UGP-2 Report: Contract CAD Generator

Deepak Chaurasia: 220330, Odd semester 2025-2026(1st semester)

Automated CAD Generation using LLM:

Automated Contract Appreciation & Compliance and conflict detection (CAD) from Unstructured PDFs and Conflict detection.

PS. Problem Statement

Goal: Given an arbitrary contract PDF (digital or scanned), automatically generate a **correct, traceable, and actionable CAD** along with compliance checks and conflict warnings.

Requirements:

1. **Structured Output**: Conform to a fixed JSON schema (salient_features, payment, submittals, notices, risks, disputes, etc.).
2. **Traceability**: Each extracted fact should include sources: ["page X", ...] where feasible.
3. **Determinism**: Avoid partial/invalid JSON; resist hallucinations. Unknown items must be **null**, not guessed.
4. **Speed**: End-to-end runtime suitable for interactive use (single LLM pass over full text).
5. **Robust Extraction**: Handle both selectable-text PDFs and image-only scans (OCR fallback).
6. **Explainability**: Highlight conflicts (e.g., multiple retention rates) and support compliance Q&A.

1ST Approach: Core component

Stage	Description	Technologies
OCR	Extracts text from scanned or digital PDFs using hybrid OCR	pdfminer.six, PaddleOCR, EasyOCR
Chunking	Splits clauses semantically while preserving headings	sentence-transformers
Embedding	Encodes semantic vectors + hybrid keyword search	HNSWlib, BM25, FAISS
RAG Generation	Retrieves relevant clauses → LLM summarizes sections	LangChain, Hugging Face LLMs
CAD Generation	Structured DOCX output with auto-filled sections	python-docx, WeasyPrint
Chatbot Interface	Contract-level interactive QA	Streamlit, LangGraph

Component	Tool/Library Used
OCR	EasyOCR / Tesseract
Preprocessing	spaCy / NLTK
Embeddings	Sentence-BERT / OpenAI Embedding
Vector Store	FAISS / Chroma
LLM	FLAN-T5 / GPT-4
Framework	LangChain
Visualization	Streamlit (optional UI) (local hosting)

System Architecture Overview

The pipeline follows the **RAG (Retrieval-Augmented Generation)** architecture, consisting of three major components:

1. **Document Preprocessing and Embedding**
2. **Retrieval using Vector Similarity Search**
3. **Answer Generation via a Large Language Model (LLM)**

Flow Chart:

1. Input: Raw contract text or PDF.
2. Preprocessing → Chunking into smaller text segments.
3. Embedding → Convert each chunk into a numerical vector.
4. Store → Save embeddings in a vector database (FAISS or Chroma).
5. Query → User asks a question (e.g., *"When is the payment due?"*).
6. Retrieval → Query is embedded and compared against stored vectors.
7. Generation → Relevant chunks + user question passed to an LLM (e.g., FLAN-T5, GPT-4) to produce the final, human-readable answer.

Document Chunking and Embedding

3.1 Chunking

Contracts are typically long, making it impractical to feed them entirely to an LLM.
We split documents into **semantic chunks** (e.g., 400–800 words) using a RecursiveCharacterTextSplitter.
Each chunk is stored as a **LangChain Document object**, preserving metadata like clause number, section title, and page reference.

3.2 Embedding

Each text chunk is transformed into a **vector embedding**, a high-dimensional numeric representation capturing the meaning of the text.

- **Model used:** text-embedding-3-small (OpenAI) or all-MiniLM-L6-v2 (Sentence-BERT).

- **Output:** A vector (e.g., 384–1536 dimensions).
- **Purpose:** Similar clauses have embeddings close to each other in vector space.

Example:

Text Chunk	Embedding (simplified)
"Payment within 15 days of delivery."	[0.21, -0.34, 0.91, ...]
"Supplier delivers within 30 days."	[0.18, -0.12, 0.22, ...]

4. Retrieval Process

4.1 Query Embedding

When a user asks a question (e.g., *"When should the client pay?"*), it is converted into a vector using the same embedding model.

4.2 Similarity Search

The query vector is compared with all stored chunk embeddings using **cosine similarity** or **dot product**.

Example:

Chunk	Similarity Score
Payment clause	0.91
Delivery clause	0.25
Termination clause	0.12

The **top-k most relevant chunks** are retrieved and passed to the LLM.

4.3 Vector Database

We use:

- **FAISS** for high-speed similarity search and scalability.
- (Optional) **Chroma DB** for easy LangChain integration.

FAISS advantages:

- Handles millions of vectors efficiently.
- Supports GPU acceleration.
- Offers multiple index types (Flat, HNSW, IVF-PQ).

5. Generation Phase (LLM Integration)

Once relevant chunks are retrieved, both the **query** and **retrieved text** are fed into an LLM.

Model Used

- **Google FLAN-T5 (Large)** for text generation and reasoning.
- Alternative models tested: GPT-4 Turbo, Claude 3.5 Sonnet, and Mixtral 8x7B.

Process Example

Input to LLM:

Question: When is the payment due?
Context: The client must make payment within 15 days after delivery.

Output:

The payment is due within 15 days after delivery.

The model generates a concise and factual answer grounded in retrieved contract text.

7. Compliance Checklist Integration

The pipeline can incorporate a **Compliance Checklist** module to automatically verify contractual compliance points, e.g.:

- Payment timeline compliance.
- Delivery and penalty terms.
- Confidentiality and dispute resolution clauses.

Each checklist item can be verified through RAG-based retrieval and LLM-based evaluation.

Approach 3rd

4. System Overview

The pipeline comprises five stages:

1. **Ingestion & Extraction**: pdfplumber for selectable text per page; pytesseract OCR fallback. Page markers --- PAGE N ---are injected to preserve citation anchors.
2. **One-Shot Schema Prompting (ChatGPT)**: Full extracted text → ChatGPT (e.g., gpt-4o-mini / gpt-4.1/gpt5) via response format=Json object, guided by a schema-first instruction. Output: **strict JSON**.
3. **Post-Processing & Rendering**: JSON is validated and used to generate **DOCX** and a **summary PDF**.

4. **Compliance Checking**: For each rule, a compact one-shot prompt returns a JSON verdict with quote and sources.
5. **Conflict Detection**: Regex-based heuristics detect mismatches (payment days, retention %, DLP months, commencement vs site possession, arbitration vs court), mapping to practical categories and quick-fix hints.

High-Level Dataflow

PDF → (pdfplumber/OCR) → Page-aware text → One-shot ChatGPT → CAD JSON → {DOCX, Summary PDF} + {Compliance JSON} + {Conflict Report}

Implementation Approach

The project implements a **one-shot Contract Appreciation Document (CAD) generator and chatbot** that reads contract PDFs (scanned or digital), extracts text, and uses the ChatGPT API for structured reasoning and JSON generation.
The system eliminates chunking errors, broken JSON, and multi-window confusion using a **fast, single-pass prompt pipeline**.

Step 1 — Input and Extraction

- The user uploads a contract PDF through the Streamlit interface.
- The pipeline first tries **pdfplumber** to extract selectable text.
- If the text layer is absent, each page is rendered to an image and OCR-processed with **pytesseract**.
- Output: list of pages + concatenated text with --- PAGE N --- markers.

Step 2 — Chunking and Tokenization

- Extracted text is tokenized using the ChatGPT tokenizer.
- The text is divided into overlapping **token windows** (window_tokens, overlap_tokens) to stay within API limits.
- Each window preserves page headers for accurate source referencing.

Step 3 — LLM Integration

- Replaces Hugging Face calls with **ChatGPT API** (gpt-4.1-mini or gpt-4.1-turbo).
- Prompts are pre-defined (CONTRACT_Q_PROMPT, CAD_JSON_PROMPT, COMPLIANCE_JSON_PROMPT) and sent directly via the ChatGPT completion endpoint.
- Responses are parsed using JSON repair logic (_sanitize_json_like) to ensure valid, complete JSON.

Step 4 — Top-k Window Selection

- Each window is scored by keyword frequency (employer, contractor, payment, law, etc.).
- The **top-k windows** (typically 3–8) are passed to the LLM for CAD generation.
- This heuristic guarantees only the most relevant portions of the contract are processed, reducing latency and cost.

Step 5 — CAD JSON Generation

- For each top window, ChatGPT returns a JSON strictly following the CAD schema (salient features, payment, risks, claims).
- Multiple JSONs are merged by merge_json_objects(), preserving non-null fields and lists.
- Fallback regex extraction ensures that missing fields (e.g., employer name, contract price) are still populated.

Step 6 — Document Exports

- The final CAD JSON is saved and simultaneously rendered into:
  - **DOCX** via python-docx (headings, tables, lists).
  - **PDF summary** via reportlab (A4 layout with page headers).
- Sources are attached automatically using quote-to-page mapping.

Step 7 — Compliance Checker

- Each compliance rule ("Is there a termination clause?", etc.) is run through the ChatGPT API with COMPLIANCE_JSON_PROMPT.
- Regex fallbacks verify evidence if LLM confidence < 0.8.
- Results are stored as a structured JSON report and displayed in a table with rule, status, quote, sources, and confidence.

Step 8 — Conflict Detection

- Independent regex modules check for mismatches:
  - Payment term inconsistencies.
  - Retention percent variations.
  - Defect liability period conflicts.
  - Arbitration vs Court clause overlaps.
  - Commencement vs Site Possession date issues.
- Each conflict is categorized (Contractual / Financial / Quality / Dispute) with a resolution hint.

List of Conflicts:

1. Commencement vs Site Possession Conflict

- **Purpose:** To ensure that the *site possession date* does not occur after the *commencement date*.

- **Logic:** Uses regex to extract all date patterns near “site possession” and “commencement of work” keywords, parses them via dateparser, and compares.
- **Conflict Trigger:** If the latest site possession date is **later** than the earliest commencement date.
- **Severity:** Critical
- **Category:** Contractual Disputes (Site Possession)
- **Resolution Hint:** Align commencement with possession date or provide time-extension (EOT) clause.

2. Payment Term Mismatch

- **Purpose:** Detects inconsistent payment terms (e.g., 14 days vs 30 days after certification).
- **Logic:** Regex for “payment within X days” across all pages.
- **Conflict Trigger:** Multiple distinct payment term durations are detected.
- **Severity:** High
- **Category:** Contractual Disputes (Payment)
- **Resolution Hint:** Standardize the payment period based on the main contract body or latest annexure.

3. Retention Percentage Mismatch

- **Purpose:** Checks for inconsistent retention percentages across clauses or annexures.
- **Logic:** Regex for “retention … X%” within 20 characters of the word “retention.”
- **Conflict Trigger:** Multiple unique retention values (e.g., 5%, 10%) appear.
- **Severity:** High
- **Category:** Financial / Payment
- **Resolution Hint:** Harmonize retention % in payment schedule and update annex.

4. Defect Liability Period (DLP) Mismatch

- **Purpose:** Ensures uniformity in defect liability or warranty duration.
- **Logic:** Regex for “defect liability period” / “warranty period” followed by numerical duration.
- **Conflict Trigger:** Multiple distinct durations (e.g., 12 months vs 18 months).
- **Severity:** High
- **Category:** Quality / Warranty
- **Resolution Hint:** Adopt stricter DLP or clarify which document governs.

5. Arbitration vs Court Conflict

- **Purpose:** Detects contradictory references to both arbitration and court jurisdiction.
- **Logic:** Page-level keyword search for both “arbitration” and “court.”
- **Conflict Trigger:** Presence of both terms in different or overlapping clauses.
- **Severity:** Critical
- **Category:** Dispute Resolution

- **Resolution Hint:** Retain arbitration clause as primary dispute mechanism, with limited court intervention.

6) Performance Security % Mismatch

- **Purpose:** Ensure a single percentage for Performance Security/Bank Guarantee.
- **Detection Basis:** r'(performance (?:security|guarantee|bank guarantee)).{0,80}?(\\d{1,2})\\s*%?'
- **Trigger:** More than one distinct % found (e.g., 5% vs 10%).
- **Severity:** High
- **Category:** Financial / Security
- **Resolution Hint:** Confirm the correct % with the Employer and amend all references and annexures.

7) Liquidated Damages (LD) Rate Mismatch

- **Purpose:** Avoid conflicting LD rates across clauses.
- **Detection Basis:** r'(liquidated damages|LD).{0,120}?(\\d{1,2})\\s*%?'
- **Trigger:** Multiple distinct LD % values detected.
- **Severity:** High
- **Category:** Delay / Damages
- **Resolution Hint:** Harmonize LD rate; prefer the governing clause or latest dated schedule.

8) Liquidated Damages Missing Cap

- **Purpose:** Verify presence of an LD cap/maximum.
- **Detection Basis:** r'(cap|maximum|ceiling).{0,60}?(\\d{1,3})\\s*%?' searched near LD context.
- **Trigger:** LD mentioned but **no cap** found anywhere.
- **Severity:** Medium
- **Category:** Delay / Damages
- **Resolution Hint:** Add explicit LD cap to limit exposure (e.g., 10% of Contract Price).

9) Mobilization Advance Inconsistency

- **Purpose:** Keep one consistent mobilization advance %.
- **Detection Basis:** r'(mobilization|mobilisation).{0,40}advance.{0,80}?(\\d{1,2})\\s*%?'
- **Trigger:** Multiple different advance % values.
- **Severity:** High
- **Category:** Financial / Advance
- **Resolution Hint:** Fix a single % and specify recovery mechanics (from IPCs) uniformly.

10) Escalation Clause Inconsistency

- **Purpose:** Resolve contradictions between “no escalation” and an escalation formula/index.
- **Detection Basis:**
    - “Not applicable”:
      r'(price|cost)\\s+escalation.{0,40}(not\\s+applicable|no\\s+escalation)'
    - Basis exists:
      r'(price|cost)\\s+escalation.{0,80}(WPI|CPI|index|formula|basis|escalation shall apply)'
- **Trigger:** Both patterns appear in the document.
- **Severity:** High
- **Category:** Price Adjustment
- **Resolution Hint:** Decide applicability; if applicable, specify index/formula and delete “not applicable” language.

11) Warranty vs Guarantee vs DLP Overlap

- **Purpose:** Keep consistent durability/defect coverage periods.
- **Detection Basis:** r'(warranty|guarantee|defects liability).{0,60}?(\\d{1,3})\\s*(months?|years?)'
- **Trigger:** Distinct durations detected (e.g., 12 vs 18 months).
- **Severity:** Medium
- **Category:** Quality / Warranty
- **Resolution Hint:** Align all durations; if different scopes, explicitly differentiate the coverage.

12) Termination Notice Period Mismatch

- **Purpose:** Standardize notice requirements for termination.
- **Detection Basis:** r'(termination|terminate).{0,80}?(\\d{1,3})\\s+days'
- **Trigger:** Different notice periods found (e.g., 7 vs 30 days).
- **Severity:** High
- **Category:** Termination
- **Resolution Hint:** Adopt a single notice period and update all cross-references.

13) Multiple Governing Law Jurisdictions

- **Purpose:** Avoid conflicting governing law references.
- **Detection Basis:** r'(governed by|governing law).{0,60}?laws? of\\s+([A-Za-z ,.&-]+)'
- **Trigger:** More than one distinct jurisdiction (e.g., “India” and “England & Wales”).

- **Severity:** Critical
- **Category:** Legal / Governing Law
- **Resolution Hint:** Choose one governing law; remove or subordinate any conflicting references.

14) Force Majeure Missing

- **Purpose:** Ensure force majeure protection exists.
- **Detection Basis:** Absence of force majeure anywhere.
- **Trigger:** No match at all.
- **Severity:** Medium
- **Category:** Force Majeure
- **Resolution Hint:** Insert a standard clause with examples, obligations to notify, and mitigation steps.

15) Force Majeure Weak (No Examples)

- **Purpose:** Strengthen vague force majeure coverage.
- **Detection Basis:** force majeure present but no examples like “act of God, war, riot, epidemic, flood…”
- **Trigger:** Clause exists but lacks illustrative events.
- **Severity:** Low
- **Category:** Force Majeure
- **Resolution Hint:** Add non-exhaustive examples and a clear procedure (notice, timelines, mitigation).

16) Tax Responsibility Conflict

- **Purpose:** Clarify who bears GST/TDS/other taxes.
- **Detection Basis:**
    - Tax mention: r'(GST|TDS|tax(?:es)?)'
    - Responsibility phrasing: r'(employer|contractor|owner)\\s+(?:shall|will|to)\\s+(?:bear|pay|be responsible)'
- **Trigger:** Conflicting parties appear responsible across clauses.
- **Severity:** Medium
- **Category:** Taxes / Commercial
- **Resolution Hint:** Specify split of tax responsibilities (e.g., GST by Employer, TDS by Employer, all other taxes by Contractor) consistently throughout.

Summary Table

#	Conflict Type	Severity	Category	Detection Trigger	Resolution Hint
1	Commencement vs Site Possession	Critical	Contractual Disputes (Site Possession)	Possession date later than commencement date	Align commencement with possession or grant EOT
2	Payment Term Mismatch	High	Contractual Disputes (Payment)	Different “payment within X days” values	Standardize timeline (main body or latest annex)
3	Retention % Mismatch	High	Financial / Payment	Multiple retention % detected	Fix single % and update annex
4	Defect Liability Period Mismatch	High	Quality / Warranty	Multiple DLP/warranty durations	Adopt stricter/gov. duration or clarify scopes
5	Arbitration vs Court Conflict	Critical	Dispute Resolution	Both arbitration and court language	Prefer arbitration; carve-out for interim relief
6	Performance Security % Mismatch	High	Financial / Security	Multiple performance security % values	Confirm %; amend all references
7	LD Rate Mismatch	High	Delay / Damages	Multiple LD % values	Harmonize to governing clause
8	LD Missing Cap	Medium	Delay / Damages	LD present but no cap anywhere	Insert explicit cap (e.g., 10% of Contract Price)
9	Mobilization Advance Inconsistency	High	Financial / Advance	Multiple mobilization advance %	Fix single % and recovery method
10	Escalation Clause Inconsistency	High	Price Adjustment	Both “no escalation” and formula/index present	Decide applicability; remove contradictions
11	Warranty vs Guarantee vs DLP Overlap	Medium	Quality / Warranty	Different durations across terms	Align durations; clarify scope differences
12	Termination Notice Mismatch	High	Termination	Different notice days (e.g., 7 vs 30)	Adopt one period and sync cross-refs
13	Governing Law Multiple	Critical	Legal / Governing Law	More than one jurisdiction	Choose one; delete conflicts

14	Force Majeure Missing	Medium	Force Majeure	No force majeure clause found	Insert standard FM clause with examples
15	Force Majeure Weak	Low	Force Majeure	FM exists but no examples	Add non-exhaustive examples + procedure
16	Tax Responsibility Conflict	Medium	Taxes / Commercial	Different parties tagged responsible	Clarify and align tax responsibilities

Step 9 — Chat Module

- A conversational interface built with **Streamlit Chat** interacts with the ChatGPT API key.
- The user can ask free-form queries ("What is the payment schedule?").
- Responses are generated using either the **sliding-window** or **concatenate** strategy and stored in session memory (Conversation Buffer Memory).

Step 10 — Deployment and Performance

- Deployed as a Streamlit web app with cached model/session state.
- Designed for single-shot API calls → **no multi-pass chunk merging** → faster response (< 4 s avg per question).
- Output formats (JSON, DOCX, PDF) enable downstream review and versioning.

Approach 3rd

How This Extends Specifically for small LLM

When you say:

"Exactly same as 2nd one but if we are using small LLM then we should send data in chunk and rest exactly the same. extend this for our google flant large model"

Concrete execution for flan-t5-large:

1. **Do not send full 200 pages at once.**
2. Use:
    a. window_tokens ≈ 384–448
    b. overlap_tokens ≈ 64–96
3. **CAD Generation**:
    a. Use CAD_JSON_WINDOW_PROMPT with flan.
    b. Run per window.
    c. Parse each response to JSON.

    d. Merge with merge_json_objects.
4. **Compliance Check**:
    a. Use COMPLIANCE_JSON_PROMPT with flan.
    b. Run per rule per window (with keyword-filtering to reduce calls).
    c. Aggregate to one row per rule.
5. **Conflict Detection**:
    a. Pure regex/heuristic on full text; independent of model; unchanged.

Nothing "serial Q&A over every single field" is hard-coded; instead:

- Each window returns a full CAD-shaped JSON.
- Merge function composes the final CAD.
- This keeps the logic the same as your Approach 2 (structured JSON), but adapted for flan's short context.

Limitations:

1.Privacy policy: since Contract document is legal document so its risky to upload directly open LLM which may result in training on that dataset and will to public that will no longer be private document.

Legal defensibility: LLM outputs are non-deterministic and can hallucinate or miss clauses → weak as evidence in disputes/audits.

Ambiguous sourcing (OCR noise, page merges) can undermine citations.

2.You cannot feed very large documents (if tokens limit is crossing, we will have to send it in chunks and use similarity score and find most relevant chunks and then find clause and answer inside that chunks) . BoQs can overflow context → truncation, missed signals, or costly multi-pass windowing.

3.Variance in Contract: Contracts differ (FIDIC, EPC, O&M, supply, PPP, country-specific law). A single prompt struggle to generalize every definition/term.

How to overcome:

1.We can use Enterprise API Key (To avoid Training on your legal documents).

2.Use chunking first then Upload with Faiss Database.