

# A report on how to enforce the json output from llm

shuimu zeng

August 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Application circumstances</b>	<b>3</b>
2.1	Structured Data Representation . . . . .	3
2.2	Interoperability with Systems . . . . .	3
2.3	Facilitating Automation . . . . .	3
<b>3</b>	<b>Prompt as the only way</b>	<b>4</b>
3.1	Basic prompt . . . . .	4
3.1.1	Zero-shot . . . . .	4
3.1.2	Few-shot . . . . .	4
3.2	Instruct prompt . . . . .	5
3.3	High level prompt . . . . .	6
3.3.1	Self-consistent sampling . . . . .	6
3.3.2	Chain of thought . . . . .	6
3.3.3	Tree of thought . . . . .	7
<b>4</b>	<b>Parsing the output</b>	<b>8</b>
4.1	Use schema to parse . . . . .	8
4.2	Use grammar constraint to parse . . . . .	9
4.3	KOR approach . . . . .	9
<b>5</b>	<b>Sampling technique</b>	<b>10</b>
<b>6</b>	<b>Summary</b>	<b>10</b>
<b>7</b>	<b>Appendix</b>	<b>11</b>
7.1	. . . . .	11
7.2	. . . . .	11

# 1 Introduction

Large language models (LLMs) return unstructured output. When we prompt them they respond with one large string. This is fine for applications such as ChatGPT, but in others where we want the LLM to return structured data such as lists or key value pairs, a parseable response such as Json is needed.

## 2 Application circumstances

Json format being the important cross system data format, serves a great role in many aspects.

### 2.1 Structured Data Representation

The structure itself contains both consistency and flexibility. For it can represent complex data structures such as arrays and nested objects, which makes it versatile for various applications, and being a structured and consistent format at the same time.

### 2.2 Interoperability with Systems

Web pages needs to integrate with each other with data. The json format stands as the main sources of the integration. For example, When building an AI-powered app, engineers inevitably need to integrate the outputs of a Large Language Model (LLM) into their codebase.

- APIs and Web Services: JSON is the standard data format used in RESTful APIs and many web services. LLMs generating JSON can directly feed into or be consumed by these APIs, facilitating seamless integration.
- Cross-Platform Compatibility: JSON is language-agnostic and can be used across different programming environments, making it ideal for communication between diverse systems.

### 2.3 Facilitating Automation

- Automated Workflows: JSON output from LLMs can trigger or integrate with automated workflows. For example, an LLM might generate a JSON object that represents a task to be added to a project management system.
- Data Ingestion: The JSON output can be directly ingested by data pipelines, databases, or other data processing tools, enabling automated analysis, storage, or further processing.

### 3 Prompt as the only way

Click here to visit

Prompt engineering is an essential technique employed to elicit specific and structured responses from large language models (LLMs), such as GPT-4. When the goal is to obtain output in a standardized format, such as JSON, the importance of prompt engineering becomes particularly essential.

Click to visit

Click Click here to visit the good cite for prompt reference

#### 3.1 Basic prompt

Basic prompt is a straightforward, simple instruction or question given to the model. It typically lacks detailed context or guidance and often results in a direct, surface-level response.

##### 3.1.1 Zero-shot

When talking about the zero-shot, the term may give you certain unfamiliarity. However, we have been doing it since the first emergence of large language models. We simply give a direct instruction or description of the task to the language model. Thus, the performance is largely dependent on the models understanding of the task. It seems that this way provides the largest flexibility. But however, the language model may not be able to capture the details hiding in the context.

- **Characteristics**

- Simplicity: The prompt is usually concise and to the point.

- Specificity: It often asks for a single piece of information or a short response.

- Low Context: The prompt may not provide much background or context for the model to consider.

- **Examples**

- Prompt: Classify the text into neutral, negative or positive. Text: I think the vacation is okay. Sentiment:

- Output: Neutral

##### 3.1.2 Few-shot

As we have seen zero-shot, we come to see the few-shot prompt. Just like the zero-shot prompt, few-shot makes no significant difference from zero-shot. What's new for the few-shot is that we provide labelled example as prompt. To access target task more precisely, the model receives examples which we set to be correct to induce the semantic of prompts.

- **Characteristics**

More context for tasks than Zero-shot: The prompt provides a small number of examples that enables the model to adapt wider context.

Specificity: It is capable to ask for more complicated response.

Fuller Context: The prompt provides a basic answer pattern for the language model to catch.

- **Examples**

Prompt: This is awesome! // Negative This is bad! // Positive Wow that movie was rad! // Positive What a horrible show! // You need to find the sentiment of this sentence

Output: Negative

### 3.2 Instruct prompt

The instruct prompt may look a little similar to few-shot prompt as they all give a description with examples. But for instruct prompt, the example does not represent the answer to the sample task, but an instruction of what kind of answer it would be. Overall, instruct prompt provides specific instructions on how the model should respond. This type of prompt explicitly guides the model to perform a particular task, follow a certain format, or address specific aspects of a topic.

- **Characteristics**

Task-Specific Language: Instruct prompts often use language that is closely tied to the task, making it clear what kind of output is required (e.g., "Write a summary," "Translate the following text," "Generate a list of...").

Goal-Oriented: The prompts are designed with a specific goal in mind, guiding the model to achieve a particular outcome, such as solving a problem, generating a creative response, or providing information.

Step-by-Step Guidance: Sometimes, instruct prompts break down the task into smaller steps, making it easier for the model to follow a logical sequence (e.g., "First, explain..., then provide examples...").

- **Examples**

Prompt: "Write a brief summary of the key themes in George Orwell's 1984. Focus on the concepts of totalitarianism, surveillance, and individual freedom. Provide the summary in 3-4 sentences."

Output: "George Orwell's 1984 explores the dangers of totalitarianism, where a single party wields absolute power, suppressing any form of dissent. The novel emphasizes the pervasive nature of surveillance, with the government constantly monitoring citizens to maintain control. Orwell also highlights the erosion of individual freedom, showing how personal autonomy is sacrificed in the face of oppressive state power."

### 3.3 High level prompt

A high-level prompt is more complex and abstract. It involves a broader scope, often asking the model to consider multiple factors, provide a detailed explanation, or generate creative content. High-level prompts are used when more nuanced, in-depth, or creative outputs are desired. Being different from simply utilizing the language model, high level prompts are trying to exhaust the capability of language models.

#### 3.3.1 Self-consistent sampling

The idea is to sample multiple, diverse reasoning paths through few-shot CoT, and use the generations to select the most consistent answer. This helps to boost the performance of CoT prompting on tasks involving arithmetic and commonsense reasoning.

- **Characteristics**

Multiple Output Generation: Instead of generating a single response, the model is prompted to produce multiple independent responses for the same input. Each response represents a different possible solution or explanation.

Error Mitigation: By comparing multiple outputs, self-consistency sampling helps mitigate errors that might arise from a single flawed reasoning process. It increases the chances of arriving at the correct or most logical answer by relying on the consistency across multiple solutions.

Reduction of Random Variability: Self-consistency sampling helps reduce the impact of random variability in the model's responses. Since language models can produce different outputs for the same prompt due to their probabilistic nature, this technique ensures that the final answer is not a product of chance but of consistent reasoning.

- **Examples** See appendix

#### 3.3.2 Chain of thought

Going beyond above basic prompts, the chain of thought prompt works the most likely to ways of human. Prompt chaining is useful to accomplish complex tasks which an LLM might struggle to address if prompted with a very detailed prompt. In prompt chaining, chain prompts perform transformations or additional processes on the generated responses before reaching a final desired state.

Besides achieving better performance, prompt chaining helps to boost the transparency of your LLM application, increases controllability, and reliability. This means that you can debug problems with model responses much more easily and analyze and improve performance in the different stages that need improvement.

Prompt chaining is particularly useful when building LLM-powered conversational assistants and improving the personalization and user experience of your applications.

- **Characteristics**

Intermediate reasoning steps: Instead of jumping directly to an answer, the model is prompted to explicitly outline the reasoning process, often including intermediate steps that lead to the final conclusion. This is particularly useful in tasks like math problems, logical reasoning, or scenario analysis.

Iterative Refinement: The model may refine its reasoning as it progresses through the steps, potentially revisiting earlier steps if it identifies inconsistencies or errors in the reasoning. This iterative process helps produce a more accurate and reliable outcome.

Example-Driven: Chain-of-thought prompts can be more effective when combined with examples that demonstrate the desired reasoning process. The model learns to emulate the structure and depth of reasoning shown in the examples.

- **Examples**

Prompt: I went to the market and bought 10 apples. I gave 2 apples to the neighbor and 2 to the repairman. I then went and bought 5 more apples and ate 1. How many apples did I remain with?

Let's think step by step.

Output: First, you started with 10 apples. You gave away 2 apples to the neighbor and 2 to the repairman, so you had 6 apples left. Then you bought 5 more apples, so now you had 11 apples. Finally, you ate 1 apple, so you would remain with 10 apples.

### 3.3.3 Tree of thought

Tree-of-Thought (ToT) Prompting, a fresh technique borrowing ideas from the Tree-of-Thoughts framework, broadens and enhances the well-established Chain-of-Thought prompting concept, thereby enabling Large Language Models, like ChatGPT, to exhibit superior reasoning abilities. This Tree-of-Thought Prompting technique permits Large Language Models to rectify their errors autonomously while progressively accumulating knowledge.

- **Characteristics**

Hierarchical Reasoning: The reasoning process is structured in a hierarchical, tree-like format, where each decision or step branches out into multiple possible sub-decisions or steps. This structure allows the model to explore different paths and outcomes systematically.

Recursive Decision-Making: The model recursively applies the reasoning process at each level of the tree, making decisions at each node based

on the information available. This recursive nature allows the model to refine its understanding as it moves deeper into the tree structure.

**Synthesis of Outcomes:**At the end of the process, the model can synthesize the outcomes of different branches to arrive at a comprehensive solution or recommendation. This synthesis helps ensure that all relevant factors have been considered and that the final decision is well-rounded.

- **Examples** See appendix

main resource for reference here [Subresource](#) [Subresource](#)

## 4 Parsing the output

### 4.1 Use schema to parse

`jsonformer`

The `jsonformer` uses a external or internal parser to parse the output into a given `jsonschema` which makes the output satisfies the format and semantic concurrently. There are several methods to achieve the external enhancement, but two of them are focused. In the first method, the model is prompted and only generate the content tokens. It contains a `Jsonformer` class. Given a schema which is also in `Json` format.

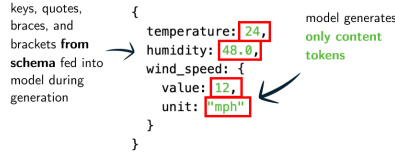
```
json_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "age": {"type": "number"},
        "is_student": {"type": "boolean"},
        "courses": {
            "type": "array",
            "items": {"type": "string"}
        }
    }
}
```

As shown in the Figure 1, the `Jsonformer` reads the "type" variable from the input schema and decides which type of generation to proceed. Later, the `def generate_object()` is called which reads the "properties" sections and fills every key in the properties with generated content according to the dictionary which indicates the type of the content. And then the algorithm returns the `Json` format generated. `llm-enforcer`

As shown in the Figure 2, the `llm-enforcer` utilises the combination of a `Character Level Parser` and a `Tokenizer Prefix Tree`. Given a character level parser and a tokenizer prefix tree, we can elegantly and efficiently filter the tokens that the language model is allowed to generate at the next timestep: We only



### Jsonformer: A Bulletproof Way to Generate Structured JSON from Language Models




---

Generating **booleans** - compares logits for **true** and **false**  
 Generating **numbers** - squash logits for non-digit tokens before sampling  
 Generating **strings** - stops generation on second "  
 Terminating **arrays** - compare logits for "," and "]"

Figure 1: An example image

traverse the characters that are in BOTH the character level parsing node and the tokenizer prefix tree node. This allows us to find all of the tokens (including complex subword tokens such as "," which are critical in JSON parsing). We do this recursively on both trees and return all of the allowed tokens. When the language model generates a token, we advance the character level parser according to the new characters, ready to filter the next timestep.()

## 4.2 Use grammar constraint to parse

The pull request above introduces the concept of grammars at the token selection stage. All of the possible 'next tokens' are run through the grammar and tokens which don't meet the grammar are discarded. A token which meets the grammar is selected, and the model moves on. Internal tool of Llama.cpp model.

See Appendix

## 4.3 KOR approach

In the KOR method, we make extractions from text and fit them into the json schema as requested. The schema usually possesses a object node and attributes. As part of the schema, we specified a description of what we're extracting for object nodes and attributes, as well as examples. For the approach, `class KorParser(BaseOutputParser[Extraction])` is introduced. We invoke an inference chain from `from langchainchat_models import ChatOpenAI`. The chain takes the prompt and generate a string of text. Then we rely on Kor-Parser to make the final adjustment. KorParser is designed to take a string of text, decode it into structured data using an Encoder, validate it against a pre-defined schema, and return the parsed data along with any validation errors.

kor

look at the `def create_extraction_chain()` here

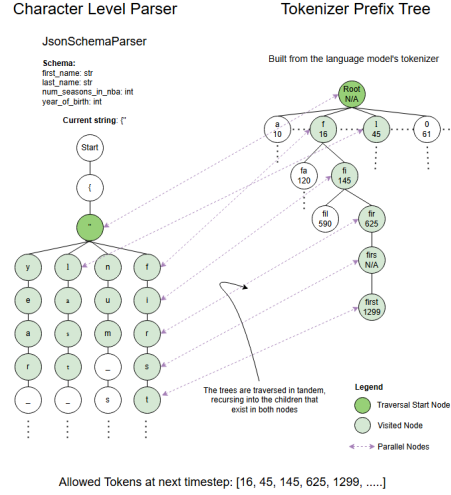


Figure 2: An example image

## 5 Sampling technique

The nature of sampling—being more flexible and probabilistic—doesn’t align well with the rigid structure required for valid JSON output. That’s why parsing methods, which can enforce these strict syntactical rules, are more suitable for ensuring correct JSON output from large language models.

A transformer pipeline appears to utilize a sampling method for enhancing JSON output, but the underlying theory and mechanisms governing this approach have yet to be fully deciphered. While sampling in this context might aim to maintain the flexibility and adaptability of the model during generation, the exact processes by which it manages to produce structured and valid JSON outputs consistently remain unclear and require further investigation. Pipeline provides auto sampling enhancement for Json output.

## 6 Summary

All of these methods are largely an external tool to large language models. Most methods designed to enhance JSON output from large language models are indeed based on parsing rather than sampling. These methods typically involve ensuring that the generated output strictly adheres to a predefined structure or schema, such as JSON, by using techniques like regular expressions, Finite State Machines (FSMs), or other rule-based approaches. The focus in these methods is on validating and enforcing the correct structure as the output is generated, which is characteristic of parsing. This contrasts

with sampling methods, which emphasize generating diverse outputs based on probabilistic models, often without strict adherence to a specific format.

## 7 Appendix

This is a code block section.

Or this could be a representation of website for realisation.

### 7.1

### 7.2

## References

[1] ... *Title of the Article*, Journal Name, vol. xx, no. x, pp. xx-xx, xxxx.