

Processing Text Data: Should I Just Throw It Into an LLM?

ABIGAIL HADDAD

FEB 28, 2025



3



Share

The other day, I had to parse the messiest text data I've ever seen. It was like Frankenstein's Data Monster: each system it went through before getting to me stitched on its own mismatched parts and bizarre appendages. The original fields I needed to extract were in there somewhere, but I had to carefully remove all these grafted-on pieces first in order to find them.

It took me a couple of hours but I got most of the way there. I used multiple Python packages for processing [markup languages](#), plus [regular expressions](#) to clean up leftover tags and characters. It wasn't perfect, but it got me the core fields I needed and I can get the additional fields if necessary.

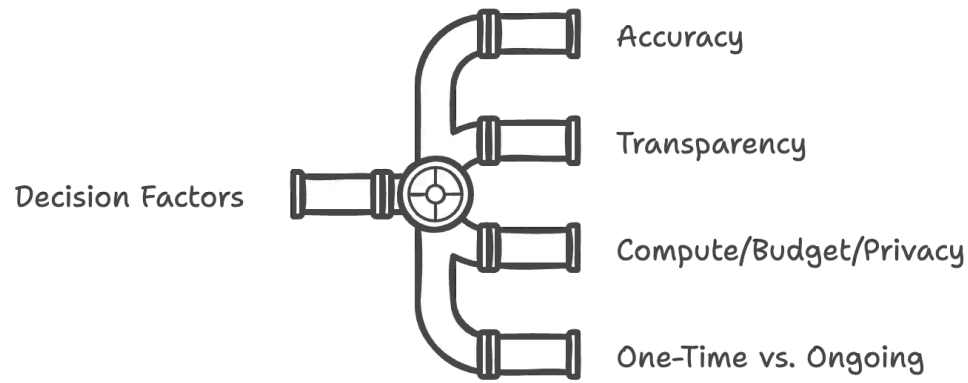
Thanks for reading The Present of Coding!
Subscribe for free to receive new posts and
support my work.

But why didn't I just throw it into an LLM? Surely, an LLM with instructions about what fields to extract would have been able to do it? *Why bother with single-purpose tools for text processing when there's an increasingly cheap and capable LLM Swiss Army knife that promises to cut through any problem?*

I have reasons! In this post, I'll discuss the factors that go into my decision about whether to use an LLM for data processing tasks like cleaning text, classification, or

Named Entity Recognition, where there are also non-LLM options.

Should I Use an LLM?



These factors are:

- **Accuracy:** *How accurate does my solution need to be? How does each method perform both in terms of how often it fails and how it fails?*
- **Transparency:** *How much do I need to understand what my model is doing and why? Can I predict the output given an input? How important is it that the process be deterministic—that given the same input, I'll always get the same output?*
- **Compute/Budget/Privacy:** *Do I have enough compute? Can I send data to an external API, and if so, what's my budget?*
- **One-Time vs. Ongoing:** *Am I processing one dataset that exists already and is fully visible to me, or does this need to handle future data on systems I won't directly observe?*

Accuracy

How accurate does my solution need to be? How does each method perform both in terms of how often it fails and how it fails?

Different problems have different acceptable levels of accuracy and ways it's ok for them to fail.

And there aren't great heuristics for when an LLM will outperform other approaches—for instance, for some classification problems other methods will do quite well, and for others they won't.

After you've decided what to try, how do you determine relative accuracy rates for a method? For my text cleaning process, I was able to get to a working solution quick enough that I didn't bother trying out anything else. I also didn't build tests because I could eyeball that it was working—which was good enough for a proof of concept.

But when I need to compare solutions in areas where I can't immediately tell how well each of them is working, I write tests (not in a formal [unit test](#) framework) and then evaluate how different methods perform. And even if I'm *definitely* using an LLM, I need this testing process anyway, because I'll test different models, prompts, and parameters like token count or temperature, and I want that to be simple and automated.

Transparency

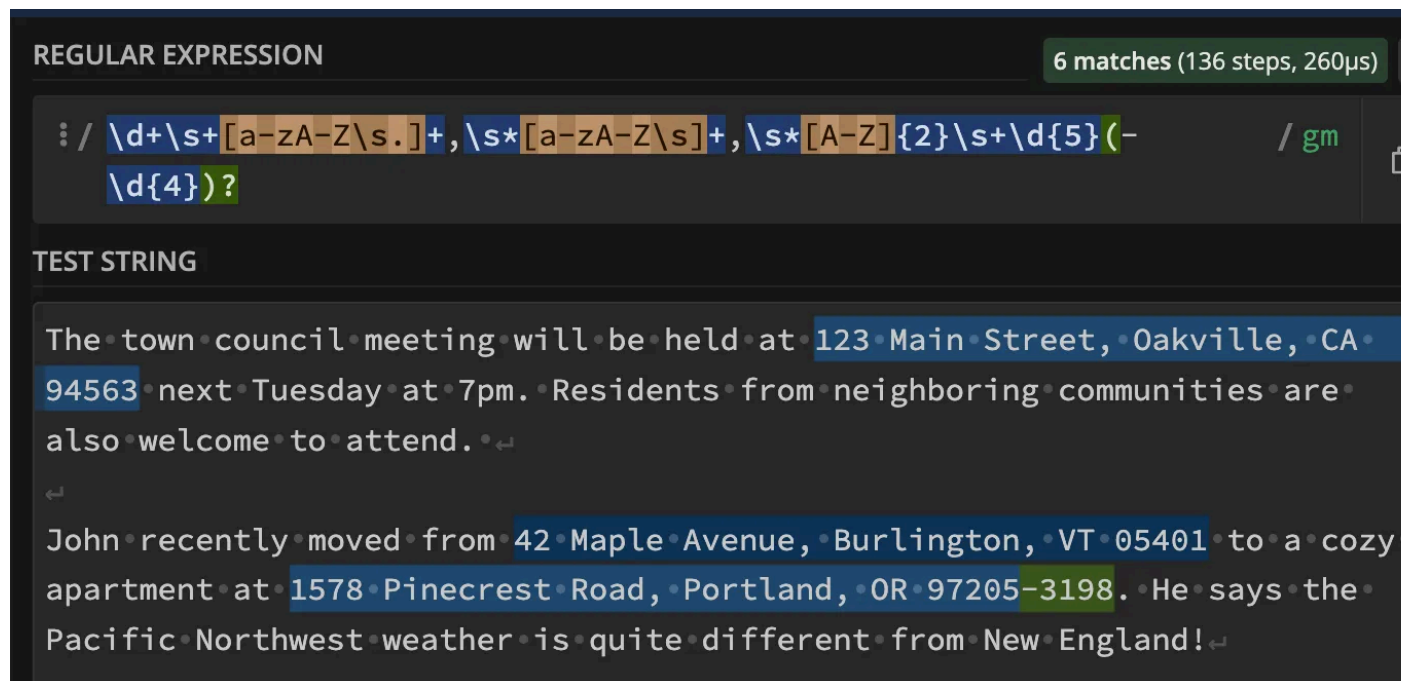
How much do I need to understand what my model is doing and why? Can I predict the output given an input? How important is it that the process be deterministic—that given the same input, I'll always get the same output?

These issues—transparency, explainability, determinism—are distinct but overlap. Some methods are easily understandable, predictable, and deterministic—like a regular expression that finds specific text patterns. Some methods are none of these: a proprietary LLM, for instance—though for a specific use case, it might be essential to be deterministic, and [you can test this](#).

Other methods have some properties but not others: [BERT](#), a smaller language model is deterministic, but I won't understand its behavior like I would regular expression there's no single, explainable pattern it's finding. Can I predict what my BERT model will output based on previous performance better than I can predict an LLM? It depends on my problem.

One reason to care about these factors: the better handle I have on how my model works, the better I can predict what kinds of new data will break my tools and how they will break.

For instance, in my text processing example, an LLM would likely be more flexible to format changes than what I actually wound up building: if my data starts going through a totally different system, and no one tells me, my current solution isn't going to handle it well. But if data format changes break my current method, the failure will be obvious: the fields simply stop coming through! I can easily write tests to detect this. With less transparent methods, I might not know when or how they'll fail, because there could still be something that looks like correct output.



The screenshot shows a web-based regular expression tool. At the top, it says "REGULAR EXPRESSION" and "6 matches (136 steps, 260µs)". The regex pattern is: `/\d+\s+[a-zA-Z\s.]+\s*,\s*[a-zA-Z\s]+\s*,\s*[A-Z]{2}\s+\d{5}(-\d{4})?/gm`. Below the pattern, it says "TEST STRING". The test string contains two lines of text. The first line is: "The town council meeting will be held at 123 Main Street, Oakville, CA 94563 next Tuesday at 7pm. Residents from neighboring communities are also welcome to attend." The second line is: "John recently moved from 42 Maple Avenue, Burlington, VT 05401 to a cozy apartment at 1578 Pinecrest Road, Portland, OR 97205-3198. He says the Pacific Northwest weather is quite different from New England!". The matches are highlighted in blue and green.

Regular expressions can identify text patterns like addresses and phone numbers

I might also care about transparency and related factors because they're inherently important to my use case.

Compute/Budget/Privacy

Do I have enough compute? Can I send data to an external API, and if so, what's my budget?

If my data can't leave my environment due to privacy or regulatory concerns, I'm limited to tools I can run in that environment—which may rule out proprietary LLMs or even larger open-source models. Because of this, for local processing, I lean toward non-LLM solutions when possible. I can run some models via [Ollama](#), but except for small models with low volume, it takes time. I try to parallelize work so I'm not waiting for code to run, but longer processing times inevitably slow me down when developing.

By contrast, for data that can be sent externally and use cases where I might want an LLM, I frequently try the gpt-4o-mini model before trying anything else, because it's incredibly cheap and surprisingly capable.

Text tokens

Price per 1M tokens · Batch API price ☐

Model	Input	Cached input	Output
<code>gpt-4o</code> ↳ <code>gpt-4o-2024-08-06</code>	\$2.50	\$1.25	\$10.00
<code>gpt-4o-audio-preview</code> ↳ <code>gpt-4o-audio-preview-2024-12-17</code>	\$2.50	-	\$10.00
<code>gpt-4o-realtime-preview</code> ↳ <code>gpt-4o-realtime-preview-2024-12-17</code>	\$5.00	\$2.50	\$20.00
<code>gpt-4o-mini</code> ↳ <code>gpt-4o-mini-2024-07-18</code>	\$0.15	\$0.075	\$0.60
<code>gpt-4o-mini-audio-preview</code> ↳ <code>gpt-4o-mini-audio-preview-2024-12-17</code>	\$0.15	-	\$0.60
<code>gpt-4o-mini-realtime-preview</code> ↳ <code>gpt-4o-mini-realtime-preview-2024-12-17</code>	\$0.60	\$0.30	\$2.40
<code>o1</code> ↳ <code>o1-2024-12-17</code>	\$15.00	\$7.50	\$60.00
<code>o3-mini</code> ↳ <code>o3-mini-2025-01-31</code>	\$1.10	\$0.55	\$4.40
<code>o1-mini</code> ↳ <code>o1-mini-2024-09-12</code>	\$1.10	\$0.55	\$4.40

There's huge price variation by model with OpenAI

For my text processing problem, privacy and compute needs were a factor—and my regular expression/Python solution runs locally, almost instantly, and costs nothing per run. An local LLM approach would have been much more resource-intensive.

One-Time vs. Ongoing

Am I processing one dataset that exists already and is fully visible to me, or does this need to handle future data on systems I won't directly observe?

For one-time analysis, the only kind of accuracy I care about is accuracy *on the data that's actually in front of me*. For ongoing systems that are designed to work without constant human review, I have to think about how the data could change and how my methods would respond.

This is related to the issues I mentioned earlier around how each method will break. For instance, if incoming data changes break my regular expressions, the most likely

outcome is empty fields and a broken pipeline—an obvious failure. But with an LLM I've added a model to my pipeline that can fail silently as data changes.

External LLM services also carry dependency risks: API endpoints change, pricing models shift, and models get deprecated.

These factors don't mean never using LLMs in ongoing systems—sometimes they're the only viable solution. But they do create trade-offs that should push me toward non-LLM solutions when possible, and toward robust monitoring approaches when not.

Why I Didn't Use an LLM for My Messy Text Data

Circling back to my original problem, I didn't throw my weirdly-formatted text data into an LLM because:

- I got a good solution without it
- I know exactly how my solution works and I have a good sense of when and how it will fail
- It uses minimal compute and has no ongoing costs
- It's a solution I'm comfortable maintaining long-term.

LLMs: Powerful Tools, Not Universal Solutions

A core issue in my decision-making is that adding an LLM step means adding another machine learning model to my project—one that's opaque and likely non-deterministic.

If I can easily check whether it's working, then this may not matter much.

But if I can't easily spot failures, or if it's going into an ongoing process where I would regularly check outputs, I need to decide if I'm ready for ongoing model monitoring and maintenance, the same as I would for any other machine learning model. This

concern increases if I'm using an external model that could change or disappear entirely.

So if I take this route, I should think carefully about ongoing testing and maintenance. For my use case, half of the testing is easy: I should write tests to check whether all characters in the LLM response appear in my original field and in the correct order because the LLM is just supposed to remove characters, not add any. This addresses the "did it make stuff up" question. But the "did it drop important stuff" question is harder, and for lots of other tasks I could have an LLM do, ongoing evaluation would be much more complicated.

LLMs are powerful tools, but that doesn't mean they're the right solution for every task. But the calculus can also shift as newer models have better performance, lower compute requirements, and lower prices. Six months from now, I might make different decisions for some problems as the capabilities/cost tradeoff continues to evolve. The important thing is having a framework to evaluate these decisions—that is, knowing what's important in each use case—and having an approach that's sufficiently test-driven and modular that you're able to swap in new methods as appropriate.

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

Thanks for reading The Present of Coding!
Subscribe for free to receive new posts and
support my work.



3 Likes

Discussion about this post

Comments

Restacks



Write a comment...

© 2025 Abigail Haddad • [Privacy](#) • [Terms](#) • [Collection notice](#)
[Substack](#) is the home for great culture