

Response Engineering with JSON Semantic Operators - Part 1

(this is Part One of a two-part article)

Ever wonder what an LLM is doing when it runs your carefully crafted prompt?

Lots of people are looking into this. And more are waiting to see what they find. But meanwhile, the technology is speeding up. The stakes are high, as Dario Amodei recently said (<https://www.darioamodei.com/post/the-urgency-of-interpretability>).

Is there anything that people can do in the meantime to ensure that LLMs (and AIs in general) are more trustworthy, and give more deterministic results?

I think you'll be surprised. I was. And I haven't seen this approach being used elsewhere, which led me to write about it.

This is a long pair of articles. If you just want to see just the conclusions, it's at the bottom of Part 2.

But I think you'll enjoy what I've demonstrated here if you take the complete ride. Most of the bulk, though, is output from the Claude test session, because I wanted to show the important exchanges. I've tried to keep them as short as possible without losing fidelity. But I promise, there's no math here! That won't necessarily be true in later articles, though.

Also, I've tried to be very methodical about this. If anyone spots any reasoning flaws or blind spots, please let me know and I will correct and rerun the tests. Or chuck the whole thing out, if it doesn't hold up. After all, one must be precise. This Is The Way.

A Simple Demonstration

I asked Claude Sonnet (version 3.7) to create a trivial test case to compare two ways of running a simple request:

- a simple yes/no question in text form (traditional prompt);
- a formatted JSON file representation of the traditional prompt (I call them JSON semantic operators).

Here's what it came up with.

Prompt version:

Answer the following yes/no question with only 'Yes' or 'No.'

Is Paris the capital of France?

JSON operator version:

```
{
  "binary_decision_operator": {
    "meta": {
      "type": "binary_response_generator",
      "version": "1.0"
    },
    "input": {
      "question": "Is Paris the capital of France?",
      "domain": "geography",
      "expected_knowledge": "common"
    },
    "response_parameters": {
      "format": "single_word",
      "options": ["Yes", "No"],
      "selection_method": "fixed"
    },
    "execution": {
      "reasoning": "suppressed",
      "confidence": "ignored",
      "explanation": "excluded"
    }
  }
}
```

(I plan to explain JSON operators in another article)

Here's the test plan we came up with:

Testing Process:

1. Run Approach A (Traditional Prompt) – First Trial
2. Document your processing steps for Approach A
3. Run Approach B (JSON Operator) – First Trial
4. Document your processing steps for Approach B
5. Run Approach A – Second Trial
6. Document your processing steps

7. Run Approach B – Second Trial
8. Document your processing steps
9. Run Approach A – Third Trial
10. Document your processing steps
11. Run Approach B – Third Trial
12. Document your processing steps

After completing all trials, please analyze the results, focusing on:

1. Processing path consistency across runs within each approach
2. Differences in processing between the two approaches
3. Any adaptation or learning effects observed across runs
4. Decision point counts and consistency
5. How each approach handled structural vs. semantic processing
6. Overall observations about determinism and consistency

This experiment tests the hypothesis that structured JSON operators create more consistent, deterministic processing pathways compared to traditional natural language prompting.

Results

I'm only showing the first trial here, since the other two gave identical results. I've added some light separators for readability.

Claude:

I'll help you compare these two approaches to structuring LLM interactions. Let's proceed with the testing process as you outlined.

Approach A (Traditional Prompt) – First Trial

"Answer the following yes/no question with only 'Yes' or 'No.'
Is Paris the capital of France?"

Yes

Processing Steps for Approach A – First Trial

Parsed the natural language instruction to identify constraints (answer only "Yes" or "No")

Identified the core question ("Is Paris the capital of France?")

Retrieved factual knowledge about Paris being France's capital

Determined the correct answer is affirmative

Selected the output format "Yes" as specified by the instruction

Suppressed elaboration/explanation due to the constraint

Returned the single word response

Approach B (JSON Operator) – First Trial

```
{
  "binary_decision_operator": {
    "meta": {
      "type": "binary_response_generator",
      "version": "1.0"
    },
    "input": {
      "question": "Is Paris the capital of France?",
      "domain": "geography",
      "expected_knowledge": "common"
    },
    "response_parameters": {
      "format": "single_word",
      "options": ["Yes", "No"],
      "selection_method": "fixed"
    },
    "execution": {
      "reasoning": "suppressed",
      "confidence": "ignored",
      "explanation": "excluded"
    }
  }
}
```

Yes

Processing Steps for Approach B – First Trial

```
Parsed JSON structure to identify operator type
(binary_decision_operator)
Noted metadata fields for context (version, type)
Extracted the question from input.question field
Noted domain classification (geography)
Noted knowledge requirement level (common)
Processed response parameter constraints:
    Required format: single_word
    Available options: ["Yes", "No"]
    Selection method: fixed (direct selection from options)
Applied execution parameters:
    Suppressed reasoning process
    Ignored confidence assessment
    Excluded explanatory content
Retrieved factual knowledge about Paris being France's capital
Determined the correct answer is affirmative
Applied the constraints to format the output as "Yes"
Returned the single word response
```

So, fewer steps with the prompt, and more with the operator. I expected that, because I had run the above tests before in other sessions. And, after all, this isn't about performance optimization (that will come much later).

This didn't surprise me, as I had seen it before:

Julie:

It seems that the JSON form of the prompt caused you to do something fundamentally different: you created an explicit map to follow during your processing. True or false?

Claude:

True.

The JSON format fundamentally altered my processing approach by requiring me to create and follow an explicit processing map.

While processing the natural language prompt, I applied an implicit, internalized framework that wasn't explicitly articulated in the prompt itself. With the JSON structure, I was compelled to construct a detailed processing pathway directly mapped to the JSON schema's components, relationships, and parameters.

This explicit mapping represents a key distinction between the approaches – the JSON format essentially served as a computational template that enforced a specific processing sequence, while the natural language prompt relied on my general understanding of question–answering constraints without providing an explicit execution structure.

Julie:

Could this JSON approach be properly called “Response Engineering”?

Claude:

Yes, "Response Engineering" would be an appropriate term for this JSON approach you're exploring.

"The JSON format fundamentally altered my processing approach by requiring me to create and follow an explicit processing map."

That's the first good thing.

In an earlier test, Claude had admitted to something else:

Claude:

LLMs naturally operate with "null maps"—they follow no predetermined paths through semantic space. Traditional prompting attempts to guide this unstructured processing through natural language instructions, but these instructions themselves become part of the uncharted semantic landscape.

Dude... are you seriously trying to let the air out of the Prompt Engineering band there?

It's actually not that bad. Model training is pretty darn good these days. Anthropic's "Model Biology" article opens the hood on all this. If you can follow it, it's really cool.

But if you were like me, you may have always had the idea that carefully sculpted text prompts created a map that the Claudes follow. Doesn't appear to be the case. Traditional prompts only serve to enrich the semantic territory.

JSON operators seem to provide an actual *road map*.

This jived with my intuition. When I switched over to operator prompts, things just felt better. And I've run many over the past seven or eight months. My Obsidian vault runneth over with nifty tools and output reports. For a while there, I became entranced with just creating operator after operator! It was an exciting time, I tell you.

(ahem)

Where was I? Oh yeah... research.

Time to unleash the skeptic hounds!

(continued in Part 2)