Adding Structure to Your Text Data

Anatomy of a Text Processing Pipeline

ABIGAIL HADDAD
MAR 03, 2025

I thought I was doing a bunch of different things over the past couple of years, but turns out they were mostly variations on a theme: **Processing Unstructured Text D**

Taking USAJobs posts and extracting names of software tools and programming languages? *Processing Unstructured Text Data.*

Creating a website that uses ModernBERT to predict if something is edible? *Proces Unstructured Text Data.*

Writing grading rubrics for synthetic data? Pulling content from PowerPoints to create one-slide summaries? All of these are *Processing Unstructured Text Data.*

What Defines This Workflow?

In each case, I have a set of documents and I'm asking the same questions about ea of them. For instance: w*hat software tools and programming languages appear in each jo posting?*, *what are the key talking points from these slides?*, or *did this set of directions generated by an LLM contain a particular step*?

This is distinct from a RAG chatbot, where users ask an open-ended set of question In my workflows, we're looking for the same pieces of information from each of the documents, and that might change over time, but we don't have to answer just any question that a user might ask—and there often aren't users directly interacting wi the system, just reviewing output or taking an action based on the results.

That said, part of the evaluation portion of RAG systems does fit into this pattern. When you're using tools like RAGAS to evaluate your RAG pipeline, you're taking chunks of text that were passed to the LLM via the retrieval step, as well as the LL output, and you're asking questions like "is the output properly grounded in the so text?" That evaluation process is fundamentally the same workflow—processing unstructured text with the same question applied repeatedly.

This pattern clearly keeps showing up in my work—but it took me awhile to realize and to realize that there were things I'd learned from one project that I was bringin to other ones.

In this post, I'll break down this workflow pattern, discuss some tools for each of th steps, and share what I've learned about structuring these projects effectively.

The Three-Part Pipeline

These projects typically follow a three-part structure:

- **Get text:** This might involve getting text directly from users, pulling it from PowerPoints or PDFs, or collecting it from an LLM that you're evaluating.
- **Process that text:** This might involve an LLM, but it also might involve regular expressions for pattern matching, BERT for classification, or another single-purpose model like language classification or translation.
- **Do something with the results**: Sometimes this is just making heat maps or otl result visualizations. Sometimes I'm displaying results for a user, or populating PowerPoint slides or other templates with structured data.

Let's explore each stage of this pipeline in more detail, looking at the specific tools and approaches I've found useful.

Get Text

Sometimes the text might be given to you directly, sometimes you might have to pu out of a document. But if a file has text, you can get it. There are packages for nearl any format.

- **Microsoft Office Files**: python-docx for Word documents, python-pptx for PowerPoint files, PyPDF2 or pdfplumber for PDFs
- **Structured Data**: pandas for Excel and CSV files, json for JSON data
- **Web content**: beautifulsoup4 and requests for HTML and XML
- **Specialized Formats**: Libraries for email archives like libraton, pytesseract for OCR, and various other packages for specialized formats you might encounter your specific domain

There are also a couple of use cases where "getting the text" involves an LLM.

The major one is when the text you're processing is synthetic text data: text generat by an LLM that you're then going to evaluate. I'll talk about that in the next section because I use the same tools to interact with LLMs whether it's for getting text or processing text.

The other is a special case of "getting text" where you're extracting images and gett a multimodal model—I use Gemini—to "convert" them into a usable text descriptic In the project where I'm doing this, I use a specific prompt where I tell the model what I'm interested in: any text that's in the image, plus any data or anything else th conveys an overall meaning. I also say what to ignore, like colors and formatting—unless they're communicating data.

I haven't tested this yet to see how well it's working, but if I continue to develop tha project, I should add in more formal testing of that step, because it's not the same a just pulling actual text out of documents—it's a machine learning model. There are other text processing tools which are also machine learning models: for instance, if you're OCRing text or transcribing text from audio. Depending on your use case, yc

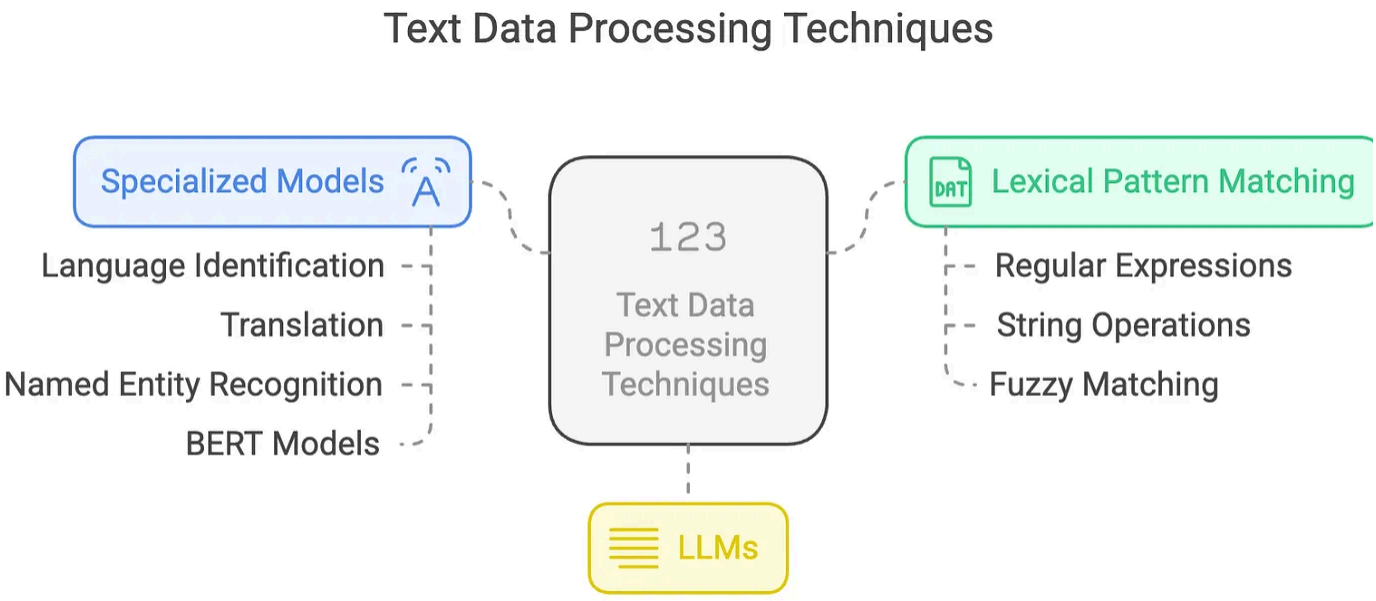
may treat them like settled technology, or you may need to test their performance w your specific use case.

Process Text

Once you've gotten the text, the next step is processing it to extract meaningful information. This might be actual "extraction", in the sense of finding a substring i your text. Other times, this is a classification task: for instance, labeling output fror RAG as "grounded" or "not grounded" in the text chunks that were passed to it. Finally, this might be generative: synthesizing text to produce something like a summary.

Just like with getting text, there are specialized tools depending on what you're tryi to accomplish:

- **Lexical Pattern Matching**: Regular expressions, string operations, and fuzzy matching tools for extracting structured data or finding specific patterns in ter of the characters in the text.
- **Specialized Models**: This is a huge bucket that includes language identificatio and translation, Named Entity Recognition (that's finding entity classes like names or places or people—or software tools and programming language), or BERT or related models for classification. You might use these models out of tl box, or you might fine-tune them on your specific data.
- **LLMs**: It's rare that I have a workflow for processing unstructured text data tha doesn't involve LLMs. The lexical pattern matching and specialized models I u typically are add-ons to the main LLM workflow, which is sometimes classification and sometimes generative.



I wrote more here about the factors that affect whether I choose an LLM for any particular text processing problem. But when I am, the issues that affect both whicl models I use and which tools I access them with are:

- **The need to run models locally**: I can't always send my data out to a proprietar LLM.
- **Swapping out models/providers**: I want to switch models and providers easily i my code.
- **Getting 'structured output'**: I frequently want the LLM to return a JSON objec with the output structured in a particular way—for instance, a list of programming languages, or a PASS or FAIL in one field and an explanation of t reasoning in another field.

There are many different tools that provide these features; I'm not doing a comprehensive rundown here. But the two I've used the most are Ollama, for runni models locally, and LiteLLM, which can work either with non-local models or via Ollama for local ones. They both make it easy to swap out model names, which is especially good for testing or just for developing using a cheaper or faster model. A

they both support structured outputs. (Although not every model allows this, no ma what tools you're using to access it.)

Do Something With The Results

After processing the text data, we need to do something meaningful with that information. What we do with the results is just as varied as the previous steps.

- **Showing it directly to a user**: I don't have this workflow very much, but you mi have a tool that's designed to directly take input from users and return the mod answer directly, with no other context—or maybe with a probability score of so sort.
- **Summarizing it**: Sometimes you don't need to show anyone the output directly but you need to summarize it. For instance, if you're assessing different LLMs f their ability to remain grounded on different types of questions, you might wan heat map showing each LLM's performance on some set of test questions.
- **Populating a template**: I'm starting to get into this with a project where I populate a PowerPoint template with various fields that an LLM has produced the previous step. PowerPoint doesn't make this easy, but there are other file formats that are more amenable to being populated this way.

The Fourth Step: Monitoring

For projects that aren't one-off but ongoing, you'll likely want a monitoring process that flags when something breaks and gives you a path to fix it—for instance, for LL classifiers, by switching models or tweaking prompts.

I haven't built this out yet, but I see two main approaches. One is leveraging natura labeling processes, where your models predict something that can be verified later through real-world outcomes or user actions, allowing your monitoring system to track performance using standard metrics.

But without that—and I suspect in most cases of this workflow, you won't have that you'll need periodic sampling and review of outputs. Maybe that can be done with a bigger, more capable LLM—but then, you have to monitor that one as well! More likely, you need human review.

The key here is being strategic about which outputs to select for review, maximizin the value of reviewer time. This strategy will be specific to your use case—for insta prioritizing examples where confidence scores are borderline, in cases where you h confidence scores, or sampling from new types of documents your system hasn't se before. Either way, this feedback loop is important for keeping text processing pipelines reliable over time.

Design Patterns and Ongoing Challenges

As these projects have evolved, I've developed certain patterns and practices to mak the work more manageable. I'm not a software developer and I'm not writing softw —but I still need code I can debug and build on.

Here are some practices I've adopted to varying degrees with that in mind:

- **Modular Components**: I structure my code so text extraction, processing, and output generation are separate components that can be run independently. Thi particularly important because the text processing step typically takes the long and sometimes costs money, if I'm hitting an external LLM. I don't want to re- it every time I'm tweaking my heat map formatting.
- **Ease of Testing**: I need to be able to easily vary LLM models, and sometimes al parameters like token count, temperature, or prompt, to observe how they affec results. The simplest approach is using lists of models and temperatures in the code itself. For more complex scenarios, I use text or JSON files to populate different fields of a class which handles the details of the LLM call.

- **Logging**: I've started structuring my code so that, each time anything runs, the results output into a new folder with a name that reflects the date and time it started running. This folder contains not just each of the outputs but also a log with everything that happened. With LLMs, the benefit of generating these kin of log files is higher, because if something breaks, you can dump the log file int an LLM and see if it immediately spots what broke.

```
text_pipeline/
├── main.py           # Runs the full pipeline or individual components
├── extract.py        # Extracts text from sources, outputs to data/raw.json
├── process.py        # Processes text using LLMs, outputs to data/processed.jso
├── visualize.py      # Generates heatmaps from processed data
├── data/
│   ├── raw.json      # Intermediate storage for extracted text
│   └── processed.json # Intermediate storage for processed results
├── results/          # Directory for final visualization outputs
│   └── heatmap.png   # Example visualization output
└── README.md
```

This is one way to structure a very simple repo

That said, I'm still running into challenges where I feel like I need more backgroun software development.

The biggest one is that I can code quickly with LLMs, so I try stuff out that I would otherwise, like spinning up a new classifier or specialized model and getting it to w on a sample string. But that means there's a lot of code, and it's not necessarily structured in a way that's well-architected for my actual workflow.

For instance, if I have three different LLM classifiers, plus several smaller or specialized models—and I want to be able to easily change which of those I'm runn as part of my text processing pipeline, as well as swap out prompts for my classifier order to test them—my code needs to be really modular and well-structured not jus across each step of my workflow but within the text processing step itself.

This Pattern Is Everywhere

Now that I'm looking for it, I'm seeing this "process a stack of documents to extrac the same set of information" pattern everywhere, in other peoples' use cases as well mine: finding fraudulent billing in medical records, determining what project someone is working on for billing purposes based on their computer activity, writir new proposals based on previous work and new client requirements.

What makes this pattern distinct is that you're not answering ad-hoc questions – you're systematically extracting the same information or making the same judgmen across a set of documents. Was there fraud? Which project should this bill to? Wha expertise do we have that's relevant to a new contract?

And even though each specific implementation might feel really different, having t mental model is helpful as far as not just re-using code, but re-using design pattern If you're working with unstructured text data, I hope identifying this pattern helps too.

Got a project involving unstructured text or image data you'd like help with? I'm available part-time consulting and coding. You can reach me at abigail.haddad at gmail.

2 Likes

Discussion about this post

Comments Restacks

Write a comment...