robust performance, maintainability, and compliance in both proof-of-concept and potential production environments:

- Document Diversity: Clinical PDFs vary widely in formatting, scan quality, and layout. The workflow needs to handle both clean digital text and poor-quality scanned images, potentially within the same document.
- OCR Accuracy Issues: OCR performance drops significantly with handwritten notes, poor scans, or complex layouts. The system needs reliable error detection and clear fallback strategies when OCR fails completely.
- Resource and Timeout Limits: Large or complex documents can exceed memory limits or processing timeouts in serverless environments. The proposed architecture handles this by routing such documents to container instances with higher resource limits. For extremely large documents, processing may potentially still need to be broken into smaller steps (page-by-page OCR, text chunking).
- Medical Entity Recognition Challenges: Disease mentions can appear as abbreviations, synonyms, misspellings, or context-dependent terms. While the implemented NER pipeline combines spaCy-based extraction with normalization and fuzzy matching to mitigate this, domain-specific or novel terms may still be missed.
- ICD Mapping Complexity: Multiple diseases can map to the same ICD code, and single diseases may have multiple valid codes depending on specificity. Real-world clinical usage may require nuanced mapping decisions.
- Data Privacy and Security: Clinical documents may contain PHI, requiring encryption at rest and in transit, secure processing environments, and audit trails for compliance (e.g., HIPAA).
- Error Handling and Debugging: Comprehensive logging and error tracking are needed to identify which documents failed and why, especially critical for clinical applications that require full auditability and traceability.
- Tool Licensing: Some powerful libraries may have restrictive licenses that limit commercial or clinical use. Favoring permissive licenses reduces deployment barriers but may limit available functionality.
- Cloud Integration and Scalability: The workflow should be modular and stateless where possible, making it easier to distribute across cloud infrastructure, scale horizontally, and recover from failures.

Question 4: List the key performance metrics you plan to use to evaluate your workflow

To evaluate the effectiveness and reliability of the workflow, several key performance metrics will be tracked throughout development and testing. These metrics address both the clinical accuracy of disease extraction and coding, as well as the operational performance and robustness of the system. The most important metrics include:

- Extraction Accuracy:
  Percentage of disease mentions and ICD codes correctly identified, measured against a gold standard (manually annotated) dataset.
- Precision:
  Of all disease mentions/ICD codes identified by the workflow, the percentage that are actually correct.
- Recall:
  Of all the true disease mentions/ICD codes present in the gold standard, the percentage that the workflow correctly identified.
- F1 Score:
  Harmonic mean of precision and recall, providing a single summary of extraction/mapping quality.
- OCR/Text Extraction Coverage:
  Percentage of pages or documents where text is successfully extracted, either digitally or via OCR.
- Processing Throughput:
  Number of documents processed per hour or per day.
- Processing Time per Document:
  Average time required to process a document end-to-end, from ingestion to output.
- OCR Success Rate:
  Percentage of pages or documents where OCR completes without errors.
- Error/Failure Rate:
  Proportion of documents that fail at any stage (OCR, extraction, mapping), tracked by logs and error reports.
- Manual Review Rate:
  Percentage of cases requiring human intervention due to errors or uncertainty.

- Unit Test Coverage (pytest):
  Portion of code covered by automated unit tests, ensuring reliability during development.
- Resource Utilization and Scalability:
  Ability to process documents of varying sizes and types without hitting memory, timeout, or scaling limits (especially important for cloud/serverless environments).