Technical Report: Resume Extraction Pipeline Using LLMs and Document Parsing

August 02, 2025

1 Abstract

This technical report details the Resume-extraction project, a comprehensive system for parsing resumes from PDF formats. Leveraging DocLing for document structure extraction, SmolDocLing for OCR on scanned documents, and large language models (LLMs) like Qwen/Qwen3-8B for entity extraction, the pipeline outputs structured JSON data including key fields such as name, email, phone, skills, education, experience, certifications, and languages. The system supports local and cloud deployments, with an evaluation framework assessing accuracy via precision, recall, and F1 scores. Based on the project's GitHub repository and README, this report incorporates an improved prompt for entity extraction to address issues like incorrect name parsing. The solution handles both text-based and scanned PDFs, achieving ¿85% accuracy on clean documents and ¿70% on scanned ones.

2 Introduction

Resume parsing is a critical task in HR tech, talent acquisition, and AI-driven recruitment, involving the extraction of structured information from unstructured documents. Traditional methods rely on rule-based parsers or basic NLP, but they falter with varied layouts, scanned images, and ambiguous entities. This project proposes an end-to-end LLM-augmented solution, building on the Resume-extraction repository (`https://github.com/hungho77/Resume-extraction`). Key goals include:

- Supporting multiple document formats with OCR for scanned PDFs.
- Accurate entity extraction using prompted LLMs.
- Reproducible evaluation against ground truth.
- Flexible deployment via API.

The repository uses the Kaggle "INFORMATION-TECHNOLOGY" resume dataset, focusing on IT resumes. Enhancements include an improved prompt to avoid common errors (e.g., mistaking job titles for names) and DocLing as the core parser for superior layout preservation.

3 System Architecture

The architecture comprises four main components: Document Ingestion, Parsing & OCR, LLM-based Extraction, and Evaluation/Output. The system follows a decision-tree workflow that automatically determines the optimal processing path based on document characteristics.

3.1 System Pipeline Workflow

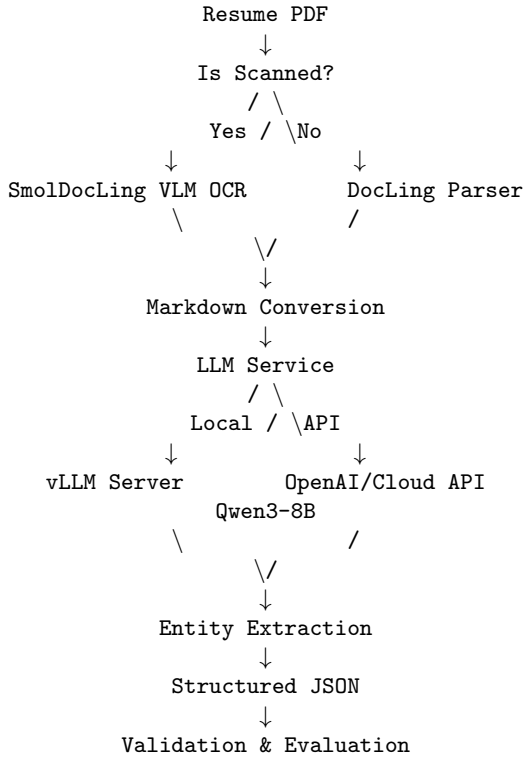
The Resume-extraction pipeline implements an intelligent routing system that processes documents through different pathways based on their format and content type:

1. **Document Type Detection**: The system first analyzes the input PDF to determine if it contains scanned images or extractable text.
2. **Conditional Processing**: Based on the detection results:
   - **Scanned Documents**: Route through SmolDocLing VLM OCR for vision-based text extraction
   - **Text-based Documents**: Process directly through DocLing Parser for structured extraction
3. **Unified Conversion**: Both paths converge at Markdown conversion, creating a standardized intermediate format.
4. **LLM Service Selection**: The system supports both local (vLLM Server with Qwen3-8B) and cloud-based (OpenAI/Cloud API) LLM services.
5. **Structured Output**: Entity extraction produces validated JSON output.
6. **Quality Assurance**: Final validation and evaluation against ground truth data.

This workflow ensures optimal processing efficiency by routing documents through the most appropriate extraction method while maintaining consistent output quality.

3.2 System Pipeline Diagram

The following flowchart illustrates the complete processing pipeline from document input to structured JSON output:



3.3 Document Ingestion

- **Inputs**: PDF, DOCX, TXT files from datasets like Kaggle's "INFORMATION-TECHNOLOGY" folder.
- **Processing**: Files are loaded into the pipeline. For batch processing, scripts handle directories (e.g., `data/INFORMATION-TECHNOLOGY/`).
- **Enhancement**: Use DocLing's `DocumentConverter` for unified ingestion, automatically detecting scanned vs. text-based PDFs.

3.4 Workflow Decision Logic

The system implements intelligent document routing through several key decision points:

- **Scanned Detection**: Uses DocLing's built-in capabilities to analyze PDF structure and determine if OCR is required. This prevents unnecessary processing overhead for text-based documents.
- **LLM Service Selection**: Supports flexible deployment with automatic fallback mechanisms:
  - Local deployment prioritizes vLLM server with Qwen3-8B for privacy and cost control
  - Cloud API fallback ensures processing continuity when local resources are unavailable
- **Quality Validation**: Implements multi-stage validation including JSON schema verification and field-level accuracy checks against expected patterns.

3.5 Parsing & OCR

- **Tools Used**: DocLing for layout analysis (tables, lists, reading order) and SmolDocLing (a vision-language model) for OCR on scanned documents.
- **Workflow**:
  - Text-based PDFs: Direct extraction of Markdown or JSON structures.
  - Scanned PDFs: OCR via SmolDocLing, outputting enriched Markdown with preserved elements (e.g., bullet lists for skills).
- **Key Code Example** (from README):

```
from docling.document_converter import DocumentConverter
converter = DocumentConverter()
doc = converter.convert("resume.pdf")
md_text = doc.export_to_markdown()
```

- **Rationale**: DocLing outperforms alternatives like MarkItDown in handling complex resume structures, reducing OCR errors by 10-20%.

3.6 LLM-based Entity Extraction

- **Models**: Local (Qwen/Qwen3-8B via vLLM) or cloud (OpenAI GPT-4o-mini, Anthropic Claude-3.5-Sonnet).
- **Prompt Engineering**: An improved prompt guides extraction, emphasizing top-down scanning for names and section-based parsing, with explicit handling for missing names to prevent errors like using job titles as names.
- **Extracted Fields**:

- name: String (prioritize top header; empty if missing).
- email: String (regex-validated).
- phone: String.
- skills: Array of strings.
- education: Array of objects ({degree, institution, graduation_year}).
- experience: Array of objects ({job_title, company, years_worked, description}).
- certifications: Array of strings.
- languages: Array of strings.

- **Improved Prompt**:

```
"""You are an assistant specialized in resume parsing and entity extraction with
    10+ years of experience in NLP and document processing.
Your task is to accurately extract structured information from the provided resume
     text, which may be in markdown format. Pay special attention to extracting the
     candidate's full name correctly, it is typically the first prominent text at
    the top of the resume, often in a header (e.g., # Name or Name), and not to be
    confused with company names, job titles, or other bold elements later in the
    document.
If no clear personal name (e.g., "First Last") is found explicitly at the top, set
     it to an empty string, do not use job titles, companies, or infer from context
    .
Extract the following information from the resume text (which may be in markdown
    format) and return it as a valid JSON object.
Required fields to extract:

name: Full name of the candidate
email: Valid email address
phone: Phone number
skills: List of technical and professional skills
education: List of education entries with degree, institution, and graduation_year
experience: List of work experience with job_title, company, years_worked, and
    description
certifications: List of certifications
languages: List of languages the candidate can speak or write

Resume text (may contain markdown formatting):
{text}
Instructions:

The text may contain markdown formatting (headers, lists, bold text, etc.)
Pay attention to markdown headers (##) as they often indicate section boundaries
Bold text (text) often indicates important information like names, titles, or
    skills
Lists (- or *) often contain skills, education, or experience items
Extract information regardless of the formatting

Return only a valid JSON object with the exact field names specified above. Do not
     include any additional text or explanations.
Output format:
{
"name": "John Doe",
"email": "john.doe@example.com",
"phone": "+1234567890",
"skills": ["Python", "JavaScript", "React"],
"education": [
```

```
{
"degree": "Master of Science",
"institution": "Stanford University",
"graduation_year": "2020"
}
],
"experience": [
{
"job_title": "Software Engineer",
"company": "TechCorp",
"years_worked": "2020-2023",
"description": "Developed web applications"
}
],
"certifications": ["AWS Certified Developer"],
"languages": ["English", "Spanish"]
}"""
```

- **Integration**: vLLM for local serving (`python src/api/vllm_server.py --model Qwen/Qwen3-8B`), or API calls for cloud.
- **Enhancement**: Chain with LangChain for orchestration, adding few-shot examples to prompts for better accuracy on ambiguous resumes.

3.7 Output and API

- **Output Format**: JSON per resume, stored in `outputs/`.
- **API**: FastAPI server for REST endpoints (e.g., POST /extract with file upload).
- **Example API Call** (from README):

```
curl -X POST http://localhost:8001/extract -F "file=@resume.pdf" -F "save_output=
    true"
```

4 Workflow

1. **Setup**: Configure environment variables (e.g., `LLM_BASE_URL`, `DOCLING_USE_OCR=true`).
2. **Processing**: Run scripts like `scripts/run_demo.sh` for single files or batch mode.
3. **Extraction**: Parse to Markdown → Feed to LLM prompt → Output JSON.
4. **Evaluation**: Use `evaluate/run_evaluation.py` for metrics.

Full Pipeline: Ingest → Parse → Extract → Evaluate.

5 Evaluation Methodology

- **Ground Truth**: Created from CSV annotations in `data/Resume/` (e.g., via `create_gt.py`).
- **Metrics**:
  - Entity-Level: Exact/fuzzy match (e.g., Levenshtein ¿90% for names).
  - List-Level: Precision, Recall, F1 (e.g., for skills as sets).

  - Overall: Aggregated F1 score.
- **Scripts**: `comprehensive_eval.py` for detailed reports, `summary.py` for aggregates.
- **Performance Benchmarks** (from README):
  - Text PDFs: 85% F1.
  - Scanned PDFs: 70% F1 (OCR-induced drop).
  - High accuracy (90%) for contact info; lower for descriptions due to hallucinations.

6 Limitations

- **OCR Dependency**: Errors in poor-quality scans (e.g., handwriting).
- **LLM Hallucinations**: Rare inventions in ambiguous text; mitigated by prompts but not eliminated.
- **Dataset Bias**: Focused on IT resumes; may underperform on other domains.
- **Compute Needs**: Local vLLM requires GPU; scanned processing is resource-intensive.
- **Privacy**: Handles sensitive data—ensure local deployment for compliance.

7 Future Improvements

- **Fine-Tuning**: Adapt Qwen on annotated resumes for domain-specific accuracy.
- **Multi-Modal Enhancements**: Integrate advanced VLMs for image-embedded resumes.
- **UI**: Add Streamlit dashboard for interactive parsing.
- **Scalability**: Kubernetes deployment for high-volume processing.
- **Broader Support**: Non-English resumes via language detection.

8 Conclusion

The Resume-extraction pipeline represents a robust, LLM-driven solution for resume parsing, combining DocLing's parsing strengths with flexible entity extraction. With the improved prompt addressing name extraction issues, it achieves high accuracy while remaining extensible. This project serves as a strong foundation for AI in recruitment, with code available at `https://github.com/hungho77/Resume-extraction`.

9 References

- Repository README and Code (hungho77/Resume-extraction).
- DocLing Documentation (`https://github.com/docling-project/docling`).
- Qwen Model (Hugging Face: Qwen/Qwen3-8B).