SmartCV: A Four-Stage AI Pipeline for Company-Tailored Automated CV Standardization

Colin McLaughlin, Jaiman Sharma, Saanvi Sood, Aurik Disler, Kayla Burzese, Akash Singh
Queen's University
mclaughlin.colin@queensu.ca, 23wq@queensu.ca, saanvi.sood@queensu.ca,
24ll45@queensu.ca, 23sbg5@queensu.ca, akash.singh@queensu.ca

***Abstract*—Many consulting and engineering firms need resumes to be manually reformatted into standardized CV templates for project proposals. This process is time-consuming due to the wide variability in resume formats and structures. In collaboration with Technical Management Group Ltd. (TMG), we developed SmartCV, an automated pipeline that converts heterogeneous resumes into standardized company-specific CV formats. The system combines deterministic document processing with AI-assisted structural inference and template-based rendering. Results show that SmartCV reliably produces validated outputs with low operational cost while remaining flexible enough to support future templates and to be extended to other organizations facing similar document standardization challenges.**

I. Introduction

Organizations that prepare client proposals and consulting submissions frequently require contractor CVs to be presented in a standardized format. However, resumes are typically submitted in heterogeneous layouts with inconsistent section structures, formatting styles, and bullet hierarchies. As a result, significant manual effort is required to extract relevant information and reformat it into a company-specific template.

A. Motivation

This challenge was presented to our team by Technical Management Group Ltd. (TMG), a project management and consulting firm operating in the mining, energy, and construction sectors. TMG regularly prepares technical proposals that include standardized contractor CVs. According to the company's internal workflow, formatting a single CV manually can take approximately one hour, while macro-assisted formatting reduces the task to roughly twenty minutes per CV. The organization's goal is to reduce this process to a largely automated workflow requiring only brief manual proofreading and small edits, ideally taking only a few minutes per document.

Automating this process presents several challenges. CVs contain highly variable document structures, inconsistent section naming, and mixed formatting conventions across formats such as DOCX and PDF. Any automated system must therefore be able to robustly extract information from heterogeneous documents while producing a final output that conforms precisely to a predefined corporate template.

B. Related Works

Automated resume parsing has been widely studied as a document information extraction task, where unstructured CVs are converted into structured representations of entities such as education, work experience, and skills. Early systems commonly relied on rule-based or traditional natural language processing techniques to extract these fields [1]. More recent work has explored machine learning approaches that frame resume parsing as a structured prediction problem, using sequence labeling models to classify tokens or text spans into semantic categories [2].

Research in document understanding has also introduced models that incorporate document layout information. LayoutLM and related architectures jointly model textual and spatial features to improve understanding of visually structured documents [3]. More recently, large language models have been applied to document reasoning tasks, demonstrating the ability to infer structure and generate structured outputs from heterogeneous document inputs [4].

SmartCV builds on these developments by applying a schema-constrained large language model within a deterministic processing pipeline to automatically standardize heterogeneous resumes into company-specific CV templates without requiring large labeled datasets.

C. Problem Definition

This work addresses the problem of automated CV standardization. Given an input resume document d in either DOCX or PDF format, the objective is to generate an output document T that conforms to a fixed company-specific template while preserving the factual content of the original CV.

The target output structure follows a predefined schema required by TMG, which includes sections such as Profile, Expertise and Results, Key Projects Summary, and Education and Qualifications.

The system must therefore perform three tasks simultaneously: extracting relevant content from heterogeneous documents, organizing that content into the required section schema, and rendering the final result in a standardized Word template.

To address these challenges, we propose SmartCV, a four-stage pipeline consisting of parsing, normalization, AI-assisted structuring, and template-based rendering. The system combines deterministic document processing with constrained AI assistance in order to balance structural reliability with the flexibility required to interpret diverse resume formats.

II. Methodology

The system is implemented as a four-stage processing pipeline consisting of parsing, normalization, AI-assisted structuring, and template rendering. Each stage operates on a clearly defined intermediate representation and produces structured output that becomes the input to the next stage. This staged architecture was chosen to separate low-level document processing tasks from higher-level semantic interpretation and formatting. As a result, each component of the pipeline can be tested, modified, or replaced independently without affecting the rest of the system.

A. Pipeline Overview

1) Parsing Layer: The parsing layer converts raw resume files into an internal representation suitable for downstream processing. The system supports both Microsoft Word (.docx) and PDF inputs using format-specific extraction methods. DOCX files are parsed using the python-docx library, which allows direct traversal of document elements such as paragraphs, bullets, headings, and table cells while preserving basic formatting metadata. For PDF files, the system uses PyMuPDF to extract text sequentially from the document, reconstructing content order based on the positional flow of text.

All extracted content is converted into a unified block-based representation, where each block corresponds to a contiguous unit of text such as a paragraph or bullet item. Resumes are treated as an ordered sequence of blocks rather than a hierarchical document structure, which helps preserve the original reading order and simplifies downstream processing. This approach is particularly useful given the high variability of resume formats, including inconsistent bullet encodings, layout differences between DOCX and PDF files, and irregular section structures.

2) Normalization Layer: In SmartCV, normalization is the process that cleans up raw parsed resume blocks to create a more uniform structure before semantic interpretation. After parsing, blocks may contain anomalies such as extra whitespace, tabs, foreign bullet markers, mixed dash symbols, and other artifacts from the extraction process. The normalization step addresses these issues by collapsing multiple spaces into one, removing leading and trailing whitespace and tabs, standardizing dashes to a hyphen, and unifying bullet formatting by detecting common bullet glyphs and rewriting them into a consistent style. This ensures that later stages of the parsing process do not misinterpret visually different but semantically identical list items.

Additionally, normalization removes irrelevant elements such as empty blocks, footer content, and page numbers. Header content is preserved only if it contains meaningful data, such as a candidate name or contact information, while decorative or noisy elements are excluded. SmartCV also consolidates blocks affected by layout fragmentation, merging wrapped bullet lines into a single item when a bullet fragment spills into the following line, and combining adjacent paragraph blocks if heuristics indicate that they are part of the same sentence or paragraph.

By minimizing noise, standardizing formatting, and repairing fragmented text, normalization prepares a cleaner and more structurally stable document for subsequent semantic and interpretive analysis, improving accuracy in section detection, header interpretation, and resume content understanding.

3) AI-Structuring Layer: The AI structuring layer organizes normalized content blocks into the standardized CV schema required by the TMG format. The target schema defines a fixed set of sections consistent with company conventions. These include Profile, Expertise & Results, Key Projects Summary, and Education & Qualifications. Each section contains structured content elements such as paragraphs, bullet lists, or job entries derived from the normalized block sequence.

A large language model is used to interpret the semantic structure of the resume and assign blocks to the appropriate sections. Rather than generating new content freely, the model operates under strict constraints: each output element must reference the original document through block identifiers produced during earlier pipeline stages. This grounding ensures that all generated structure remains traceable to the source resume.

The system prompt defines the target schema and instructs the model to perform only semantic organization of existing content rather than rewriting the document. This design limits hallucination risk, reduces token usage, and allows deterministic validation and rendering stages to operate reliably on the structured output.

4) Rendering Layer: The rendering layer converts the structured resume representation into the final formatted document. The system loads the official TMG Word template and inserts structured content into predefined locations while preserving the template's formatting and style definitions.

Section headings, paragraphs, and bullet lists are rendered using the template's named Word styles to ensure visual consistency with existing TMG documents. Job entries are formatted using table structures to reproduce the layout of the template, including aligned role titles, organizations, and dates.

A template-preserving rendering strategy was chosen rather than constructing formatting programmatically. This approach ensures consistent styling, simplifies maintenance when template designs change, and allows the system to reliably generate proposal-ready CVs that match the organization's established document standards.

B. Output Validation & Evaluation

SmartCV verifies the correctness of AI-generated structures through schema validation and grounding checks. The AI output must conform to a predefined CV schema describing the header, sections, and permitted block types. Responses are validated using a Pydantic schema to ensure structural consistency and required field coverage.

To ensure reliability, all structured elements must reference block identifiers originating from the normalized document representation. This constraint guarantees that the system reorganizes existing content rather than generating new text.

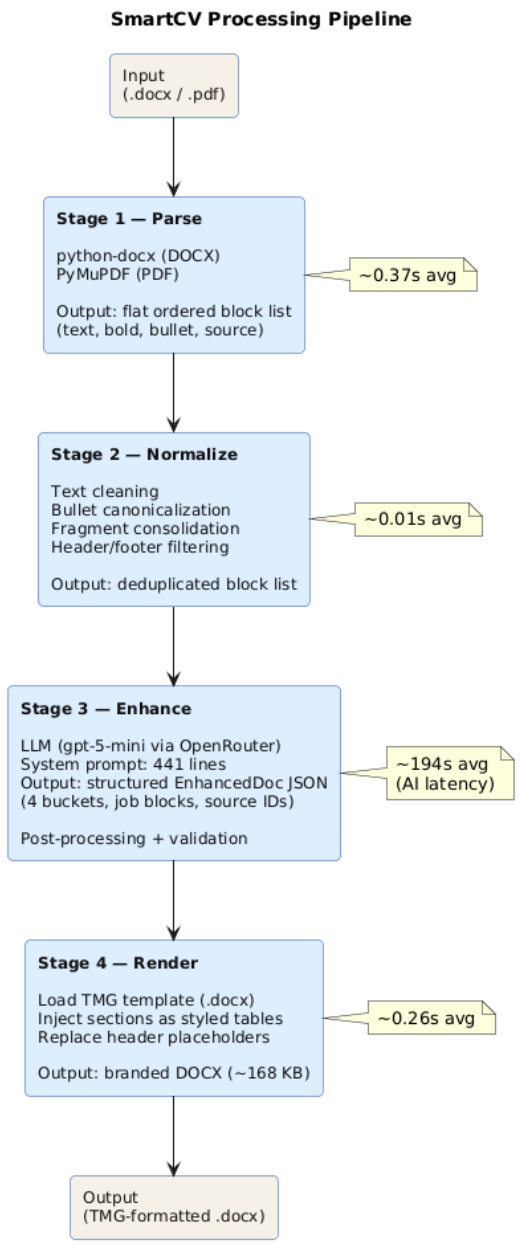

Fig. 1. Overview of the SmartCV processing pipeline.

A confidence score is computed based on schema validation results, section completeness, and coverage of source blocks. System performance is evaluated using metrics such as section detection accuracy, formatting correctness relative to the TMG template, and processing time per document across a set of representative contractor CVs.

C. System Integration & Implementation

SmartCV is implemented as a self-contained Windows desktop application distributed as a single executable installer. The system architecture bundles three primary components: an Electron shell, a PyInstaller-compiled Python backend, and a local SQLite database.

At runtime, Electron launches and spawns the compiled Python backend as a child process, starting a FastAPI server on localhost:8000. The Electron window loads the React frontend, which communicates with the backend through HTTP requests in the same manner as during development. When the application closes, Electron terminates the backend process. A local SQLite database stores conversion records, token usage statistics, and generated documents, enabling the application to operate without external infrastructure.

The backend is implemented in Python (3.11) using the FastAPI framework to orchestrate the four-stage processing pipeline. A Server-Sent Events endpoint streams progress updates to the frontend as each stage completes. Document parsing uses python-docx for DOCX files and PyMuPDF for PDF documents. AI-assisted structuring is performed through a large language model accessed via the OpenRouter API, with the model configurable at runtime through a settings interface. Database interactions are handled using SQLAlchemy's asynchronous engine.

The frontend is implemented using React 18, TypeScript, and Tailwind CSS. It provides a drag-and-drop upload interface, a real-time progress indicator across the pipeline stages, an in-browser DOCX preview powered by docx-preview, and a validation panel displaying model confidence scores and warnings. A dashboard view presents historical conversions and aggregated statistics derived from the database.

Desktop packaging is performed using electron-builder, which bundles the Electron shell, compiled backend, and installer into a single distributable. Because Electron functions only as a process manager and native window host, the core pipeline, database schema, and frontend code remain unchanged between development and desktop deployment.

D. Cloud-Based LLM Inference vs. Local Model Training

An important architectural decision in SmartCV was to use a cloud-hosted large language model API for the structuring stage rather than training or fine-tuning a local model. Fine-tuning a model for this task would have required access to hundreds or thousands of contractor CVs paired with manually labeled JSON input–output examples matching the target schema. Constructing such a dataset would involve significant manual annotation effort and would be complicated by the high variability present in real-world resumes. Contractor CVs vary widely in format, layout structure, section ordering, length, and language, and frequently include both general and domain-specific terminology related to engineering, mining, construction, and energy sectors. This variability would make creating a consistent training dataset difficult and expensive.

A fine-tuned model would also reduce system flexibility. Changes to the output schema, modifications to the CV template, or adapting the system for another company would likely require additional retraining. The task further requires models capable of handling large context windows due to the combination of full resume content and a detailed output schema specification.

Using a cloud-based LLM API provides several advantages for this application. Large pretrained models can reason about document structures they have not explicitly seen before, allowing the system to handle diverse resume formats and edge cases more effectively. The model can infer logical groupings such as job histories and bullet lists even when formatting is inconsistent, and can recognize when information is unclear rather than forcing incorrect structures. Additionally, the operational cost of API inference remains low relative to the time saved from automating manual formatting workflows. By using deterministic stages for parsing, normalization, and template-based rendering, the system limits the role of the language model to structural reasoning, reducing the number of tokens required for inference and making cloud-based deployment both practical and scalable.

Future work may explore hybrid approaches such as locally hosted or fine-tuned models for organizations requiring stricter data privacy or high-volume processing environments.

III. Results

This section evaluates the performance and reliability of the SmartCV pipeline using empirical data collected from live pipeline executions. All metrics were derived from terminal logs produced during full pipeline runs across approximately six distinct CV inputs. The evaluation focuses on runtime performance, document structure processing, AI output validation, and rendering reliability.

A. Pipeline Runtime Performance

Table I summarizes the runtime of each pipeline stage. Results show that the parsing, normalization, and rendering stages execute extremely quickly, while the AI enhancement stage dominates total runtime.

TABLE I
AVERAGE RUNTIME PER PIPELINE STAGE

Stage	Average Time (s)
Parse	0.37
Normalize	0.01
Enhance (AI)	193.7
Render	0.26
Total	**194.3**

Across all runs, the average total processing time per CV was 194.3 seconds, with a minimum runtime of 173.7 seconds and a maximum of 235.3 seconds. The AI enhancement stage accounted for approximately 99.7% of the total pipeline runtime, while the remaining deterministic stages collectively averaged under 0.65 seconds.

These results demonstrate that the system's computational cost is dominated by the LLM inference step rather than the document processing pipeline itself.

B. Document Block Processing

The parsing stage converts each resume into a sequence of document blocks representing paragraphs, bullets, or headings. Table II summarizes the structural characteristics of processed resumes.

TABLE II
DOCUMENT BLOCK STATISTICS

Metric	Value
Average parsed blocks per CV	162
Minimum parsed blocks	62
Maximum parsed blocks	454
Average normalized blocks	151
Average block reduction	8.2%

Normalization reduced block counts by an average of 8.2%, primarily by merging fragmented bullet lines and removing formatting artifacts. In some cases the reduction was significantly larger (up to 30%), indicating that normalization effectively consolidates noisy PDF extraction artifacts.

C. AI Output Characteristics

The AI structuring stage converts normalized blocks into the structured schema required by the TMG template. Table III summarizes prompt and output sizes observed during evaluation.

TABLE III
AI PROMPT AND OUTPUT STATISTICS

Metric	Average
Prompt length (characters)	13,248
Output length (characters)	18,382
Estimated prompt tokens	$\sim$3,312
Estimated output tokens	$\sim$4,596
Total tokens per CV	$\sim$7,908
Estimated cost per CV	$\sim$ \$0.003

The system prompt accounts for a large portion of the input size due to the detailed schema constraints used to guide AI structuring. Despite this, the total estimated inference cost per CV remains extremely low.

D. Validation and Reliability

Each AI response is validated against a strict schema and scoring system that evaluates structural correctness and source grounding. Table IV summarizes validation outcomes across all runs.

TABLE IV
VALIDATION METRICS

Metric	Value
Average confidence score	96.25 / 100
Minimum confidence score	95
Maximum confidence score	100
Validation warnings	0
Runs passing validation	8 / 8

All pipeline executions passed validation successfully, and no warnings were produced. The consistently high confidence scores indicate that the AI outputs reliably conform to the required schema and structural constraints.

Further analysis of the most recent pipeline run confirmed that all normalized source blocks were referenced in the generated output, resulting in 100% source block coverage with no dropped content.

E. Rendering Consistency

The final stage renders the structured document into the TMG Word template. Table V summarizes rendering outcomes.

TABLE V
RENDERING RESULTS

Metric	Value
Successful renders	7 / 8
Failed renders	0
Interrupted runs (server reload)	1
Average output DOCX size	171,732 bytes
Output size variance	$\pm 1.5\%$

All runs returned successful HTTP responses and produced valid DOCX outputs. The narrow range of output file sizes indicates that document formatting is dominated by the template structure rather than the input CV content, confirming that the template-preserving rendering strategy produces highly consistent outputs.

F. Summary

Overall, the evaluation demonstrates that SmartCV reliably transforms heterogeneous resumes into standardized CV documents while maintaining strict schema validation and formatting fidelity. The deterministic stages of the pipeline execute extremely quickly, while the majority of processing time is spent in the AI structuring stage. Despite this, the system achieves consistent outputs, strong validation scores, and low per-document inference costs.

IV. Conclusion

This work presented SmartCV, an automated pipeline for transforming heterogeneous resumes into standardized, company-specific CV documents. The system combines deterministic document processing with AI-assisted structural inference to extract, organize, and render resume content into the formatting standards required by Technical Management Group Ltd. (TMG). Evaluation results demonstrate that the system consistently produces validated outputs with stable rendering behavior and low operational cost, while maintaining complete source coverage and reliable schema compliance.

A key strength of the SmartCV architecture is its modular pipeline design. By separating parsing, normalization, structural inference, validation, and rendering into independent stages, the system can be easily adapted to support future formatting changes or additional template requirements. In practice, this means that new CV formats or proposal templates can be supported by modifying the schema and rendering layer without requiring significant changes to the overall system architecture.

The system also demonstrates how large language models can be effectively integrated into deterministic document-processing pipelines. Rather than relying on a fully learned end-to-end model, SmartCV uses a schema-constrained AI structuring stage that reasons about diverse resume layouts and edge cases while maintaining strict validation and rendering guarantees. A cloud-hosted LLM API was selected for this stage due to the high variability of contractor CV formats, the absence of large labeled training datasets required for fine-tuning, and the need for models capable of processing large context windows that include both resume content and schema constraints. This approach allows the system to generalize to unseen resume formats while avoiding the cost and complexity of building and maintaining a specialized fine-tuned model.

Although SmartCV was developed for TMG's internal proposal workflow in the mining, energy, and construction sectors, the underlying methodology is broadly applicable. Many consulting, engineering, and professional services organizations face similar challenges when aggregating resumes from employees or contractors into standardized formats for project proposals. The combination of structured document parsing, schema-constrained AI reasoning, and deterministic template rendering demonstrated in this work provides a practical and extensible approach for automating these workflows.

Future work may involve exploring hybrid approaches using locally hosted or fine-tuned models for organizations requiring stricter data privacy or high-volume processing environments. Together, these extensions would allow SmartCV to evolve from a document transformation pipeline into a broader platform for managing standardized professional profiles across organizations.

V. Acknowledgments

The authors thank Technical Management Group Ltd. (TMG) for providing the industry use case that motivated this project and for their guidance on CV formatting requirements used in engineering and consulting proposals within the mining, energy, and construction sectors. In particular, we thank Taylor Balsky and Charleigh Priestman from TMG for their feedback, testing, and guidance throughout the development process. The authors also acknowledge the support of QMIND and the CUCAI program for providing the research and development environment in which this work was conducted and presented.

References

[1] R. Saini and A. Gupta, "Resume parser using natural language processing," *ResearchGate*, 2022.

[2] M. Ghaffari *et al.*, "Hierarchical resume parsing via sequence labeling," *arXiv preprint arXiv:2309.07015*, 2023.

[3] Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou, "Layoutlm: Pre-training of text and layout for document image understanding," in *KDD*, 2020.

[4] G. Kim *et al.*, "Docllm: A layout-aware generative language model for document understanding," *arXiv preprint arXiv:2401.00908*, 2024.