User and assistant conversation

Here is a regular multi-turn user assistant conversation and how its formatted.

Input Prompt Format

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a helpful assistant<|eot_id|><|start_header_id|>user<|end_header_id|>

Who are you?<|eot_id|><|start_header_id|>assistant<|end_header_id|>


```

Model Response Format

```
I'm an AI assistant, which means I'm a computer program designed to simulate conversation and answer questions to the best of my ability. I'm here to help you with any questions or tasks you may have, and I'll do my best to provide you with accurate and helpful information.

I don't have a personal name, but I'm often referred to as a "virtual assistant" or a "chatbot." I'm a machine learning model, which means I've been trained on a large dataset of text and can generate responses based on patterns and context.

I can help with a wide range of topics, from general knowledge and trivia to more specialized subjects like science, history, and technology. I can also assist with tasks like language translation,
```

text summarization, and even generating creative content like stories or poetry.

So, what can I help you with today?<|eot_id|>

```

##### Notes

This format is unchanged from Llama3.1

## Zero shot function calling

For Llama3.2 1B and 3B instruct models, we are introducing a new format for zero shot function calling.

This new format is designed to be more flexible and powerful than the previous format.

All available functions can be provided in the system message. A key difference is in the format of how the assistant responds with function calls.

It is pythonic in the form of `[func1(params_name=params_value, params_name2=params_value2...), func2(params)]` instead of the `json` or `&lt;function&gt;` tag that were defined in Llama3.1.

Here is an example for the same,

##### Input Prompt Format

```

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are an expert in composing functions. You are given a question and a set of possible functions.
Based on the question, you will need to make one or more function/tool calls to achieve the purpose.
If none of the function can be used, point it out. If the given question lacks the parameters required by the function,
also point it out. You should only return the function call in tools call sections.

If you decide to invoke any of the function(s), you MUST put it in the format of [func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]
You SHOULD NOT include any other text in the response.

Here is a list of functions in JSON format that you can invoke.

```
[
    {
        "name": "get_weather",
        "description": "Get weather info for places",
        "parameters": {
            "type": "dict",
            "required": [
                "city"
            ],
            "properties": {
                "city": {
                    "type": "string",
                    "description": "The name of the city to get the weather for"
```

```
            },
            "metric": {
                "type": "string",
                "description": "The metric for weather. Options are: celsius, fahrenheit",
                "default": "celsius"
            }
        }
    }
]<|eot_id|><|start_header_id|>user<|end_header_id|>

What is the weather in SF and Seattle?<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

````
##### Model Response Format
```
[get_weather(city='San Francisco', metric='celsius'), get_weather(city='Seattle', metric='celsius')]<|eot_id|>
```
````

```
##### Notes

- The output supports multiple tool calls natively
- JSON format for defining the functions in the system prompt is similar to Llama3.1
```

Zero shot function calling with user message

While the default is to provide all function calls in a system message, in Llama3.2 text models you can also provide information for all the available tools in a user message.

Input Prompt Format

```
<|begin_of_text|><|start_header_id|>user<|end_header_id|>

Questions: Can you retrieve the details for the user with the ID 7890, who has black as their special request?
Here is a list of functions in JSON format that you can invoke:
[
    {
        "name": "get_user_info",
            "description": "Retrieve details for a specific user by their unique identifier. Note that the provided function is in Python 3 syntax.",
        "parameters": {
            "type": "dict",
            "required": [
                "user_id"
            ],
            "properties": {
```

```
            "user_id": {
            "type": "integer",
             "description": "The unique identifier of the user. It is used to fetch the specific user details from the database."
            },
            "special": {
                "type": "string",
                    "description": "Any special information or parameters that need to be considered while fetching user details.",
                "default": "none"
                }
            }
        }
    }
]
```

Should you decide to return the function call(s),Put it in the format of [func1(params_name=params_value, params_name2=params_value2...), func2(params)]

NO other text MUST be included.<|eot_id|><|start_header_id|>assistant<|end_header_id|>

```
```

Model Response Format

```
[get_user_info(user_id=7890, special='black')]<|eot_id|>
```

```

##### Notes

- The tool call format for the model is the same whether your function calls are provided in the system or user message.
- While builtin tool calls end with a <|eom_id|>, notice the <|eot_id|> for zero shot tool calls.

## Code Interpreter

Code Interpreter continues to work in 3.2 text models similar to Llama 3.1 model family.

Here is an example,

##### Input Prompt Format

```

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Environment: ipython

Cutting Knowledge Date: December 2023

Today Date: 24 September 2024

<|eot_id|><|start_header_id|>user<|end_header_id|>

Write code to check if number is prime. Use it to verify if number 7 is prime<|eot_id|><|start_header_id|>assistant<|end_header_id|>

```

##### Model Response Format

```
<|python_tag|>def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    max_divisor = int(n**0.5) + 1
    for d in range(3, max_divisor, 2):
        if n % d == 0:
            return False
    return True

print(is_prime(7)) # Output: True<|eom_id|>
```

##### Notes
```

- Note `Environment: ipython` in the system prompt.

- Note that the response starts with `<|python_tag|>` and ends with `<|eom_id|>`

Zero shot function calling E2E format

Here is an example of the e2e cycle of tool calls with the model in a muti-step way.

Input Prompt Format

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are an expert in composing functions. You are given a question and a set of possible functions.
Based on the question, you will need to make one or more function/tool calls to achieve the purpose.
If none of the function can be used, point it out. If the given question lacks the parameters required by the function,
also point it out. You should only return the function call in tools call sections.

If you decide to invoke any of the function(s), you MUST put it in the format of [func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]
You SHOULD NOT include any other text in the response.

Here is a list of functions in JSON format that you can invoke.
```

```
[
    {
        "name": "get_weather",
        "description": "Get weather info for places",
        "parameters": {
            "type": "dict",
            "required": [
                "city"
            ],
            "properties": {
                "city": {
                    "type": "string",
                    "description": "The name of the city to get the weather for"
                },
                "metric": {
                    "type": "string",
                    "description": "The metric for weather. Options are: celsius, fahrenheit",
                    "default": "celsius"
                }
            }
        }
    }
]<|eot_id|><|start_header_id|>user<|end_header_id|>

What is the weather in SF?<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

<|python_tag|>[get_weather(city="San Francisco", metric="celsius")]<|eot_id|><|start_header_id|>ipython<|end_header_id|>

"25 C"<|eot_id|><|start_header_id|>assistant<|end_header_id|>

```

##### Model Response Format

```
The weather in San Francisco is 25 C.<|eot_id|>
```

##### Notes

- The output of the function call is provided back to the model as a tool response ( in json format ).
- Notice `<|start_header_id|>ipython<|end_header_id|>` as the header message preceding the tool response.
- The model finally summarizes the information from the tool response and returns the result to the user.

## Prompt format for base models

For base models (Llama3.2-1B and Llama3.2-3B), the prompt format for a simple completion is as
```

follows

Input Prompt Format

```
<|begin_of_text|>The color of the sky is blue but sometimes it can also be
```

Model Response Format

```
 gray or even purple. The color of the sky can change depending on the time of day, the weather, and the amount of pollution in the air. The color of the sky can also be affected by the presence of dust, smoke, and other particles in the air.

## Step 1: Identify the factors that
```

Notes

Same as Llama3.1

Thank You!