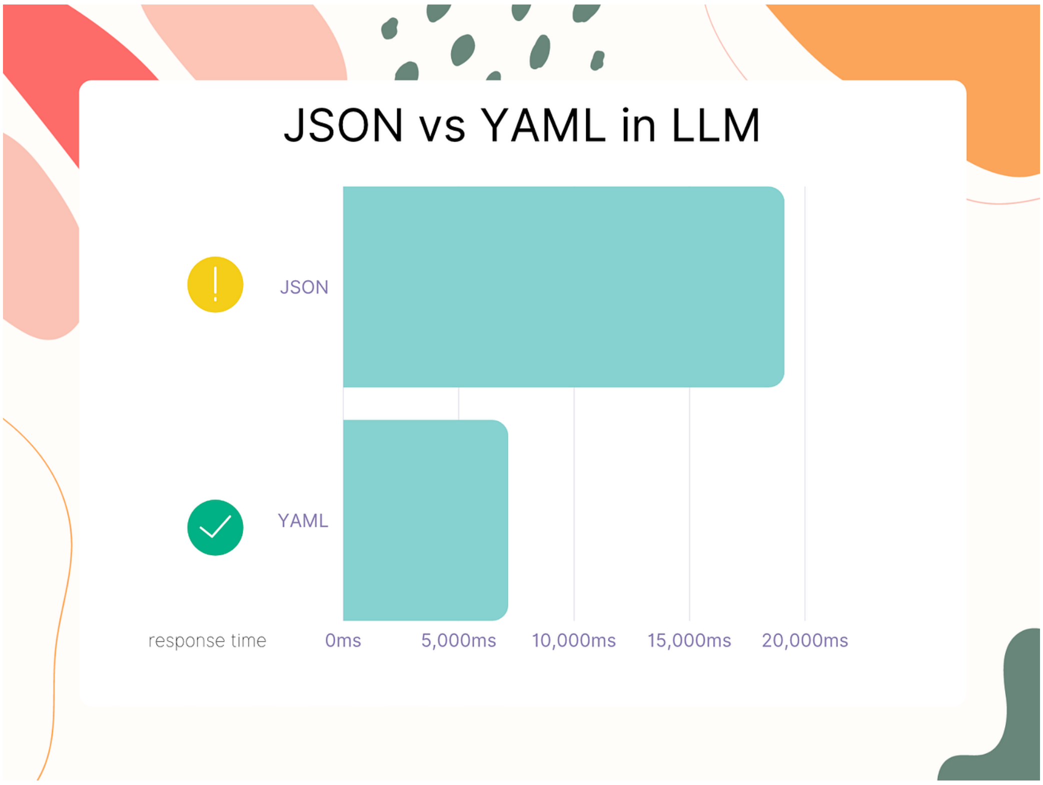
YAML vs. JSON: Which Is More Efficient for Language Models?

Elya Livshitz



In early 2020, I had the unique opportunity to gain access to OpenAI's GPT-3, a cutting-edge language model that seemed to possess almost magical capabilities. As I delved deeper into the technology, I discovered numerous ways to leverage its power in my personal and professional life, utilizing it as a life hack to expedite tasks and uncover novel concepts.

I quickly realized that working with GPT was not as intuitive as I had initially anticipated. Despite the introduction of ChatGPT, which aimed to bridge the gap and make this groundbreaking technology accessible to a wider audience, users still need a comprehensive understanding of how to maximize the potential of this innovative tool.

Over the past few months, I have conversed with numerous engineers and entrepreneurs who incorporate language models into their services and products. A recurring theme I observed was the attempt to solicit responses from language models in a JSON format. However, I discovered considerable consequences on output quality due to wording, prompt structure, and instructions. These factors can significantly impact a user's ability to control and fine-tune the output generated by GPT and similar language models.

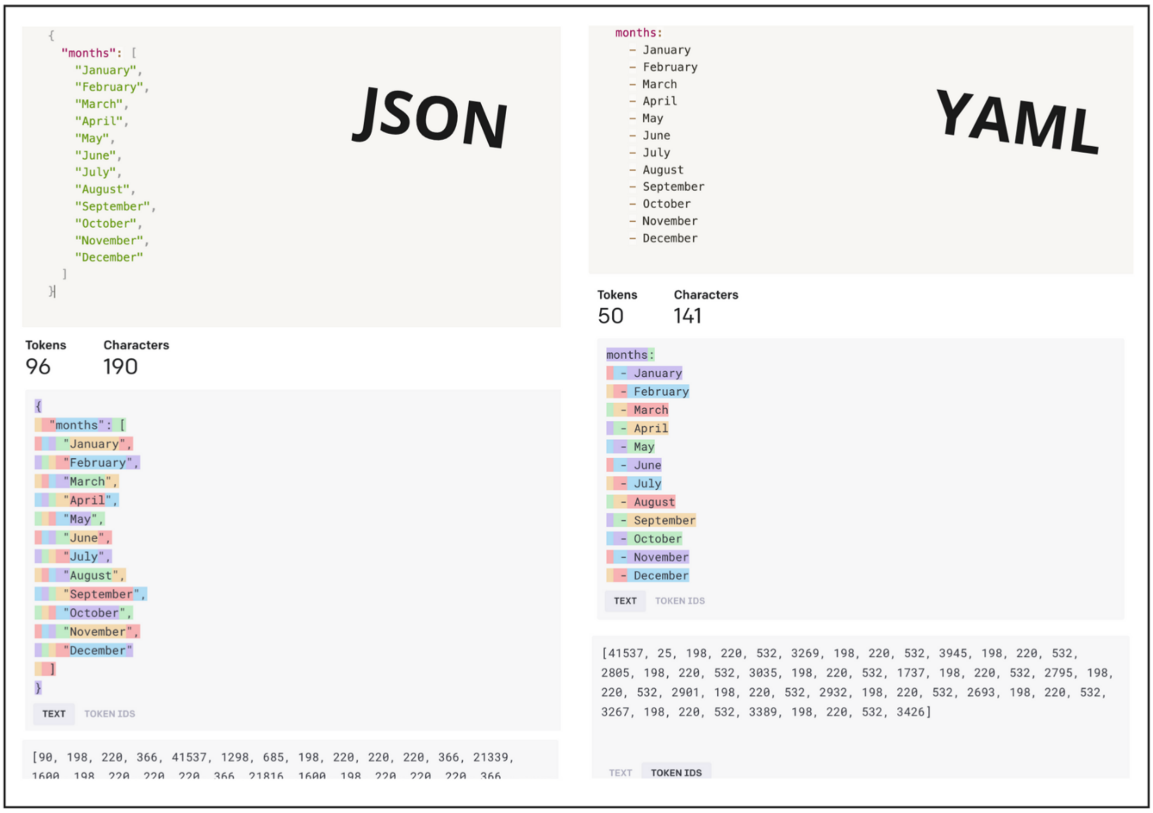
My intuition from my experiments was that JSON wasn't an efficient format to ask from a language model for various reasons:

1. Syntax issues: JSON is a sensitive format for quotes, commas, and other reserved symbols, which makes it difficult for language models to follow instructions consistently.
2. Prefix and suffix in the response: Language models tend to wrap the output with unnecessary texts.
3. Excessive costs: JSON format requires opening and closing tags, producing excessive text characters, and increasing the overall tokens and your costs.
4. Excessive execution time: Using language models as part of your application, especially if it's customer-facing, can be very sensitive to response time. Due to all of the above points, JSON can result in slow and flaky results, which can impact your user experience.

Empirical Experiments

After sharing my advice about JSON vs YAML a few times, I conducted an empirical study to prove my assumptions.

In order to test how GPT efficiency when it parses text of the same content, I asked GPT to generate a simple list of month names in JSON format and compared it to YAML format and compared using the [Tokenizer tool by OpenAI](more about tokens later). This simple example demonstrated about a 50% reduction in costs when using YAML:



```
{
  "months": [
    "January",
    "February",
    "March",
    "April",
    "May",
    "June",
    "July",
    "August",
    "September",
    "October",
    "November",
    "December"
  ]
}
```

Tokens 96 Characters 190

```
{
  "months": [
    "January",
    "February",
    "March",
    "April",
    "May",
    "June",
    "July",
    "August",
    "September",
    "October",
    "November",
    "December"
  ]
}
```

```
[90, 198, 220, 366, 41537, 1298, 685, 198, 220, 220, 220, 366, 21339,
1600, 198, 220, 220, 220, 366, 21816, 1600, 198, 220, 220, 220, 366
```

```
months:
  - January
  - February
  - March
  - April
  - May
  - June
  - July
  - August
  - September
  - October
  - November
  - December
```

Tokens 50 Characters 141

```
months:
  - January
  - February
  - March
  - April
  - May
  - June
  - July
  - August
  - September
  - October
  - November
  - December
```

```
[41537, 25, 198, 220, 532, 3269, 198, 220, 532, 3945, 198, 220, 532,
2805, 198, 220, 532, 3035, 198, 220, 532, 1737, 198, 220, 532, 2795, 198,
220, 532, 2901, 198, 220, 532, 2932, 198, 220, 532, 2693, 198, 220, 532,
3267, 198, 220, 532, 3389, 198, 220, 532, 3426]
```

The YAML approach here saved 48% in tokens and 25% in characters.

It is clear that YAML is significantly more cost/time-effective than JSON in those cases.

Deeper Look

Now, let’s look deeper into bigger completion performance time and the penalty for parsing the output as JSON or YAML. For parsing, I suggest using the js-yaml package for parsing the output into JS objects and PyYAML for Python.

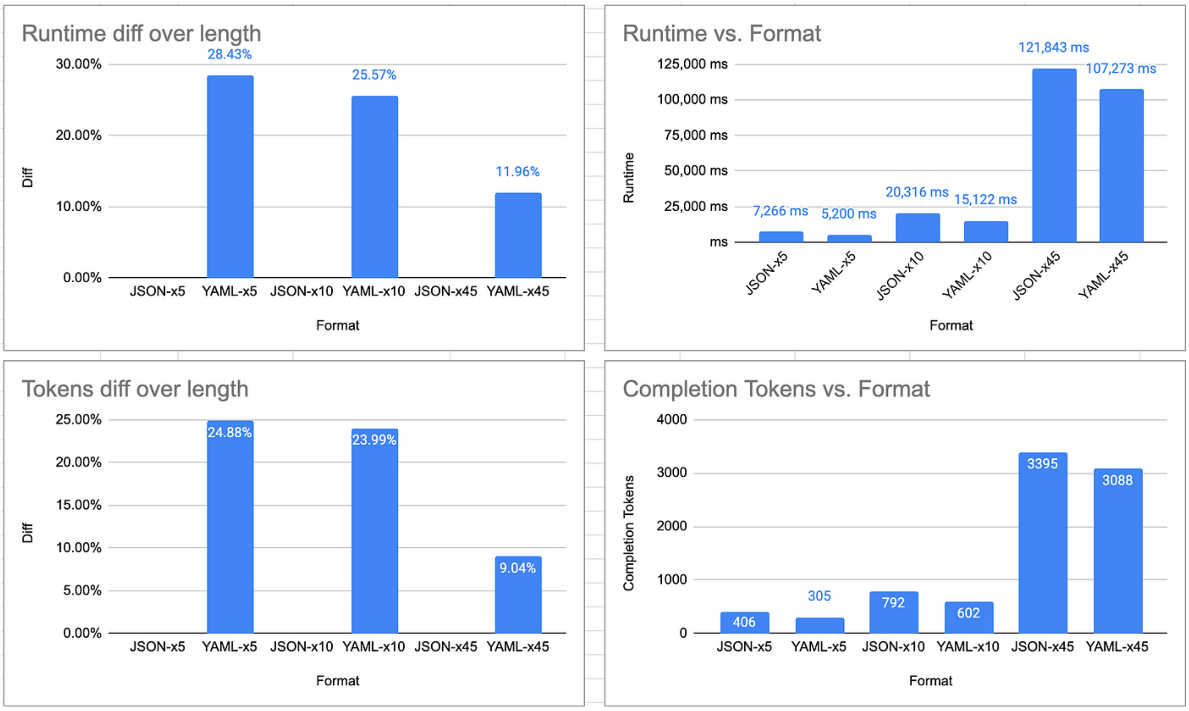
I’ve used this prompt to generate a somewhat deterministic test set with a predefined structure and measured results on various completion sizes (x5, x10, and x45, which consumed the whole tokens window):

Generate basic demographic info about 10 top countries (by population). Should include those fields: country, population, capital, official_language, currency, area_km, gdp_usd, under the root "countries". Output in {{format}} format, reduce other prose.(format: YAML|JSON)

Here’s the results I got:

Format	Length	Runtime	Completion Tokens	Characters	Parse Time	Runtime Diff	Tokens Diff
JSON-x5	x5	7,266 ms	406	1211	1 ms		
YAML-x5	x5	5,200 ms	305	905	4 ms	28.43%	24.88%
JSON-x10	x10	20,316 ms	792	2543	1 ms		
YAML-x10	x10	15,122 ms	602	1760	7 ms	25.57%	23.99%
JSON-x45	x45	121,843 ms	3395	10804	1 ms		
YAML-x45	x45	107,273 ms	3088	9729	11 ms	11.96%	9.04%

YAML tended to be faster and had a smaller footprint, but the gap degrades when getting closer to max token limit



Comparing YAML diffs over response length (left) and runtime/tokens (right)

The final JSON and YAML outputs can be found in the GH gist, accordingly.

If you were using this prompt on the scale of 1 million requests per month using JSON and GPT-4, switching to YAML would result in saving 190 tokens and would save you $11,400 (based on the pricing on this paper's day) per month with this simple trick.

Why Does This Happen?

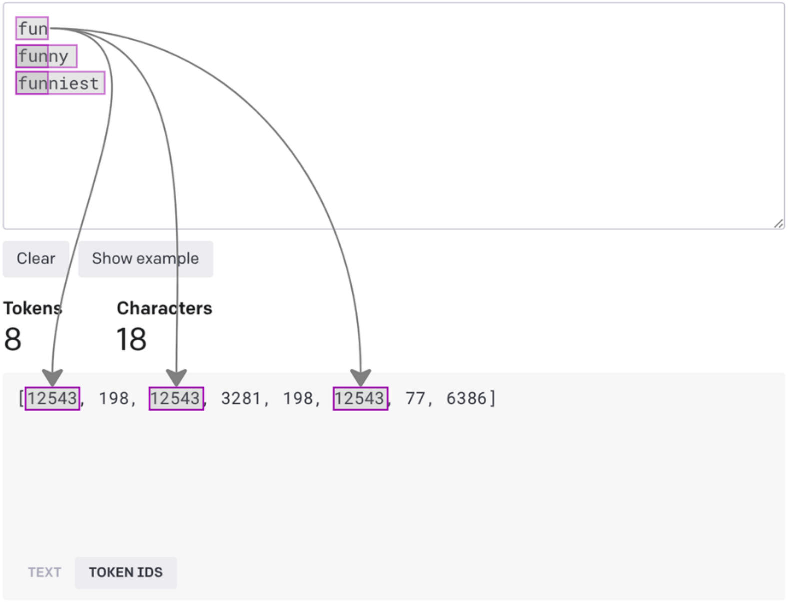
To understand why this happens, we need to understand how language models process text into tokens and tokens back into text.

Language models are machine learning models, and machines don't really understand "words" as a whole text, so words have to be encoded into a representation that machines can process. Each word could be represented by a unique ID, which is a machine-friendly representation. This is usually referred to as "Index-Based Encoding." Though it is somewhat inefficient as words with multiple variations like "fun," "funny," and "funniest" are semantically close, they will be represented in totally different and distinct IDs.

In 1994, Philip Gage introduced a new data compression technique that replaces common pairs of consecutive bytes with a byte that does not appear in that data. In other words, by splitting words into parts, we could yet represent words by unique token IDs and still store and retrieve them efficiently. This technique is called Byte Pair Encoding (BPE) and is used as subword tokenization. This technique has become the foundation for models such as BERT, GPT models, RoBERTa, and more.

To properly handle the token "est," for example, in the cases of "estimate" and "highest" ("est" appears at the beginning or the end but has different meanings), BPE attempts to combine pairs of two bytes or parts of words.
More on how GPT-3 tokens work is described well by Piotr Grudzien here.

Using the ***Tokenizer tool by OpenAI***, it can be demonstrated as follows:



```
fun
funny
funniest
```

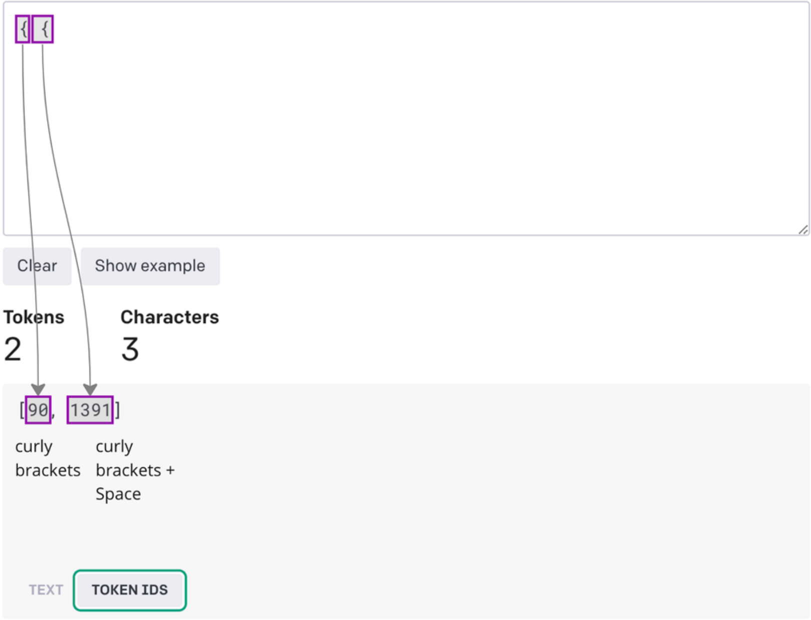
Tokens 8

Characters 18

```
[12543, 198, 12543, 3281, 198, 12543, 77, 6386]
```

BPE breaking words during subword tokenization

When this concept comes with single characters, such as curly brackets, we see something interesting:



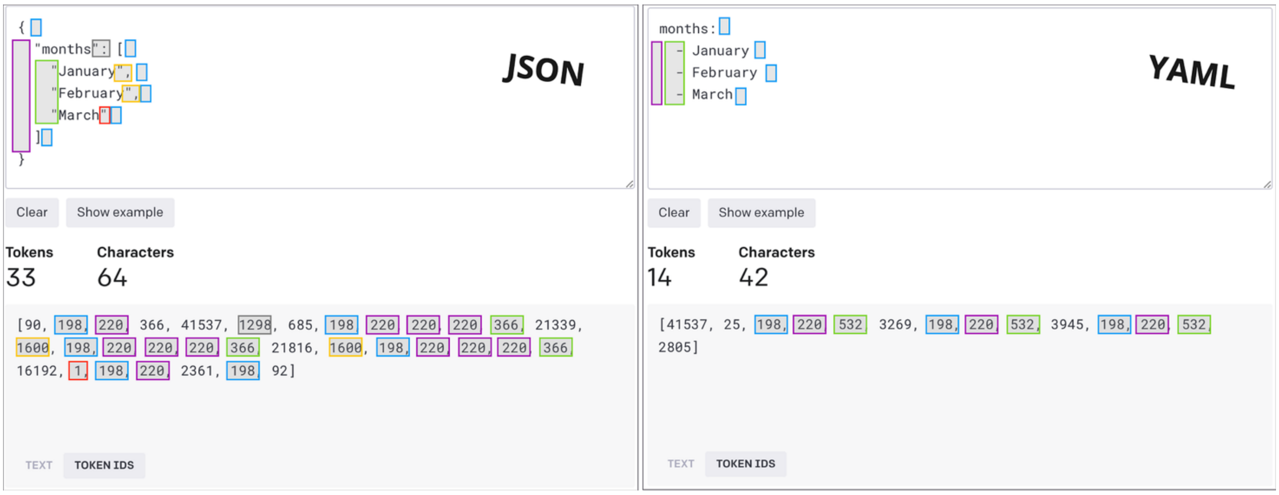
```
{ {
```

Tokens 2
Characters 3

```
[90, 1391]
```

Although we see the same character, BPE decides to categorize them differently

This fundamental behavior alone plays well in how YAML is structured (line breaks and spaces as special characters, without the need to open and close curly brackets, quotes, and commas) compared to JSON, which requires opening and closing tags. Opening and closing tags impact the underlying representation in tokens, eventually causing extra LLM spins and might impact the general ability to follow instructions. So, not only does this save characters, but it also generally helps language models represent words with token IDs that are more common in their BPE vocabulary.



```
{
  "months": [
    "January",
    "February",
    "March"
  ]
}
```

Tokens 33
Characters 64

```
[90, 198, 220, 366, 41537, 1298, 685, 198, 220, 220, 220, 366, 21339,
1600, 198, 220, 220, 220, 366, 21816, 1600, 198, 220, 220, 220, 366,
16192, 1, 198, 220, 2361, 198, 92]
```

```
months:
  - January
  - February
  - March
```

Tokens 14
Characters 42

```
[41537, 25, 198, 220, 532, 3269, 198, 220, 532, 3945, 198, 220, 532,
2805]
```

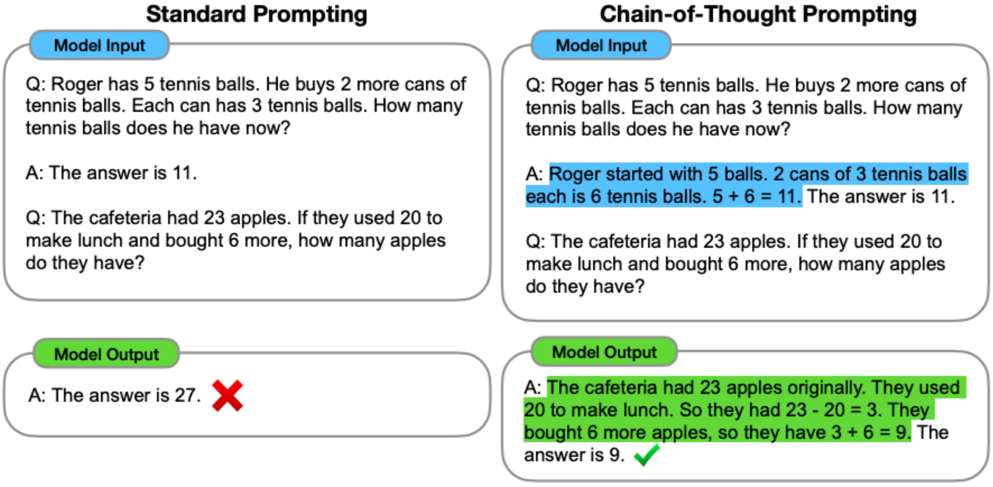
In comparing JSON and YAML, it is evident that the distribution of tokens in JSON is non-consistent, whereas YAML presents a more organized structure. This theoretically enhances the LLM's capacity to allocate more spins on content rather than focusing on structural aspects, consequently improving the overall output quality.

In conclusion, while JSON is generally faster to parse and consume than YAML, YAML is significantly more cost/time-efficient than JSON and can help language models produce precisely the same content faster and cheaper. Essentially, it is more efficient to request YAML, and convert the result to JSON on the code-side, instead of requesting JSON directly.

It is worth mentioning that the potential compromise might be the strictness of JSON for some formats (numbers could be printed as strings, surrounded with quotes). This can be solved by providing schema or post-parsing the fields into the right data type. Regardless, it could be good practice anyway to enforce data type conversions on code-side.

Appendix- Chain-of-Thought using YAML comments:

In addition to its advantages in speed and cost, YAML offers another significant benefit over JSON — the capacity to include comments.
Take this classic test case from "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" (Wei et al. ,2022):



Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

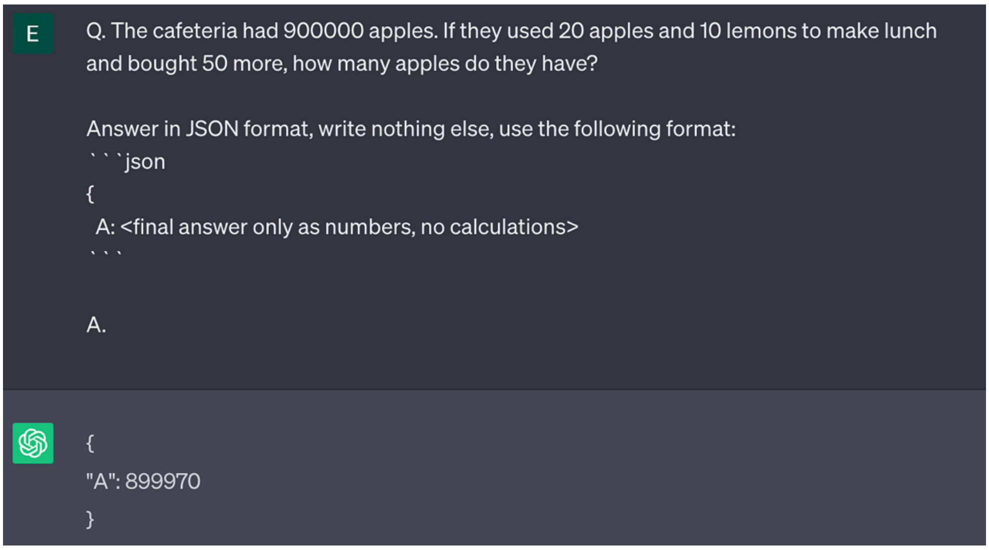
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Imagine you want this output in machine-readable format.
With JSON and no CoT, you'll get bad results:

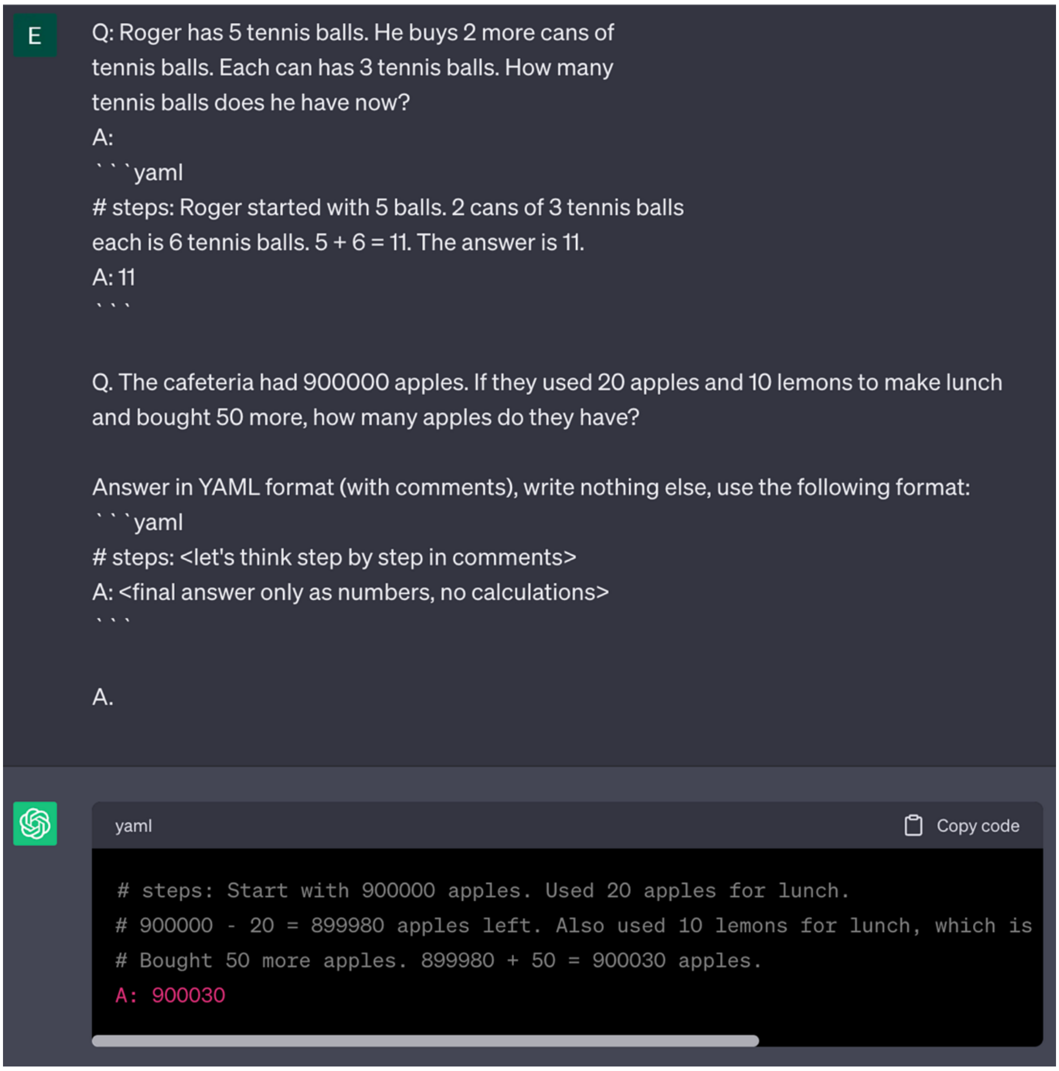


~~~
Q. The cafeteria had 900000 apples. If they used 20 apples and 10 lemons to make lunch
and bought 50 more, how many apples do they have?

Answer in JSON format, write nothing else, use the following format:
```json
{
  A: <final answer only as numbers, no calculations>
```

A.
~~~

```
{
"A": 899970
}
```

No CoT, JSON return, GPT-3.5. Wrong answer, should return 900030

However, by utilizing YAML, you can define a format that accommodates the CoT within comments while presenting the final answer in the assigned key, ultimately producing a parseable output:



Q: Roger has 5 tennis balls. He buys 2 more cans of
tennis balls. Each can has 3 tennis balls. How many
tennis balls does he have now?
A:
` ` `yaml
steps: Roger started with 5 balls. 2 cans of 3 tennis balls
each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
A: 11
` ` `

Q. The cafeteria had 900000 apples. If they used 20 apples and 10 lemons to make lunch and bought 50 more, how many apples do they have?

Answer in YAML format (with comments), write nothing else, use the following format:
` ` `yaml
steps: <let's think step by step in comments>
A: <final answer only as numbers, no calculations>
` ` `

A.

```yaml
# steps: Start with 900000 apples. Used 20 apples for lunch.
# 900000 - 20 = 899980 apples left. Also used 10 lemons for lunch, which is
# Bought 50 more apples. 899980 + 50 = 900030 apples.
A: 900030
```

CoT with YAML comments, GPT-3.5, CORRECT answer