Nitesh Pandey (PandaDoc Task)

Slide 1: Our Approach & Key Assumptions

Key Assumptions

- **PDFs are long**
  - LLMs have a limit on how much text you can send at once, so we must chunk.
- **Dynamic schema**
  - The LLM sees all schemas in the prompt and can choose or fallback to generic.
- **LLM calls**
  - We combined both the classification (type) and metadata extraction into a single LLM call per chunk, to minimize API usage and latency.
- **Prompt**
  - The user prompt includes detailed instructions and the full schema definitions so the model knows exactly what JSON shape to return.
- **JSON repair**
  - Models sometimes output near JSON, so we run simple heuristics (strip fences, fix commas) before parsing.

Approach

- **Load & Read PDFs**
  - Extract all text from each PDF using PyPDF2.
- **Break into Chunks**
  - If a contract is too long to fit in one LLM call, we split it into 25k token size.
- **Schema-Driven Extraction**
  - We defined a generic schema plus specific schemas (NDA, Service Agreement, etc.).
  - The LLM picks the best schema, if nothing matches, it uses the generic one.
- **Single LLM Call per Chunk**
  - In one prompt we ask the LLM to both classify the contract type and extract all metadata.
- **Merge & Clean**
  - We combine chunk outputs, fix any broken JSON, and remove duplicate parties/clauses.

Slide 2: How We Use the LLM & Prompts

- **LLM**
  - We used GPT-4o, which has a limit on how much text it can handle at once. So, we decided to break the PDF into smaller parts (chunks), get a JSON output for each part, and then combine all the parts into one final JSON file.

- **Dynamic Prompt**
  - Before sending text, we build a JSON list of all our schemas and insert it into the prompt. So that model can pick which schemas suit more to the content. If nothing applies then fall back to generic schemas
- **Prompt Structure**
  - It consist of four main parts
    - Identify contract type
    - Pick schemas
    - Extract
    - Output rules
  - Ask for _only valid JSON_ following the chosen schema.

- JSON Repair
  - If the LLM's output isn't perfect JSON, we run simple rules (strip code fences, fix commas) to clean it.

Slide 3: Sample Output & What Didn't Go Perfectly

Example JSON Output

```
{
  "contract_type":"nda",
  "parties":[{"role":"Party","name":"Acme Corp."}],
  "effective_date":"2023-01-01",
  "clauses":[
    {"name":"Confidentiality","present":true}
  ],
  "custom_fields":{}
}
```

Limitations We Saw

- The LLM sometimes invents a clause that isn't there. It goes with the hallucinations.
- Big contracts get split into lots of chunks, which slows things down. It also causes more duplicate entries in the JSON. It's hard to identify these duplicates because sometimes the field names are the same, but the values or content are different.
- JSON-fix rules don't catch every formatting mistake.
- Anything not in our schemas ends up under custom_fields (so we might miss some details).

Slide 4: Accuracy, Reliability & Next Steps

- **Accuracy & Logs**
  - We log how many fields get filled vs. total fields to track completeness.
  - Errors in parsing fall back to a minimal “unknown” template so the pipeline never crashes.
- **Reliability**
  - Chunking and merging works well for most large contracts, but sometimes important context between chunks gets lost.
- **Future Improvements**
  - **Human review UI** – let a human quickly correct or confirm extractions.
  - **Schema auto-generation** – detect new fields and suggest schema updates.
  - **Grounding with definitions** – feed clause definitions into the prompt so the LLM doesn’t hallucinate.
  - **Automated testing** – compare outputs against a labeled dataset to measure precision/recall.