Drop 1: LangChain in Production (Document Loader)

1. When to use Built-in vs Custom Loaders?

Built-in loaders are already available in LangChain for common formats like PDF, CSV, JSON, websites etc.
Use them when your data is in a standard format and the loader works fine out of the box.

Custom loaders are needed when your data is not "standard."

For example:

- You're pulling data from an internal API that returns mixed JSON + XML
- You have logs in `.txt` but with special formatting (timestamps, system messages)
- You want to pre-clean / enrich the data while loading (e.g., remove headers, add metadata)

Example:
In one project we had to process **chat transcripts**. They were stored in a database with custom formatting (`[Agent: Moiz] Hello` etc). Built-in loaders couldn't parse them properly. I made a custom loader to strip out names, add `role=user/agent`, and then return clean text chunks.

---

2. Handling Large Files + Rate Limits

Biggest issues:

1. Large PDFs (hundreds of pages) → Loader crashes or memory overload
2. APIs (like Google / OpenAI) → Daily request limit

Solutions:

- **Chunking**: Instead of loading one huge document, split into smaller chunks (e.g., 500–1000 words). This makes embeddings + retrieval faster.
- **Streaming**: Don't read the full file into memory. Load part by part.

- **Batching API Calls**: Send requests in small groups, add delays if API throws “rate limit exceeded.”

Example:

I had a 300-page company policy PDF. Directly loading made the pipeline super slow and embeddings file was >1GB. I solved it by splitting the document into **sections by headings** instead of blind chunks. That reduced useless text and cut size by ~40%.

Another time, while scraping 100+ webpages with a loader, Google API blocked me after 60 requests. I fixed it with a **retry + exponential backoff** logic. Loader tried again after a few seconds automatically.

---

3. Code Snippets + Edge Case Fixes

Some common edge cases beginners face:

1. **Corrupted PDFs** → `PyPDFLoader` fails.
   - Fix: Try `pdfplumber` or `unstructured` loader as fallback.
2. **Empty Metadata** → You don’t know where a chunk came from.
   - Fix: Always add filename / source URL into `metadata`. It helps in debugging later.
3. **Encoding Errors** (weird characters like `â€™`) → Some CSV/JSON have strange encodings.
   - Fix: Open files with `utf-8` or `latin-1` explicitly.

Example:

We were parsing resumes. Some PDFs were **scanned images**, so loader returned blank text. Instead of discarding, we added an **OCR step (pytesseract)** in the custom loader. That way we didn’t miss out on candidates with image-only resumes.

Another time, a CSV loader broke because one column had commas inside the text. I fixed it by explicitly setting `delimiter=","` and wrapping quotes in the loader config.

Core Concepts in LangChain

Before diving into data loaders, let's understand the basic building blocks:

Component	What it does	Real-world example
LLMs	AI models that generate text	ChatGPT, Llama, etc.
Prompts	Templates for communicating with LLMs	Asking a question in a specific format
Chains	Link multiple components together	Reading data, summarizing it, then answering questions
Document Loaders	Tools to bring in outside data	Reading PDFs, websites, or databases
Indexes	Efficient storage of data	Creating a searchable index of documents