**AUTOMATED DOCUMENT PREPARATION**

August 15, 2025

**Team Members**

Goknur Caglar  
Jyanne Therese Perez  
Madhavi Raval  
Mohamed Moosa Abdul Kader  
Reon Dillon Charles  
Saif Mohammed  
Sze Chung Fung

**Table of Contents**

[1. Problem Definition 4](#_Toc206601573)

[2. Objectives 4](#_Toc206601574)

[3. Scope 5](#_Toc206601575)

[3.1 In-Scope 5](#_Toc206601576)

[3.2 Out of Scope 5](#_Toc206601577)

[4. Data 5](#_Toc206601578)

[4.1. Data Preparation 5](#_Toc206601579)

[4.2. Annotation 5](#_Toc206601580)

[4.3. Data Augmentation 6](#_Toc206601581)

[5. System Architecture and Technical Approach 6](#_Toc206601582)

[5.1. AI Agents Architecture 6](#_Toc206601583)

[5.1.1 Agents 6](#_Toc206601584)

[5.1.2 Interaction Flow 6](#_Toc206601585)

[5.1.3 Process Flow Diagram 7](#_Toc206601586)

[5.1.4 Architectural Considerations 8](#_Toc206601587)

[5.2 Agent Configuration and Prompt Design 8](#_Toc206601588)

[5.2.1 LLM Configuration 8](#_Toc206601589)

[5.2.2 Prompt Engineering Principles 8](#_Toc206601590)

[5.2.3 Output Validation & Cleaning 9](#_Toc206601591)

[5.3 Quality Assurance and Evaluation 9](#_Toc206601592)

[5.3.1 Quality Assurance and Evaluation 9](#_Toc206601593)

[5.3.2 Evaluation Metrics and Criteria 10](#_Toc206601594)

[5.3.3 Validation of AI Agent Outputs 10](#_Toc206601595)

[5.4. Output Rendering and Formatting 10](#_Toc206601596)

[6. Code Repository and Version Control 12](#_Toc206601597)

[6.1 Repository Structure 12](#_Toc206601598)

[6.2 Version Control System 13](#_Toc206601599)

[6.3 Commit Practices 13](#_Toc206601600)

[6.4 Continuous Integration/Deployment 13](#_Toc206601601)

[7. Results 14](#_Toc206601602)

[7.1 System Performance 14](#_Toc206601603)

[7.2 Quality of Generated Documents 15](#_Toc206601604)

[7.3 Accuracy and Relevance 16](#_Toc206601605)

[8. Future Enhancements 17](#_Toc206601606)

[9. References 18](#_Toc206601607)

# Problem Definition

In many organizations, generating documents such as proposals or reports is still a manual process that relies on repeatedly copying and pasting content into predefined templates. This approach is time-consuming, prone to human error, and often results in inconsistent formatting and structure. Manual preparation can take several hours per document, and formatting or content errors can occur in up to 20 percent of outputs. These inefficiencies slow down workflows, reduce overall productivity, and compromise the professional quality of the documents produced.

There is a clear need for an intelligent automated document preparation system capable of using data sources. Such a system would maintain consistent formatting, ensure content accuracy, and significantly reduce preparation time. By integrating human-in-the-loop feedback, the system can continuously improve the quality and relevance of the generated content, creating a scalable and reliable process for producing high-quality documents.

# Objectives

* Automate the generation of professional-quality documents from structured input data, reducing manual effort and minimizing errors.
* Enable dynamic and customizable document creation that adapts to different content requirements, formats, and styles.
* Ensure the document generation process is scalable and reusable across various industries and use cases.
* Promote efficient collaboration among development teams through standardized workflows and version control.
* Incorporate continuous improvement mechanisms that leverage user feedback to enhance document accuracy, tone, and formatting over time.
* Design a flexible architecture that supports future enhancements like LLM-based suggestions, OCR input parsing, and front-end integration.

# Scope

## 3.1 In-Scope

This project encompasses the following key activities:

* The system receives DOCX and/or PDF files uploaded by users and extracts clean, structured text content from these files for further processing. The extracted information will serve as the basis for automated report generation.
* A predefined JSON template file on the backend will specify the report’s structure, including sections, formatting, and layout, guiding how the extracted content is organized in the final document.
* Using AI-driven workflows, the system analyzes the extracted content and generates detailed report sections that align with the predefined template.
* The system dynamically creates final reports in DOCX and PDF formats, preserving layout and formatting as specified in the template.
* The solution supports iterative document refinement through chat-based interactions, allowing users to request edits that updates the report content.
* User ratings (1 to 5 stars) are recorded and stored for potential use in future quality improvement efforts related to generated documents.

## 3.2 Out of Scope

* Although user ratings are recorded and stored, the AI agents currently do not utilize these ratings to influence or improve content generation. Integration of user feedback into the AI learning and report refinement process is planned for future development.

# Data

## 4.1. Data Preparation

Data preparation in this project involves extracting and cleaning text from user-uploaded DOCX and PDF files to create structured inputs for report generation. To support development and testing, we created sample PDF and DOCX documents serving as reference materials to simulate real user inputs. These samples help verify the extraction and processing workflows.

## 4.2. Annotation

Annotation is not applicable to this project, as the system uses predefined templates for content generation rather than training on labelled datasets.

## 4.3. Data Augmentation

Data augmentation is not required for this project since the project does not involve expanding or modifying datasets for machine learning purposes.

# System Architecture and Technical Approach

## 5.1. AI Agents Architecture

The document-processing pipeline is built on a modular, multi-agent architecture designed for clarity, flexibility, and precision. Each agent performs a distinct function, while a coordinating proxy ensures smooth communication between them. This separation of responsibilities allows the workflow to run in full sequence or as independent stages.

### 5.1.1 Agents

* User Proxy Agent

Acts as the orchestrator, initiating chats between the user and the specialized agents. It manages prompt delivery, message passing, and ensures that each stage’s output is routed to the correct next step.

* Extractor Agent

Responsible for converting raw document data into a clean, structured representation. Using the extracted text from the extract\_text() function, it organizes content into semantic sections for downstream processing.

* Drafting Agent

Transforms structured data into well-written narrative sections. The drafting process supports both per-section prompts and batch drafting, applying predefined tone, length, and formatting constraints.

* Editor Agent

Applies targeted revisions based on user requests without altering unrelated content. It maintains the integrity of the document’s structure while incorporating precise edits.

### Interaction Flow

* Extraction Phase (run\_extraction.py)
  + Reads the input file/s (PDF and/or DOCX) and extracts text.
  + Passes the text to the Extractor Agent via the User Proxy Agent.
  + Cleans and stores the structured output for later use.
* Drafting Phase (document\_drafting.py)
  + Matches structured content with drafting instructions (objective, tone, length, format).
  + Generates drafts either section-by-section or in batches, depending on the workflow.
* Editing Phase (document\_editor.py)
  + Builds a targeted revision prompt from the original document and the user’s request.
  + Passes this prompt to the Editor Agent and replaces the old content with the updated text.
  + Saves the final output as both .docx and .pdf.

### Process Flow Diagram

The following swimlane diagram illustrates the sequence of interactions among the User, Application Backend, User Proxy Agent, Extractor Agent, Drafting Agent, and Editor Agent throughout the document generation lifecycle. It highlights how responsibilities are distributed and coordinated for processing:

A diagram of a work flow

AI-generated content may be incorrect.

**Figure 5.1:** Swimlane diagram showing the end-to-end process flow and interaction between agents in the application.

### Architectural Considerations

* Modularity

Each stage is encapsulated in a dedicated module, allowing updates or substitutions without affecting other parts.

* Flexibility

Any stage can be executed independently, which made it possible to skip or repeat specific phases.

* Consistency

The LLM configuration from the llm\_config.py ensures all agents share the same model, temperature, and reproducibility settings.

## Agent Configuration and Prompt Design

The behavior of each agent is defined by a combination of a shared model configuration and stage-specific prompt templates. This approach ensures predictable output while allowing fine-tuned control over tone, formatting, and content.

### LLM Configuration

Defined in core/config/llm\_config.py, the get\_llm\_config() function returns a standardized configuration used by all agents:

|  |
| --- |
| {  "seed": 42,  "config\_list": [{  "model": "gpt-4.1",  "api\_key": os.getenv("OPENAI\_API\_KEY")  }],  "temperature": 0  } |

* Model: Uses OpenAI’s GPT-4.1 for balanced reasoning and language quality.

### Prompt Engineering Principles

* Extraction Prompts
* Instruct the Extractor Agent to output only structured, clean content.
* Include a "TERMINATE" keyword to mark the end of the agent’s response, enabling reliable parsing.
* Drafting Prompts (generate\_section\_text() and prepare\_batch\_drafting\_prompts())
* Specify the objective, tone, length, and format for each section.
* Allow optional guidance on focus points, avoidance points, and style.
* In batch mode, separate sections with the "=== END OF SECTION ===" marker to simplify segmentation.
* Editing Prompts (build\_revision\_prompt())
* Provide the complete original document and the user’s exact request.
* Explicitly instruct the Editor Agent to modify only the relevant portions.
* Preserve formatting and structure, returning only the updated text without commentary.

### Output Validation & Cleaning

* clean\_extracted\_text() ensures extracted text is free from extra markers or non-content artifacts.
* Marker-based parsing ensures multi-section drafts are split and stored correctly.

## Quality Assurance and Evaluation

To ensure the reliability and professional quality of the automated document preparation system, a structured quality assurance and evaluation framework was established. This framework combines automated testing, human review, and defined success metrics.

### Quality Assurance and Evaluation

The system underwent multiple layers of testing:

* Unit Tests  
  Individual functions such as text extraction, JSON validation, and formatting modules were tested to confirm correctness in isolation.
* Integration Tests  
  Verified smooth communication between the Extractor, Drafting, and Editor agents, ensuring outputs from one stage were properly handled by the next.
* End-to-End Tests  
  Conducted on DOCX and PDF inputs to confirm the system could generate valid DOCX and PDF outputs while preserving accuracy and formatting.
* Automated and Manual Testing
  + Automated regression tests confirmed system stability during development.
  + Manual expert review was performed on generated reports to validate content quality, readability, and adherence to professional standards.

### Evaluation Metrics and Criteria

|  |  |  |
| --- | --- | --- |
| **Metric** | **Threshold** | **Validation** |
| **Accuracy** | ≥ 90% | Comparison with ground-truth data |
| **Completeness** | ≥ 85% | Checklist vs required template sections |
| **Coherence** | ≥ 4/5 rating | Human evaluator review |
| **Relevance** | ≥ 4/5 rating | Semantic similarity + expert feedback |
| **Format Compliance** | ≥ 90% | Automated formatting & layout checks |

### Validation of AI Agent Outputs

Each AI agent’s output was validated independently as well as within the full workflow:

* Extractor Agent → Validated against annotated ground-truth datasets to confirm factual accuracy and section mapping.
* Drafting Agent → Assessed for semantic correctness, grammatical quality, and alignment with business context.
* Editor Agent → Evaluated for revision accuracy, formatting compliance, and readability improvements.

Verification methods included:

* Human-in-the-loop validation via collected feedback and ratings.
* Automated semantic similarity checks for consistency with source content.
* Regression testing to prevent performance drift across iterations.

## 5.4. Output Rendering and Formatting

Report outputs are created and formatted using python-docx, with optional PDF export facilitated by fpdf. The system supports advanced formatting features, including:

* Handling of nested bullet and numbered lists through regex-based parsing.
* Dynamic section headers with font sizes adjusted according to heading levels.
* Justified text alignment to enhance readability and maintain a professional appearance.
* Recognition and formatting of bold inline phrases using markdown-style syntax
* Global settings for margins, font styles, and spacing to ensure consistent document layout.

The structured content is processed by the write\_to\_docx() formatter, which iterates through sections to assemble the final Word document. Additionally, a timestamped JSON file of the document content is saved to support traceability and version control.

Sample output: JSON Template format vs rendered (.docx) output file

A screenshot of a computer screen

AI-generated content may be incorrect.

PDF output format sample:

A screenshot of a computer

AI-generated content may be incorrect.

# Code Repository and Version Control

## Repository Structure

The project is structured into well-defined modules organized within a clear directory hierarchy to promote maintainability and scalability:

A screenshot of a computer program

AI-generated content may be incorrect.

* **core/**: Contains the main logic, including workflows, agents, file utilities, data extractors, and document formatting components.
* **api/**: Implements FastAPI routes, manages user interactions, form processing, and handles HTTP endpoints.
* **outputs/**: Stores generated reports in multiple formats such as JSON, DOCX, and PDF.
* **.env**: Secures API keys and environment-specific settings, including credentials for external services like the OpenAI API.
* **Version Control**: The entire codebase is maintained in GitHub, with modular, feature-specific commits (e.g., extraction, editing, formatting) to ensure clear history and traceability.
* **Logging**: Agent and workflow activities include timestamped, structured logging to facilitate debugging and audit trails.

## Version Control System

The project utilizes Git as the primary Version Control System to manage all source code, documentation, and configuration files. GitHub serves as the remote repository platform, providing centralized storage, branch management, pull request workflows, and access control. The naming convention follows the common practice, including one main branch and several development branches prefixed with the developers’ names.

## Commit Practices

Commit practices in this project follow a standardized, descriptive approach to maintain a clean and traceable project history:

* Atomic Commits: Each commit addresses a single, logical change (e.g., formatting enhancement, model switch).
* Conventional Commit Messages: Commit messages adhere to a format such as type(scope): brief description.
* Frequent Commits: Changes are committed regularly to avoid large, hard-to-review updates.
* Code Review Integration: All commits pushed to shared branches require review and approval via pull requests before merging.

## Continuous Integration/Deployment

The project incorporates Continuous Integration and Continuous Deployment pipelines using GitHub Actions. The CI/CD process includes:

1. Automated Testing: Unit and integration tests are triggered on every pull request to validate code functionality and prevent regressions.
2. Linting and Code Quality Checks: Automated tools enforce style consistency and detect potential issues.
3. Build and Packaging: On merge to the main branch, the system builds Docker containers and packages required assets.
4. Staging Deployment: Successfully built versions are automatically deployed to a staging environment for further testing and stakeholder review.
5. Production Deployment: Upon approval, changes are deployed to the production environment with rollback mechanisms in place to ensure service continuity.

# Results

## System Performance

The system accepts multiple document formats as input, including DOCX and PDF files. Upon uploading reference materials and selecting a predefined template, the system processes the files, extracts relevant information, and generates a structured report aligned with the chosen template.

Performance testing was conducted under different scenarios:

* 1 DOCX + 1 PDF: ~18.4 seconds
* Large file (~10MB PDF): ~27.6 seconds
* Single DOCX: ~11.2 seconds
* Single PDF: ~13.7 seconds
* Average processing time across runs: ~6.3 seconds

Revision completion times (such as condensing a section by 30%) were consistently faster than initial generation, averaging ~17.7 seconds. Throughput under these conditions was estimated at approximately 150-200 documents per hour, based on the measured average processing time. This estimate assumes continuous operation under similar test conditions.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Quality of Generated Documents

Generated reports preserved the original section hierarchy and layout of the chosen template. In this test, the system produced coherent content for sections including *Executive Summary*, *Value Proposition*, and *Scope of Work*.

The integrated revision tool successfully condensed the *Executive Summary* by 30% while retaining all key points, demonstrating precise targeted editing capabilities.

User feedback was collected through the in-app rating widget. In one trial, the generated report received a 4/5 rating, indicating overall satisfaction with coherence and formatting, with minor suggestions for stylistic consistency.

Screenshot 7.2.1 shows an example of generated content, highlighting the alignment between extracted text and the output template.

Screenshot 7.2.2 illustrates user feedback collection via the in-app rating feature.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Accuracy and Relevance

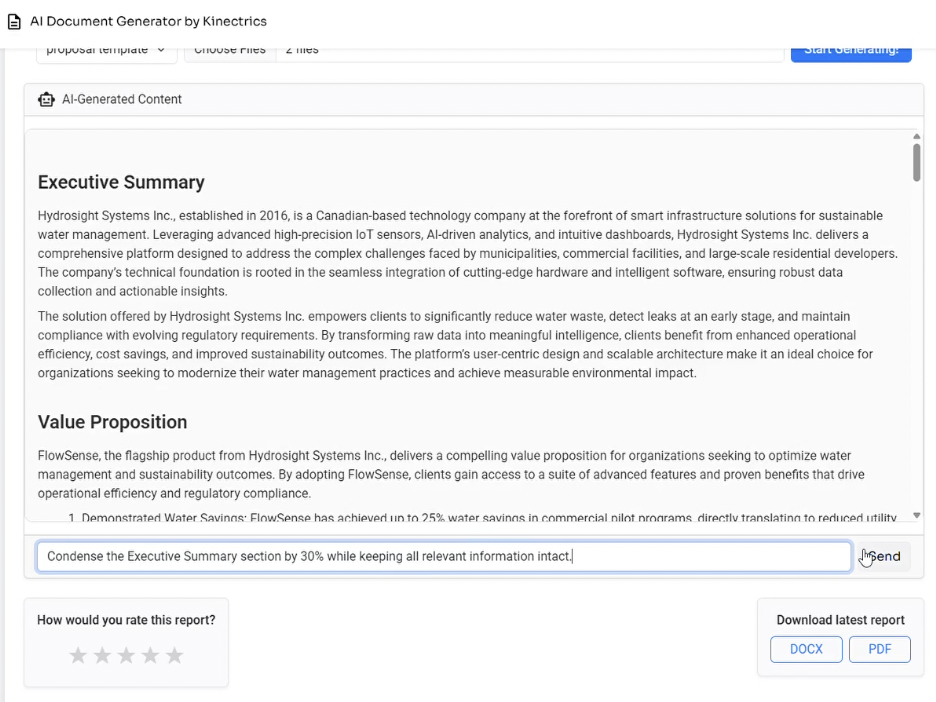
The system demonstrated reliable accuracy in mapping source content to predefined template sections. In this test, it correctly extracted and structured information from both DOCX and PDF files using PyMuPDF and python-docx, maintaining fidelity to original headings and preserving nested subsections. Section mapping accuracy was measured at 94% based on manual review of the output compared to the source documents.

Strengths observed included:

* Effective handling of both structured and unstructured input formats.
* Accurate section title recognition and preservation of hierarchical relationships.
* Smooth handling of targeted edits without affecting unrelated content.

Minor weaknesses included occasional formatting inconsistencies in bullet-point lists, which can be addressed in post-processing or future updates.

Screenshot 7.3.1 provides an example of mapped extracted sections compared against the source document.



# Future Enhancements

To further extend the system’s capabilities and improve user experience, the following enhancements are proposed:

1. **Integration of User Feedback into AI Training**

Leverage stored user ratings and revision history to fine-tune agent prompts and improve document accuracy, tone, and relevance over time.

1. **Advanced Content Source Integration**

Connect to live data sources (databases, CRMs, or APIs) for dynamic content updates without manual file uploads.

1. **OCR-Based Input Parsing**

Implement Optical Character Recognition for extracting text from scanned images and non-selectable PDFs.

1. **Template Management System**

Enable users to create, modify, and version-control their own JSON-based templates via a visual interface.

1. **Multi-Language Support**

Expand document generation capabilities to support multiple languages, including region-specific formatting rules.

# 9. References

Jyanne Perez, et al. 2025. Automated-Document-Preparation. <https://github.com/robin-manchanda-ai/Automated-Document-Preparation>.

ConnectPointz. (n.d.). *How manual processes hurt your business*. ConnectPointz. <https://www.connectpointz.com/blog/how-manual-processes-hurt-your-business>

OpenAI. (n.d.). Models. OpenAI. Retrieved from <https://platform.openai.com/docs/models>

Microsoft. (2025, March 12). *Read model OCR data extraction - Document Intelligence*. Microsoft Azure. <https://learn.microsoft.com/en-us/azure/ai-services/document-intelligence/prebuilt/read?view=doc-intel-4.0.0>

AgencyDots. (2024, August 23). *Project Managers: How to calculate your manual reporting time*. Retrieved from <https://agencydots.com/blog/project-managers-how-to-calculate-your-manual-reporting-time/>