# Document Summarizer and Highlighter

**Nam Bui**

**May 13, 2025**


## Abstract

This paper presents a comprehensive pipeline for automated text extraction, structural analysis, and content summarization from PDF documents. The methodology integrates image processing techniques using OpenCV for robust text region detection and Optical Character Recognition (OCR) enhancement via PyMuPDF. A simple direct text extraction approach is also implemented. To ensure compatibility, a robust classification system is employed to adapt to diverse PDF structures, directing inputs through an optimal pipeline for single and multi-column documents. Extracted text elements are organized into a hierarchical structure based on proximal text relationships, which is then transformed into a structured JSON representation. This structured data undergoes further processing, including watermark filtering and cohesive structure finals. A large language model (LLM) is then leveraged with this structured output scheme to generate concise summaries and identify supporting verbatim evidence snippets from the document. These highlighted evidence snippets are then programmatically integrated back into the original PDF, providing a summarized and annotated version of the input document. The pipeline aims to handle complex PDF layouts and deliver actionable insights through summarization and evidence localization to highlight and improve the existing evidence base for readers.


## Introduction

The proliferation of digital documents, particularly in PDF format, necessitates efficient methods for information extraction and summarization. Manually processing large volumes of text to identify key information and supporting evidence is time-consuming and often impractical. This project addresses the challenge by developing an automated pipeline capable of not only summarizing document content but also highlighting the precise textual evidence supporting the summary. The system is designed to handle varied document layouts, including complex multi-column structures, by first classifying the document and then routing it through an appropriate processing workflow. By integrating image processing, OCR, structural analysis, and large language models, this work aims to provide users with a tool that enhances their ability to quickly understand and utilize information from PDF documents.

## Datasets

The development and evaluation of this system utilized a diverse collection of documents from various sources and topics. The primary dataset categories include:

- **Journal Articles:** Academic papers often featuring complex layouts, tables, and figures.
- **Philosophical Texts:** Documents that can vary widely in structure and style.
- **Spill Datasets:** Collections of documents used for testing purposes.

These documents were split into approximately 3300 entries for training and testing purposes. A key characteristic of the dataset is its non-uniformity; it comprises a wide range of document structures and formatting, which is crucial for developing a robust and adaptable system. The poster indicates a bar chart showing "Tracking data status (dev, test)" with a significant portion labeled "is being college," suggesting that a substantial part of the data collection or annotation process is ongoing or related to academic sources, and another portion labeled "is standardized," implying some data has undergone a normalization process.

## Methods

The methodology encompasses several stages, from initial document processing to final summary and evidence presentation.

**Feature Extraction and Preparation**

A critical initial step involves robust feature extraction from the PDF documents. This process includes:

1. **Image Preprocessing:**
    - **Grayscale Conversion:** The input document (presumably converted to an image format if starting from PDF for this stage) is converted to grayscale.
    - **Adaptive Thresholding:** Applied to binarize the image, separating text from the background.
    - **Morphological Operations:**
        - **Dilation and Erosion:** These operations are used to connect nearby text regions and refine character shapes.
        - **Edge Detection:** Specifically, Canny edge detection is applied to identify the boundaries of text blocks from the processed image.
2. **Text Extraction:**
    - **PyTesseract OCR:** Optical Character Recognition is performed using PyTesseract for enhanced text extraction from image-based PDFs or scanned documents.
    - **Direct Text Extraction:** A simpler method for directly extracting text from digitally native PDFs is also employed.
3. **Feature Engineering:** The following features are extracted from the identified text regions:
    - **Text Blocks:**
        - median_aspect_ratio
        - median_solidity_ratio
        - median_w_h_ratio
        - percent_doc_covered
        - total_width_ratio
        - median_char_height
        - x_center_norm, y_center_norm
        - width_norm, height_norm
    - These features help in classifying and understanding the properties of different text segments.
4. **Structuring Content:**
    - A custom line-breaking algorithm groups lines of text based on the median character height, prioritizing vertical proximity.
    - A linking algorithm analyzes the proximity between text blocks to form a tree structure, crucial for consistent processing of document content. This involves establishing relationships (parent-child, sibling) between text elements.
    - The structured information is then serialized into a JSON format. This JSON object represents the hierarchical organization of the document's content, including elements like paragraphs, lists, and potentially tables or figures.
    - Watermark filtering is applied to remove irrelevant, repetitive text.
5. **Classification System:** A classification system determines if a document is single-column or multi-column. This allows the pipeline to adapt its processing strategy accordingly, ensuring optimal handling of different layouts.

**Models and Algorithms**

Several machine learning models were tested for tasks such as text block classification or structural analysis. The goal was to determine the best algorithm for a particular application within the pipeline. The models evaluated include:

- **K-Nearest Neighbors (KNN):** Tested with K values ranging from 2 to 151.
- **Gaussian Naive Bayes**
- **Decision Tree**
- **Logistic Regression**
- **Random Forest**
- **Support Vector Machine (SVM)**

For restructuring the document content, the following approaches were used:

- **Median Tracking in DataStream:** This likely refers to tracking median properties of text blocks or lines as the document is processed.
- **Monotonic Increase/Decrease in Order:** This could relate to ordering text blocks based on

their spatial positions (e.g., y-coordinates for top-to-bottom reading order).

- **Constructing Structure using Trees and Nodes:** As mentioned in feature extraction, this is a core part of organizing the document content hierarchically.

For the summarization task, a Large Language Model (LLM) is employed:

- **Base: Gemini 2.0 Flash:** This model is used to generate a concise summary and extract relevant verbatim snippets from the structured text to highlight as evidence.

**Results and Observations**

The system's performance was evaluated through various metrics and observations:

- **Processing Time:** The pipeline demonstrates notable efficiency. On a standard CPU, the full processing pipeline, including text extraction, structural analysis, summarization, and highlighting, achieves an average processing time of **6 seconds per page**. The structured extraction phase, which involves identifying and organizing text elements, is significantly faster, averaging **0.1 seconds per page**.
- Multi-Label Classification: A multi-label scoring model was developed to identify if documents were "structured" and "summarized." Performance was assessed using Precision, Recall, and F1-Score.

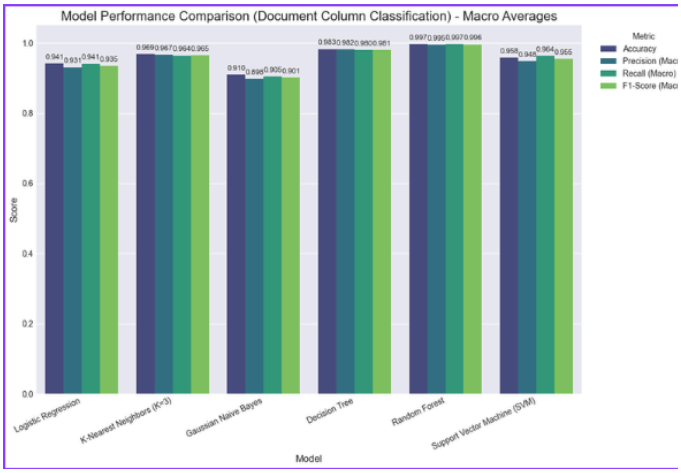
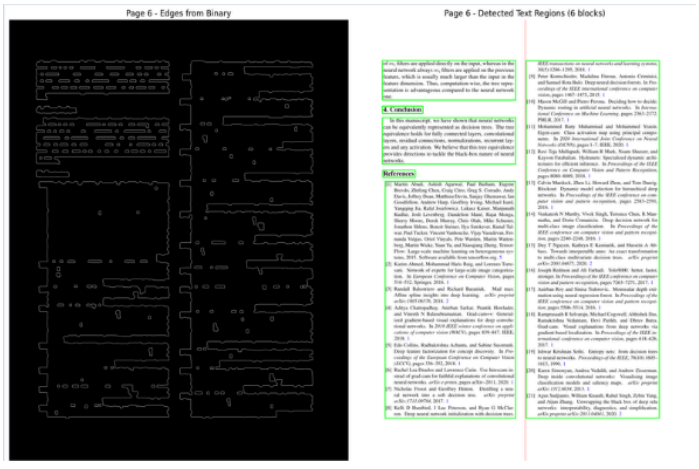



Figure 1: Bar chart showing multi-label classification performance (Precision, Recall, F1-Score) for different models, with Random Forest highlighted as top-performing.
The results show that the Random Forest model was the top-performing model, exhibiting a well-balanced performance across all metrics.

- **Decision Tree Visualization:** The Decision Tree model, while not the top performer, provided insights into distinguishing characteristics of structured vs. unstructured document layouts. The Gini impurity of the root node was 0.468, and a path with a Gini impurity of 0.0 (perfectly classifying) indicates that the model predicts a document as "structured" if median_char_height is <= 0.009 and block_count <= 12.5. This suggests that the model successfully identified features indicative of document structure.
- **Cross-Validation:** Both training and cross-validation scores were reported as high, indicating the model's ability to generalize to unseen data and is not suffering from high variance or overfitting. This suggests robustness in the developed models.
- ETL Pipeline and DehLIA Architecture: The poster mentions a "representative output from the ETL pipeline" showcasing accurate classification and processing, as well as the "flexibility and robustness provided by the DehLIA."




[Example Snippet: Visual representation of a processed document layout, differentiating title, abstract, body text, and showing multi-column handling.]

This evidence demonstrates the system's capability to differentiate between title, abstract, and body text, correctly identifying multi-column layouts and ordering text for coherent summarization. The quality of this structured output is deemed high and suitable for input to the LLM.

- Summary and Highlighting Output:

vMODB: Unifying event and data management for distributed asynchronous applications

Rodrigo Laigner
University of Copenhagen
Copenhagen, Denmark
rnl@di.ku.dk

Yongluan Zhou
University of Copenhagen
Copenhagen, Denmark
zhou@di.ku.dk

arXiv:2504.19757v1 [cs.DB] 28 Apr 2025

ABSTRACT

Event-driven architecture (EDA) has emerged as a crucial architectural pattern for scalable cloud applications. However, its asynchronous and decoupled nature introduces challenges for meeting transactional requirements. Database systems, relegated to serving as storage engines for individual components, do not recognize transactions that span multiple components in EDAs. In contrast, messaging systems are unaware of the components' application states. Weaving such asynchronous and independent EDA components forces developers to relinquish transactional guarantees, resulting in data consistency issues. To address this challenge, we design vMODB, a distributed framework that enables the implementation of highly consistent and scalable cloud applications without compromising the envisioned benefits of EDA. We propose Virtual Micro Service (VMS), a novel programming model that provides familiar constructs to enable developers to specify the data model, constraints, and concurrency semantics of components, as well as transactions and data dependencies that span across components. vMODB leverages VMS semantics to enforce ACID properties by transparently unifying event logs and state management into a common event-driven execution framework. Our experiments using two benchmarks show that vMODB outperforms a widely adopted state-of-the-art competing framework that only offers eventual consistency by up to 3X. With its high performance, familiar programming constructs, and ACID properties, vMODB will significantly simplify the development of highly consistent and efficient EDAs.

CCS CONCEPTS

• Information systems → Data management systems; • Computer systems organization → Distributed architectures.

KEYWORDS

event-driven architecture, data management, log processing, ACID

1 INTRODUCTION

Event-driven architecture (EDA) emerged as an important architectural pattern for designing scalable cloud applications. An EDA application comprises independent components communicating via asynchronous event messages backed by a messaging system, such as Kafka [38]. An event in EDA serves the purpose of exchanging data, triggering remote computations, and communicating failures. Event producers and consumers in the system are oblivious to each other, promoting loose coupling between them.

The loose coupling of EDA components boosts the independence of component development, allowing different teams to implement and evolve their components with little coordination. As a result, it facilitates the introduction of new components and features, promoting fast-paced changes and supporting business innovation [34]. In addition, EDA brings about many technical benefits, including fine-grained optimization of resource provisioning, elastic scaling of individual components, quicker upgrades and bug fixing, facilitation of feature reuse, more flexibility in integrating with external and legacy systems, etc. [7]. The popularization of EDA has prompted cloud providers to build specific-purpose technologies and adapt their services to support EDA applications. For instance, Amazon AWS offers event messaging services like EventBridge, SNS, and SQS, and workflow orchestration services like Step Functions to facilitate building EDAs [5].

The dominant practice for EDAs employs traditional web application frameworks such as Spring, ASP.NET, and Node.js to implement the application components while relying on messaging systems to enable asynchronous communication [71]. Typically, EDAs follow design patterns resembling the BASE model [61] and SAGAS [28], resorting to eventual consistency. Recent studies [39, 40, 74] and reports [3, 12–14, 22, 23, 29, 36, 45, 49, 64–67, 70, 77] indicate that such practices leave developers facing many challenges in achieving application safety properties. Being able to execute ACID transactions across multiple components in EDAs would eliminate most of the reported challenges [40].

The technical reasons that EDAs give up ACID include the following. First, event messaging and logging systems such as Kafka [38] are unaware of the progress of transaction execution and the state update operations at the individual components. Therefore, the best achievable outcome is eventual consistency, which relies on message delivery retries and eliminating the impact of duplicate messages [37, 61]. This task per se is complicated in an asynchronous system. Recently emerging distributed application frameworks like Dapr [19] address some of these challenges by offering transparent tracking of event consumption and automatic retries to achieve at least once processing. However, the responsibility for achieving exactly once processing still falls on application developers.

Second, the database systems managing the states of individual components are oblivious to how components exchange events to complete a transaction spanning multiple components [33]. In other words, they are unaware of the global structure of multi-component transactions or the data contained in the events representing the intermediate transaction states. In addition, a database system typically does not allow multiple asynchronous components to interact with the same database within the context of a single ACID transaction. Therefore, database systems cannot achieve ACID properties [9] for multi-component transactions on their own.

[Figure 4: Example of a document page with highlighted sections corresponding to the generated summary.]

A visual example demonstrates the final output of the system, showing a document page with highlighted sections corresponding to the generated summary.

**Discussion/Reflections**

The development process and results led to several key reflections:

- **Complexity Analysis:** Dedicating more resources to in-depth analysis of runtime and training algorithm complexity is crucial. More complex algorithms may offer higher accuracy but at the cost of computational resources.
- **Computational Efficiency:** Fine-tuning models is an inherently expensive and time-consuming process. Therefore, strategic implementation of efficient data structures and algorithms is emphasized to significantly reduce computational runtime.
- **Resource Optimization:** For computationally intensive tasks, balancing performance with resource constraints is vital. This includes considering trade-offs when computing resources are limited, and ensuring equal performance regardless of software architecture and algorithmic sophistication.
- **Robust Evaluation:** Advocating for applying multiple training methodologies to a range of models is recommended to achieve a more comprehensive and robust evaluation of outcomes. This helps in selecting the most suitable model for each specific task within the pipeline.
- **Strategic Implementation:** Emphasizing that strategic implementation of efficient data structures and algorithms significantly reduces computational runtime. This involves careful design choices throughout the pipeline.

**Future Work**

The project has several avenues for future development and improvement:

- **Enhanced Features using finetune model Gemma 3.0 (1B):**
  - Continue finetuning using GRPO with the proposed Reward Function
  - Implement a Gaussian curve to calculate conciseness and coherence scores for summaries.
- **Advanced Similarity Metrics:**
  - Integrate a "Modular-LCS score" in tandem with Cosine Similarity to determine confidence, support value, and reference similarity between the summary and source text.
- **Improved Text Tokenization:**
  - Implement a "Sentence Transformer" to quickly tokenize text.
- **Enhanced Information Retrieval:**

    - Focus on loading chunks and summaries, and extracting entities onto a vector database. This will aid in building a recall information system for bodies of documents.
- **Performance Optimization:**
    - Optimize runtime further by implementing multi-threading and multi-processing capabilities.

**References**

[1] BookSum dataset: https://github.com/salesforce/booksum/tree/main
[2] Gemini API: https://ai.google.dev/gemini-api/docs
[3] arXiv: (https://arxiv.org/)