Tabla de contenidos

Llama 3.1

Tarjetas modelo y formatos de indicaciones

Llama 3.1

Introducción

Esta página describe el formato de solicitud para Llama 3.1 con énfasis en las nuevas características de esa versión.

Con el posterior lanzamiento de Llama 3.2, hemos introducido nuevos modelos ligeros en 1B y 3B y también modelos multimodales en 11B y 90B. Las primeras secciones de esta página (Plantilla de **solicitud**, **Solicitud de modelo base** e **Indicación de sugerencia de modelo**) son aplicables a todos los modelos publicados en Llama 3.1 y Llama 3.2.

Aproveche esta guía para aprovechar al máximo los nuevos modelos de Llama. Aunque las solicitudes diseñadas para Llama 3 deberían funcionar sin cambios en Llama 3.1 y Llama 3.2, le recomendamos que actualice sus solicitudes al nuevo formato para obtener los mejores resultados.

Modelo de tarjeta

Para obtener información técnica completa sobre la colección Llama 3.1 de modelos de lenguaje grande, consulte la tarjeta de modelo oficial, ubicada en GitHub.

Plantilla de aviso

Fichas especiales

<table>
<tr><td>Fichas</td><td>Descripción</td></tr>
<tr><td><|begin_of_text|></td><td>Especifica el inicio de la solicitud.</td></tr>
<tr><td><|end_of_text|></td><td>El modelo dejará de generar más tokens. Este token solo lo generan los modelos base.</td></tr>
</table>

`<|finetune_right_pad_id|>`	Este token se utiliza para rellenar secuencias de texto a la misma longitud en un lote.

`<|start_header_id|>`
`<|end_header_id|>`
	Estos tokens encierran el rol de un mensaje determinado. Los roles posibles son: `[system, user, assistant, and ipython]`

`<|eom_id|>`	Fin del mensaje. Un mensaje representa un posible punto de detención para la ejecución en el que el modelo puede informar al ejecutor de que es necesario realizar una llamada a la herramienta. Se utiliza para interacciones de varios pasos entre el modelo y las herramientas disponibles. Este token es emitido por el modelo cuando se utiliza la instrucción en el símbolo del sistema, o si el modelo requiere una herramienta integrada. `Environment: ipython`
`<|eot_id|>`	Fin del turno. Representa cuando el modelo ha determinado que ha terminado de interactuar con el mensaje del usuario que inició su respuesta. Esto se usa en dos escenarios: • al final de una interacción directa entre el modelo y el usuario • al final de múltiples interacciones entre el modelo y las herramientas disponibles This token signals to the executor that the model has finished generating a response.
`<|python_tag|>`	Special tag used in the model's response to signify a tool call.

Supported Roles

There are 4 different roles that are supported by Llama text models:

1. **`system`**: Sets the context in which to interact with the AI model. It typically includes rules, guidelines, or necessary information that help the model respond effectively.
2. **`user`**: Represents the human interacting with the model. It includes the inputs, commands, and questions to the model.
3. **`ipython`**: A new role introduced in Llama 3.1. Semantically, this role means "tool". This role is used to mark messages with the output of a tool call when sent back to the model

from the executor.

4. `assistant`: Represents the response generated by the AI model based on the context provided in the , and prompts. `system` `ipython` `user`

```
[system, assistant, user, ipython]
```

Llama 3.x Pretrained

This is the simplest way to prompt llama-based models, starting with text and allowing the model to generate new content based on the user-provided input.

```
<|begin_of_text|>{{ user_message}}
```

Llama 3.x Instruct

The format for a regular multi-turn conversation between a user and the model has not changed between Llama 3 and Llama 3.1. Here is a simple example of how it's formatted.

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

Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

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

What is the capital of France?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

For more examples of the prompt template, please refer to `text_prompt_format.md` in the GitHub repository. `meta-llama`

Code Interpreter

The code interpreter feature allows the model to generate executable Python code in response to the user query. The code interpreter is triggered when is included in the prompt. `Environment: ipython` `system`

Notes

- The model's response is wrapped in a and terminated with an tag. `<|python_tag|>` `<|eom_id|>`
- The indicates a continued multi-step reasoning. That is, the model is expecting a continuation message with the output of the tool call. `<|eom_id|>`

Sample Input:

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

Environment: ipython<|eot_id|><|start_header_id|>user<|end_header_id|>

Write code to check if number is prime, use that to see if the number 7 is
prime<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Response:

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

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

Note that *the model itself does not execute the code output*; you need to use our [llama-stack-apps](), or other similar framework to leverage code interpreters. When using , the results of the code are passed back to the model for further processing. `llama-stack-apps`

Tool Calling (8B/70B/405B)

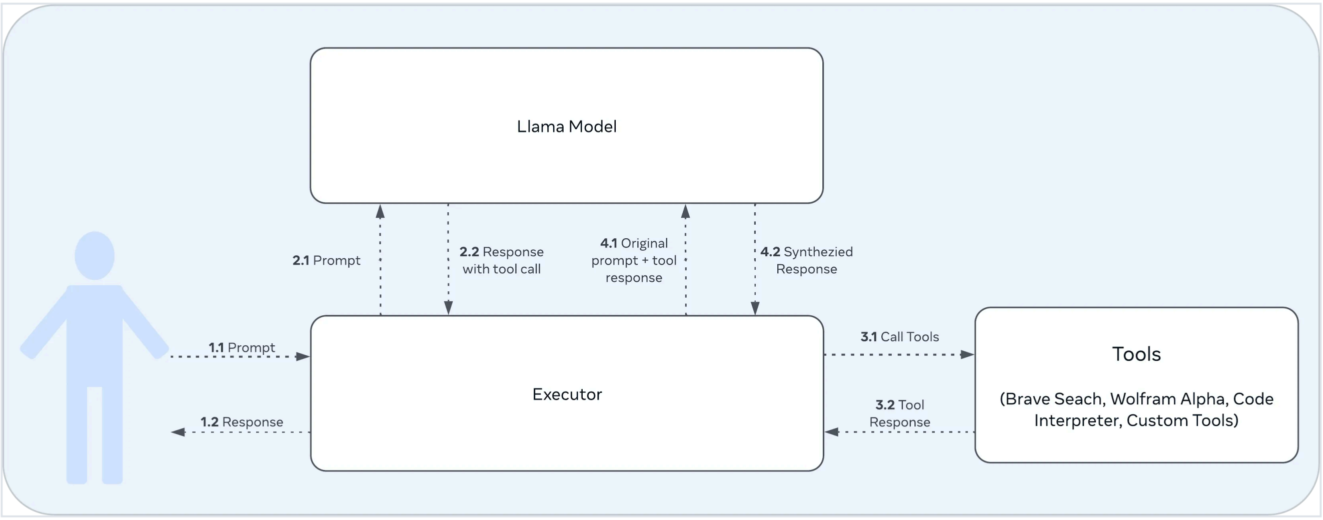
Tool-calling with the larger models can be done in three ways:

- Built-in Tools (Brave, Wolfram)
- JSON-based Tool Calling
- User-defined Custom Tool Calling

Note: We recommend using Llama 70B-instruct or Llama 405B-instruct for applications that combine conversation and tool calling. Llama 8B-Instruct can not reliably maintain a conversation alongside tool calling definitions. It can be used for zero-shot tool calling, but tool instructions should be removed for regular conversations between the model and the user.

Defining multiple tools in the system and/or user prompt can cause the model to generate multiple tool calls as a result of the user query. Developers must iterate on the prompts required to get the desired result and parse the result accordingly.

To better illustrate the flows below, this diagram shows the flow of information described:


Llama Model
2.1 Prompt
2.2 Response with tool call
4.1 Original prompt + tool response
4.2 Synthezied Response
1.1 Prompt
1.2 Response
Executor
3.1 Call Tools
3.2 Tool Response
Tools
(Brave Seach, Wolfram Alpha, Code Interpreter, Custom Tools)

User and assistant conversation

The format for a regular multi-turn conversation between a user and the model has not changed between Llama 3 and Llama 3.1. Here is a simple example of how it's formatted.

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

Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

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

What is the capital of France?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Built in Python based tool calling

The models are trained to use two built-in tools:

1. **Brave Search:** Tool call to perform web searches.
2. **Wolfram Alpha:** Tool call to perform complex mathematical calculations.

These can be turned on using the system prompt:

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

Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

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

What is the current weather in Menlo Park, California?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

A few things to note:

- Just including Environment: ipython turns on code interpreter; therefore, you don't need to specify code interpretation on the Tools: line. The model can generate python code which is interpreted by the executor, with the result provided back to the model.
- The message body of the assistant response starts with a special tag <|python_tag|>
- As alluded to above, in such an environment, the model can generate <|eom_id|> instead of just the standard <|eot_id|> . The latter indicates the turn is finished, while the former indicates continued multi-step reasoning. That is, the model is expecting a continuation message with the output of the tool call.

The built-in tool is called using *python syntax*, and not JSON format as for zero-shot tools. The python call that the model generates for each of the tools is of the form:

```
# for Search
<|python_tag|>
brave_search.call(query="...")
<|eom_id|>

# for Wolfram
<|python_tag|>
wolfram_alpha.call(query="...")
<|eom_id|>
```

To create custom tool calls using the zero-shot learning capability of the model, you can provide instructions for the tool definition in the system prompt along with the format you want the model to produce for the call. A working example is provided in the llama-agentic-system GitHub repository. You can check it out here.

The example below shows an end-to-end interaction of prompts between the user ⇔ model ⇔ tool. This sample system uses the in-built tool. (Please note: The model does not execute the call to the tool; you need to use our llama-agentic-system, or other similar framework, to leverage the tool-calling capabilities.) `wolfram_alpha`

Step - 1 User Prompt & System prompts

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

Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

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

Can you help me solve this equation: x^3 - 4x^2 + 6x - 24 = 0<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Step - 2 Model determining which tool to call

```
<|python_tag|>wolfram_alpha.call(query="solve x^3 - 4x^2 + 6x - 24 = 0")
<|eom_id|>
```

Step - 3 Response generated by tool, that is, Wolfram Alpha.

```
{
  "queryresult": {
    "success": true,
    "inputstring": "solve x^3 - 4x^2 + 6x - 24 = 0",
    "pods": [
      {
        "title": "Input interpretation",
        "subpods": [
          {
            "title": "",
            "plaintext": "solve x^3 - 4 x^2 + 6 x - 24 = 0"
          }
        ]
      },
      {
        "title": "Results",
        "primary": true,
        "subpods": [
          {
            "title": "",
            "plaintext": "x = 4"
          },
          {
            "title": "",
            "plaintext": "x = ± (i sqrt(6))"
          }
        ]
      },
      ...
    ]
  }
}
```

Step - 4 Reprompt Model with tool response

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

Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

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


Can you help me solve this equation: x^3 - 4x^2 + 6x - 24 = 0<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

<|python_tag|>wolfram_alpha.call(query="solve x^3 - 4x^2 + 6x - 24 = 0")
<|eom_id|><|start_header_id|>ipython<|end_header_id|>

{"queryresult": {"success": true, "inputstring": "solve x^3 - 4x^2 + 6x - 24 =
0", "pods": [{"title": "Input interpretation", "subpods": [{"title": "",
"plaintext": "solve x^3 - 4 x^2 + 6 x - 24 = 0"}]}, {"title": "Results",
"primary": true, "subpods": [{"title": "", "plaintext": "x = 4"}, {"title": "",
"plaintext": "x = \u00b1 (i sqrt(6))"}]}, ... ]}}<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Step - 5 Response from Agent to User

```
The solutions to the equation x^3 - 4x^2 + 6x - 24 = 0 are x = 4 and x = ±
(i√6).<|eot_id|>
```

JSON based tool calling

Meta Llama models can now output custom tool calls from a single message to allow easier tool calling. The following prompts provide an example of how custom tools can be called from the output of the model. It's important to note that the model itself does not execute the calls; it provides structured output to facilitate calling by an executor. An example executor can be found in the llama-agentic-system. The tool format is similar to the OpenAI definition and can be adjusted to better suit your needs, as noted in the following section.

Notes

- With custom tool calling, for the model to output the , the following instruction needs to be added to the system prompt: . Else, it should output . `eom_id` `Environment: ipython` `eot_id`
- The system prompt needs to be adjusted to inform the model how to deal with tool call outputs
- The tool definition is provided in the user prompt, as that is how the model was trained for the built in JSON tool calling. However, it's possible to provide the tool definition in the system prompt as well—and get similar results. Developers must test which way works best for their use case.

Step - 1 User prompt with custom tool details

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

Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

When you receive a tool call response, use the output to format an answer to
the orginal user question.

You are a helpful assistant with tool calling capabilities.<|eot_id|>
<|start_header_id|>user<|end_header_id|>

Given the following functions, please respond with a JSON for a function call
with its proper arguments that best answers the given prompt.

Respond in the format {"name": function name, "parameters": dictionary of
argument name and its value}. Do not use variables.
```

```
{
    "type": "function",
    "function": {
    "name": "get_current_conditions",
    "description": "Get the current weather conditions for a specific
location",
    "parameters": {
        "type": "object",
        "properties": {
        "location": {
            "type": "string",
            "description": "The city and state, e.g., San Francisco, CA"
        },
        "unit": {
            "type": "string",
            "enum": ["Celsius", "Fahrenheit"],
            "description": "The temperature unit to use. Infer this from the
user's location."
        }
        },
        "required": ["location", "unit"]
    }
    }
}

Question: what is the weather like in San Fransisco?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Step - 2 Model determining which tool to call

Note:

- As we are not using the instruction in the system prompt, the model ends generation with the token, not the token `Environment: ipython` `eot_id` `eom_id`

```
{"name": "get_current_conditions", "parameters": {"location": "San Francisco,
CA", "unit": "Fahrenheit"}}<eot_id>
```

Step - 3 Result from calling the tool is passed back to the model

When the result of the tool call is sent back to the model, we use the new role, , in the header. `ipython`

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

You are a helpful assistant with tool calling capabilities. When you receive a tool call response, use the output to format an answer to the orginal use question.<|eot_id|><|start_header_id|>user<|end_header_id|>

Given the following functions, please respond with a JSON for a function call with its proper arguments that best answers the given prompt.

Respond in the format {"name": function name, "parameters": dictionary of argument name and its value}. Do not use variables.

{
    "type": "function",
    "function": {
    "name": "get_current_conditions",
    "description": "Get the current weather conditions for a specific location",
    "parameters": {
        "type": "object",
        "properties": {
        "location": {
            "type": "string",
            "description": "The city and state, e.g., San Francisco, CA"
        },
        "unit": {
            "type": "string",
            "enum": ["Celsius", "Fahrenheit"],
            "description": "The temperature unit to use. Infer this from the user's location."
        }
        },
        "required": ["location", "unit"]
    }
    }
}

Question: what is the weather like in San Fransisco?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

{"name": "get_current_conditions", "parameters": {"location": "San Francisco, CA", "unit": "Fahrenheit"}}<|eot_id|>
<|start_header_id|>ipython<|end_header_id|>

{"output": "Clouds giving way to sun Hi: 76° Tonight: Mainly clear early, then areas of low clouds forming Lo: 56°"}<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Step - 4 The model generates the final response the user

```
The weather in Menlo Park is currently cloudy with a high of 76° and a low of
56°, with clear skies expected tonight.<eot_id>
```

User-defined Custom tool calling

The following prompts provide an example of how to define your own format in the system prompt and make use of it while interacting with the model. This format will use a new and custom <function> tag to enclose a JSON object with the function parameters.

Notes

- The model uses the token because of the instruction. `eom_id` `Environment: ipython`
- The system prompt defines the custom format as well as the tools the model has available.

Step - 1 System prompt with custom tool details

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


Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

# Tool Instructions
- Always execute python code in messages that you share.
- When looking for real time information use relevant functions if available
else fallback to brave_search



You have access to the following functions:

Use the function 'spotify_trending_songs' to: Get top trending songs on Spotify
{
  "name": "spotify_trending_songs",
  "description": "Get top trending songs on Spotify",
  "parameters": {
    "n": {
      "param_type": "int",
      "description": "Number of trending songs to get",
      "required": true
    }
  }
}



If a you choose to call a function ONLY reply in the following format:
<{start_tag}={function_name}>{parameters}{end_tag}
where

start_tag => `<function`
parameters => a JSON dict with the function argument name as key and function
argument value as value.
end_tag => `</function>`

Here is an example,
<function=example_function_name>{"example_name": "example_value"}</function>

Reminder:
- Function calls MUST follow the specified format
- Required parameters MUST be specified
- Only call one function at a time
- Put the entire function call reply on one line
- Always add your sources when using search results to answer the user query

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

Can you check the top 5 trending songs on spotify?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Step - 2 Model determining which tool to call

Now the model responds in accordance with the format defined in the system prompt, with the function name in the start tag, and the parameters, as a JSON object, in between the start and end tags.

Note:

- As we are using the instruction in the system prompt, the model ends generation with the token. `Environment: ipython` `eom_id`

```
<function=spotify_trending_songs>{"n": "5"}</function><eom_id>
```

Step - 3 Result from calling the tool is passed back to the model

When the result of the tool call is sent back to the model, we use the new role, , in the header. `ipython`

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


Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

# Tool Instructions
- Always execute python code in messages that you share.
- When looking for real time information use relevant functions if available
else fallback to brave_search



You have access to the following functions:

Use the function 'spotify_trending_songs' to: Get top trending songs on Spotify
{"name": "spotify_trending_songs", "description": "Get top trending songs on
Spotify", "parameters": {"n": {"param_type": "int", "description": "Number of
trending songs to get", "required": true}}}


If a you choose to call a function ONLY reply in the following format:
<{start_tag}={function_name}>{parameters}{end_tag}
where

start_tag => `<function`
parameters => a JSON dict with the function argument name as key and function
argument value as value.
end_tag => `</function>`

Here is an example,
<function=example_function_name>{"example_name": "example_value"}</function>

Reminder:
- Function calls MUST follow the specified format
- Required parameters MUST be specified
- Only call one function at a time
- Put the entire function call reply on one line
- Always add your sources when using search results to answer the user query

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

Can you check the top 5 trending songs on spotify?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

```
<function=spotify_trending_songs>{"n": "5"}</function><|eom_id|>
<|start_header_id|>ipython<|end_header_id|>

["1. BIRDS OF A FEATHER by Billie Eilish", "2. Espresso by Sabrina Carpenter",
"3. Please Please Please by Sabrina Carpenter", "4. Not Like Us by Kendrick
Lamar", "5. Gata Only by FloyyMenor, Cris Mj"]<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Step - 4 The model generates the final response the user

```
The top 5 trending songs on Spotify are:

1. BIRDS OF A FEATHER by Billie Eilish
2. Espresso by Sabrina Carpenter
3. Please Please Please by Sabrina Carpenter
4. Not Like Us by Kendrick Lamar
5. Gata Only by FloyyMenor, Cris Mj<|eot_id|>
```

Was this page helpful?

Yes

No

Overview

Models

Getting the Models

Running Llama

How-To Guides

Integration Guides

Community Support

Community ⌄

Resources ⌄

Trust & Safety ⌄