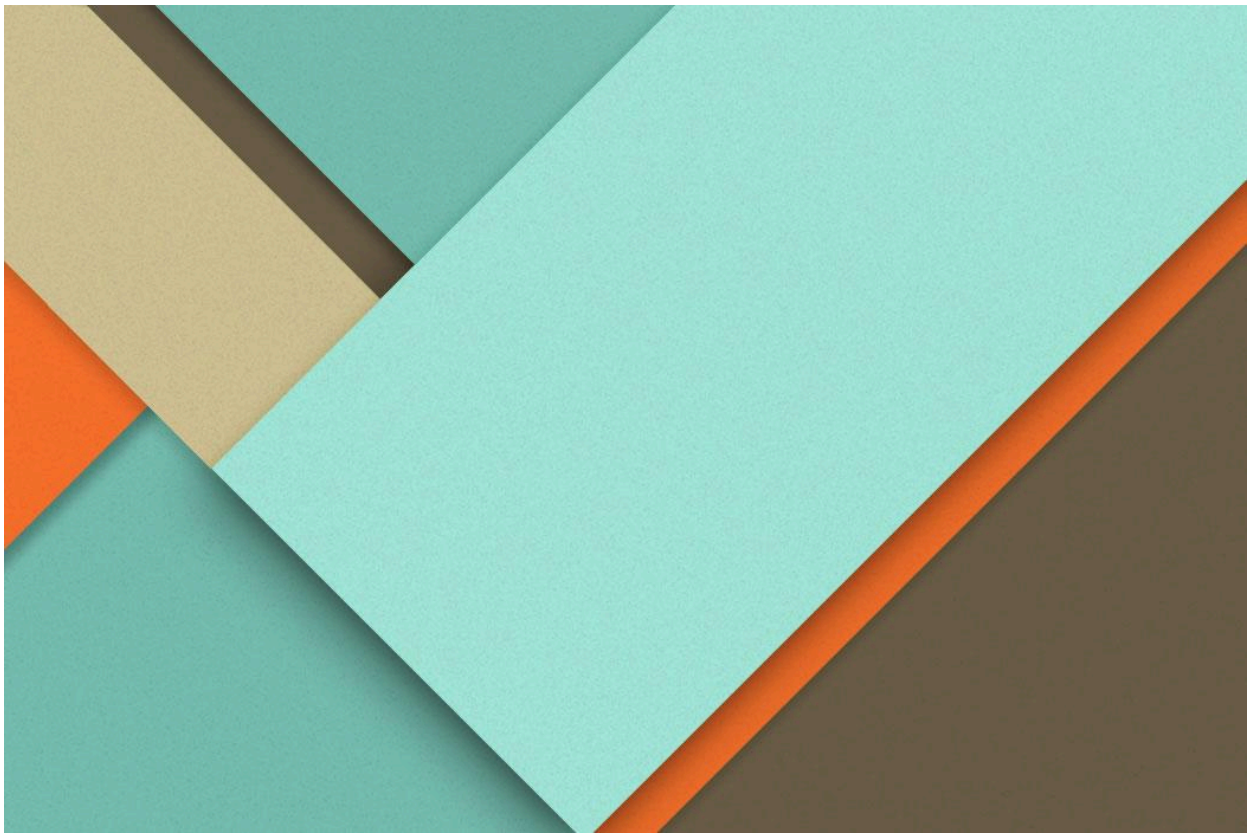

Contextual Compression for Extreme Long Inputs

Team Name : hitansh_rg

Members : Hitansh Gandhi, Dhyaan Shah and Saatvik Sachdeva

Abstract

We are creating a transparent document compression system designed to transform very large textual documents into structured, explainable JSON notes. The system does not generate free-form summaries but extracts decision-critical information such as rules, exceptions, risks, numbers and contradictions from the

original text. Every statement in the output remains traceable to its source section and page range.

Our model addresses the problem that conventional compression hides reasoning and silently drops information. Instead of acting as a black box, our pipeline shows what was kept, what was removed and why. The final output supports question answering and drill-down to the original document without requiring the full text.

Introduction

Large organisational documents such as policies, legal contracts and compliance manuals are often hundreds of pages long. Humans cannot quickly identify the portions that influence real decisions. Traditional summarization tools reduce length but sacrifice explainability and auditability.

Compression for enterprise use must satisfy three requirements:

- Retain all decision-critical content
- Provide structured representation instead of plain paragraphs
- Allow tracing each point back to the original document

This project proposes a rule-based, offline compression pipeline that converts a single large document into hierarchical JSON notes. The system avoids generative rewriting and focuses on extraction, categorization and aggregation of factual statements.

Problem Statement

The objective is to design a system that:

- Accepts a large document as input
- Produces a compressed representation
- Still supports:
    - Question answering
    - Drill-down to original text
    - Evidence and citations

The compression must be explainable. The output should make it possible to answer:

- Why was this information included?
- What was removed and why?
- Can important points be traced back?

System Overview

The pipeline follows a deterministic multi-stage workflow.

Step 1 – Structured Input

The system receives a JSON structure generated from the document:

- Document ID and metadata
- Chapters containing sections
- Each section includes:
    - section_id
    - Title
    - page_start, page_end
    - raw_text

This ensures traceability before any processing begins.

Step 2 – Sentence Processing

Each section is processed independently:

- Raw text is normalized
- spaCy is used for sentence boundary detection
- Named entities such as dates, numbers and quantities are extracted
- Every sentence is stored with its source identifiers

This stage converts unstructured paragraphs into atomic units.

Step 3 – Classification

Sentences are categorized using heuristic rules:

- **Rules** – statements with “must, shall, required”
- **Exceptions** – containing “unless, except”
- **Constraints** – limits and thresholds
- **Risks** – conflicts and warnings
- **Contradictions** – opposing statements

- **Facts** – neutral information

Priority ordering ensures exceptions override normal rules.

Step 4 – Section Summarization

For each section:

- Classified sentences are converted into notes
- Importance is estimated using:
    - presence of numbers/dates
    - category type
- Notes are grouped by type

The result is a structured section summary rather than free text.

Step 5 – Hierarchical Aggregation

Section summaries are merged:

- Sections → Chapter level
- Chapters → Document level

No new content is generated. The system only aggregates existing notes to preserve fidelity.

Step 6 – Explainability Layer

An explicit report is generated containing:

- Compression policy
- Information loss estimate
- Traceability verification

This directly satisfies the challenge requirement for transparency.

Methodology

Document Acquisition

PDF Loading

This module is responsible for converting the raw PDF document into a structured machine-readable format. The system first accepts the file path of the target document. A PDF parsing library is used to read the document page by page so that no textual information is lost during extraction. Instead of treating the document as a single block, every page is loaded individually to maintain positional information.

Text Extraction

From each page, the raw textual content is extracted while preserving the original sequence. Special attention is given to handle multi-column layouts, headers and footers which are common in policy and legal documents. The extracted text of each page is stored along with its page number so that later modules can map any statement back to its exact location.

Structural Analysis

Chapter Detection

After raw extraction, the text is analyzed to identify the hierarchical structure of the document. Regular expression patterns and formatting cues such as numbering styles, capitalized headings and indentation are used to detect chapter boundaries.

Each detected chapter is assigned a unique `chapter_id` and its title is captured from the heading line.

Section Segmentation

Within every chapter, further segmentation is performed to identify individual sections. The module searches for numbered sub-headings, bullet markers and line breaks to separate sections logically. This step ensures that long chapters are not processed as a single unit but as meaningful smaller contexts.

Page Range Mapping

For every identified section, the system records the starting and ending page numbers. This page range is critical for explainability because it allows any compressed statement to be traced back to the exact portion of the original PDF.

JSON Structure Creation

Section Object Formation

Each section is converted into a dictionary containing :

- section_id
- Title
- page_start
- page_end
- raw_text

The raw_text field contains the complete textual content of that section without any modification so that downstream processing can work on authentic data.

Chapter Object Formation

All section dictionaries belonging to a chapter are grouped together to form a chapter dictionary. This chapter object stores the chapter_id, title and the list of its sections. The design mirrors the logical layout of the original document.

Document-Level Packaging

Finally, all chapters are combined into a single JSON structure named `doc_structure`. Additional metadata such as document name, creation date or source information is also attached at this level. The final structure follows the format :

```
doc_structure = {
    "doc_id": "doc_name",
    "metadata": {key-value pairs},
    "chapters": [
        {
            "chapter_id": "...",
            "sections": [
                {
                    "section_id": "...",
                    "Title": "...",
                    "page_start": "...",
                    "page_end": "...",
                    "raw_text": "..."
                }
            ]
        }
    ]
}
```

This structured JSON becomes the input for the compression and explainability pipeline handled by the next module.

Design Considerations

Preservation of Original Content

No summarization or filtering is performed at this stage. The objective is only to transform the unstructured PDF into a well-defined hierarchical representation while keeping the text intact.

Traceability Support

By storing section identifiers and page ranges, the module guarantees that every later transformation can be linked back to the source document. This design directly supports the explainability requirement of the project.

Format Independence

The module is designed to work with different document layouts. The use of heuristic detection instead of hard-coded templates makes the system adaptable to contracts, policies and reports with varying structures.

Document Processing Module

Document Input

Our model receives a structured JSON file from the previous stage of the pipeline. This file contains the complete document hierarchy including chapters, sections, metadata and raw text. Each section dictionary stores the fields: section_id, Title, page_start, page_end and raw_text. This structured input ensures that traceability is maintained from the very beginning of the pipeline and no information is detached from its original location.

Sentence Processing

The raw text of each section is first normalized to remove irregular spacing, tabs and bullet symbols. After normalization, the spaCy library is used to perform sentence boundary detection. Instead of using a naïve split on periods, spaCy ensures that abbreviations, decimal numbers and legal references such as “7.3” are not broken incorrectly. Each detected sentence is stored along with its section_id and page range so that every statement remains linked to its source.

Classification Module

Heuristic Classification

After sentence processing, each sentence is passed to the classifier module. Instead of using a black-box model, we rely on transparent rule-based heuristics. Sentences containing keywords such as “must”, “shall” and “required” are labeled as Rules. Sentences containing “unless”, “except” and similar phrases are labeled as Exceptions. Other categories such as Constraints, Risks and Contradictions are identified using dedicated keyword sets.

Priority Logic

A strict priority order is maintained so that critical categories override general ones. For example, if a sentence contains both a rule and an exception phrase, it is marked as an Exception. This design choice prevents misclassification of high-impact statements and keeps the behaviour of the system explainable.

Section Summarization Module

Note Creation

Each classified sentence is converted into an atomic note object. The note contains the original statement, its category, extracted entities, importance level and complete source information. No paraphrasing or rewriting is performed at this stage. The system only reorganizes the original content into structured form.

Importance Estimation

Importance is assigned using deterministic rules. Statements belonging to Exceptions, Risks or Contradictions are marked as high importance. Sentences containing numbers or dates are also treated as high priority because they usually represent limits, deadlines or thresholds.

Section-Level Summary

All notes of a section are grouped by their types such as rules, exceptions and risks. This grouped structure forms the section summary which acts as the basic unit for hierarchical compression.

Aggregation Module

Chapter Aggregation

The summaries of all sections within a chapter are merged without creating any new text. The aggregator only collects existing notes and organizes them at the chapter level. This ensures that no hallucinated information enters the pipeline.

Document-Level Aggregation

After chapter aggregation, a final document summary is produced which contains all decision-critical notes from the entire document. The aggregation process is purely structural and does not involve generative modeling.

Explainability Module

Compression Policy

The system explicitly records what type of content is retained and what is discarded. Rules, exceptions, risks, contradictions, numbers and dates are preserved, while narrative explanations and repeated examples are excluded. This policy is included in the final output for transparency.

Information Loss Estimation

An estimate of information loss is computed based on the proportion of high-importance notes to total notes. This provides a quantitative indicator of how much decision-critical content survived the compression process.

Traceability Verification

Before producing the final result, the system verifies that every note contains valid source references including section_id and page range. If any statement lacks provenance, it is flagged in the explainability report.

Output

The final JSON contains:

- Document ID and metadata
- Chapters with section notes
- Document summary
- Explainability report

Each note includes:

- statement
- type
- importance
- entities
- source (section, page range)

Conclusion

In this project, we developed an explainable compression system aligned with the objectives of Round-3 of the DataForge Challenge. The pipeline prioritizes structure over prose and transparency over brevity. Unlike generic summarizers, our model does not rewrite the document but converts it into machine-readable notes with full provenance.

Key Achievements

- Hierarchical JSON representation
- Decision-critical extraction
- Offline deterministic processing
- Explicit explainability layer

Limitations

- Heuristic classification may miss implicit rules
- Contradiction detection is shallow
- Quality depends on document formatting

Future Work

- Improved semantic rule detection
- Cross-section conflict analysis
- Multi-document extension

References

- pdfplumber Documentation
- Python re library Documentation
- spaCy Documentation
- Python JSON library Documentation
- Dateutil Parser
- Standard NLP Literature