AI Solution Design Assignment – Comprehensive Documentation

Author: Darshit Pithadia
Project Goal: Converting unstructured textual data into structured JSON aligned to a complex, nested schema.

---

1. Problem Statement

In modern enterprise workflows, a recurring challenge is converting unstructured data—such as documents, emails, and policy drafts—into structured formats required for downstream systems. This task is typically manual, error-prone, and time-consuming, especially when dealing with deeply nested or dynamic JSON schemas.

The specific business scenario involved parsing a chain of internal and external emails to extract the final list of requirements and then structuring them into a strict JSON schema. The schema in question can contain more than 150k tokens, over 100 nested objects, enums, and multi-level arrays, with an expected support for large unstructured inputs (from 50 pages up to 10MB CSVs).

The solution should:

- Parse large unstructured inputs.
- Handle deep, complex schema structures with minimal assumptions.
- Output data in a schema-compliant JSON format.
- Flag low-confidence or ambiguous fields for human review.

2. Solution Overview

The architecture was designed as a **modular and extensible LLM-driven extraction pipeline**. It transforms unstructured documents into structured outputs, using intelligent chunking and field-wise extraction aligned to a flattened schema format. Key highlights:

- **LLM-based field-level extraction**: Prompts the model for one field at a time using schema + document context.
- **Scalable for large documents**: Through chunking and caching.
- **Modular Design**: Allows easy future adaptation (e.g., multiple LLMs, parallelization).

- **Validation & Confidence Scoring**: Final JSON output is schema-validated and scored per field.

---

3. Methodologies Explored

3.1. End-to-End Extraction Using Full Schema and Document

Approach: Provide the entire schema and the full document to the LLM and ask it to return the full structured JSON output in one go.

Outcome:

- Failed due to context window limitations (most models can handle ~8k–32k tokens).
- JSON outputs were incomplete and often invalid.
- No confidence scoring possible per field.

Trade-off:

- Simpler logic but completely non-scalable.
- Only viable for very small schemas and documents.

3.2. Field-Wise Extraction Without Chunking

Approach: Prompt the LLM per schema field, using the entire document as context.

Outcome:

- Improved precision for small documents.
- Failed on larger documents due to context limits again.
- Repetition of entire document per prompt increased cost.

Trade-off:

- Poor compute efficiency.
- Missed out on contextual continuity for large documents.

3.3. Semantic-Aware Chunking (Idea-Level Prototype)

Approach: Use embedding similarity or sentence segmentation to split documents based on topic or semantic boundaries.

Why Not Used Yet:

- Requires embedding models, semantic search setup (e.g., FAISS/Chroma).
- Would've increased development time significantly.
- Postponed due to timeline constraints.

Future Scope:

- Would significantly improve extraction accuracy.
- Helps ensure field-specific context is used for extraction.

3.4. Final Chosen Approach – Modular Chunking + Flattened Schema + Field-Level Prompting

Why This Approach Was Selected:

- Balanced simplicity with scalability.
- Allowed handling very large inputs and deeply nested schemas.
- Modular structure enables easy debugging, optimization, and extension

---

4. Implementation Log (Step-by-Step)

4.1. Initial Setup

- Environment configured in Python.
- Installed dependencies: `jsonschema`, `tqdm`, and Groq's Python SDK.
- Command-line interface created for input file paths (schema and text).

4.2. Schema Processing

- Flattened the JSON schema recursively to identify **leaf fields** only.

- Each leaf field had:
  - Full path (e.g., `project.details.client.name`)
  - Type (e.g., string, boolean, enum)
  - Description and constraints.

Challenge: Handling `patternProperties`, `items`, and dynamic keys.
Solution: Used generic placeholders like `$pattern$` and `[i]` in flattened paths.

4.3. Document Chunking

- Split long documents into overlapping chunks (e.g., 1500 tokens with 500-token overlap).
- Prevents truncation across sentence or section boundaries.

Trade-off: Redundant processing but better context retention.
Improvement: Adaptive chunking based on semantic headers can be added later.

4.4. Prompt Construction

- Custom prompts created per field including:
  - Schema context
  - Document context (chunk)
  - Clear instructions to return only value or null
- Includes constraints like enums or data types.

Challenge: LLMs sometimes return extra formatting or commentary.
Solution: Used strict formatting instructions.

4.5. LLM Extraction Logic

- Each field extracted from each chunk using the prompt.

- Cached previous results to avoid redundant LLM calls.
- Early exit once confident non-null value was found.

Confidence Score Heuristics:

- Full match and type-correct: High

- Null/none: Low
- Enum adjusted: Medium

Error Handling:

- Rate-limiting managed with `sleep` and retry intervals.
- Fall back to null for failure cases but continue the process.

4.6. Output Assembly

- Reconstructed the nested JSON from flattened paths.
- Schema validation done using `jsonschema`

Limitation: Current logic has basic handling for arrays and dynamic keys.

---

5. Trade-offs and Strategic Considerations

Area	Trade-off Made	Strategic Justification	Future Scope
Document Chunking	Fixed-size overlapping	Easy to implement, avoids context loss	Replace with semantic or adaptive chunking
Schema Handling	Skipped advanced logic (e.g., `allOf`, `not`)	Focused on basic and moderate schema support	Full Draft 2020-12 JSON Schema support
Confidence Scoring	Rule-based	Lightweight and explainable	Use token probabilities or LLM-native confidence
LLM Selection	Used Groq API via DeepSeek Llama	Balanced performance and cost	Add fallback and dynamic model selection
Concurrency	Sequential field processing	Easy to manage and debug	Switch to async parallel extraction

Output Reconstruction	Basic handling of patternProperties	Simplified for demo purposes	Add key prediction + recursive object extraction

---

6. Future Scope

6.1. Extraction Enhancements

- Implement few-shot prompting for edge cases.
- Handle object and array fields recursively.
- Multi-modal input (e.g., tables/images in docs).

6.2. Performance Optimization

- Parallel LLM calls using async or multithreading.
- Smart batching: extract multiple fields per chunk if tokens permit.

6.3. User Experience

- Add Web UI for document upload and result preview.
- Add field-level confidence indicators and manual review interface.

6.4. Extensibility

- Modular plugin-based schema and prompt processors.
- Vector DB integration for schema and doc retrieval.

---

7. Constraints

Constraint	Mitigation
LLM token limits	Chunking mechanism
Rate-limiting by API	Retry logic + caching

Schema complexity	Flattening, prioritization
Time-to-build	Prioritized core logic over UI or advanced retrieval
Limited memory	Process documents in parts, avoid loading entire context

8. Conclusion

The developed solution successfully demonstrates the conversion of unstructured documents into structured JSON outputs using schema-aware LLM interaction. While the first iterations exposed the complexity of balancing schema depth, document size, and LLM limitations, the final modular system offers:

- High adaptability across different schema formats.
- Effective large-document handling.
- Confidence-aware extraction flow.

The approach lays a solid foundation for future enhancements including semantic chunking, recursive object extraction, web interface deployment, and human-in-the-loop feedback.