



Documentation

Text



Chat Completion Models

The Groq Chat Completions API processes a series of messages and generates output responses. These models can perform multi-turn discussions or tasks that require only one interaction.

For details about the parameters, [visit the reference page](#).

JSON mode (*beta*)

JSON mode is a beta feature that guarantees all chat completions are valid JSON.

Usage:

1. Set "response_format": {"type": "json_object"} in your chat completion request
2. Add a description of the desired JSON structure within the system prompt (see below for example system prompts)

Recommendations for best beta results:

- Mixtral performs best at generating JSON, followed by Gemma, then Llama
- Use pretty-printed JSON instead of compact JSON
- Keep prompts concise

Beta Limitations:

- Does not support streaming
- Does not support stop sequences

Error Code:

- Groq will return a 400 error with an error code of json_validate_failed if JSON generation fails.

Example system prompts:

```
You are a legal advisor who summarizes documents in JSON
```



```
You are a data analyst API capable of sentiment analysis that responds in JSON. The JSON schema should ir
{
  "sentiment_analysis": {
    "sentiment": "string (positive, negative, neutral)",
    "confidence_score": "number (0-1)"
    # Include additional fields as required
  }
}
```



Generating Chat Completions with groq SDK

Code Overview

Python JavaScript



```
pip install groq
```

Performing a basic Chat Completion

```

1 from groq import Groq
2
3 client = Groq()
4
5 chat_completion = client.chat.completions.create(
6     #
7     # Required parameters
8     #
9     messages=[
10         # Set an optional system message. This sets the behavior of the
11         # assistant and can be used to provide specific instructions for
12         # how it should behave throughout the conversation.
13         {
14             "role": "system",
15             "content": "you are a helpful assistant."
16         },
17         # Set a user message for the assistant to respond to.
18         {
19             "role": "user",
20             "content": "Explain the importance of fast language models"
21         }
22     ]
23 )

```

Streaming a Chat Completion

To stream a completion, simply set the parameter `stream=True`. Then the completion function will return an iterator of completion deltas rather than a single, full completion.

```

24 # The language model which will generate the completion.
25 model="llama-3.3-70b-versatile",
26
27 from groq import Groq
28 # Optional parameters
29 client = Groq()
30
31 stream = client.chat.completions.create(messages=[
32     # As the temperature approaches zero, the model will become deterministic
33     # Reduce randomness
34     temperature=0.5,
35     messages=[
36         # The maximum number of tokens to generate. The request behavior of the
37         # 32768 tokens and read between prompt and completion instructions for
38         max_tokens=1024,
39         {
40             # Control is over system, nucleus sampling: 0.5 means half of all
41             # likelihood weights are sampled.
42             top_p=1,
43             # Set a user message for the assistant to respond to.
44             # A stop sequence is a predefined or user-specified text string that
45             # signals the AI to stop generating content, ensuring its responses
46             # remain "focused" and doesn't wander off into unrelated topics.
47             # markers like "[end]".
48             stop=None,

```

Performing a Chat Completion with a stop sequence

```

47     # markers like "[end]".
48     stop=None,
49
50 from groq import Groq
51 model="llama-3.3-70b-versatile",
52 client = Groq()
53 #
54 chat_completion=client.chat.completions.create(

```

```

20 # Print the completion returned by the LLM.
21 print(RequiredParameters[0].message.content)
22 # Controls randomness: lowering results in less random completions.
23 # As the temperature approaches zero, the model will become deterministic
24 # and repeat previous outputs. This sets the behavior of the
25 temperature, and can be used to provide specific instructions for
26 # how it should behave throughout the conversation.
27 # The maximum number of tokens to generate. Requests can use up to
28 # 2048 tokens shared between prompt and completion.
29 max_completion_tokens=1024, "you are a helpful assistant."
30 },
31 # Controls diversity via nucleus sampling: 0.5 means half of all
32 # likelihood-weighted options are considered.

```

Performing an Async Chat Completion

Simply use the Async client to enable async:

```

20 # "content": "Count to 10. Your response must begin with \"1, \". example: 1, 2, 3, ...",
21 # A stop sequence is a predefined or user-specified text string that
22 #, signals an AI to stop generating content, ensuring its responses
23 # remain focused and concise. Examples include punctuation marks and
24 # markers like "[end]". "Explain the importance of fast language models",
25 # If set, partial message deltas will be sent.
26 # Optional parameters
27 async def main():
28     client = AsyncGroq()
29     # Print the completion returned by the LLM.
30     for # Ask the temperature to approach zero to get deterministic
31         print(RequiredParameters[0].delta.content, end="")
32         temperature=0.0, and can be used to provide specific instructions for
33         # how it should behave throughout the conversation.
34         # Controls diversity via nucleus sampling: 0.5 means half of all
35         # likelihood-weighted options are considered.
36         top_p=1, "content": "you are a helpful assistant."

```

Streaming an Async Chat Completion

```

21 # A stop sequence is a predefined or user-specified text string that
22 # signals an AI to stop generating content, ensuring its responses
23 # remain focused and concise. Examples include punctuation marks and
24 # markers like "[end]". "Explain the importance of fast language models",
25 # If multiple stop values are needed, an array of string may be passed,
26 # stop=[" ", "6", " ", "six", " ", "Six"]
27 async def main():
28     client = AsyncGroq(model="llama-3.3-70b-versatile",
29     # If set, partial message deltas will be sent.
30     stream=True)
31     # Optional parameters
32     # Required parameters
33     # Print the completion returned by the LLM.
34     print(client.chat.completions.create(
35         # Ask the temperature to approach zero to get deterministic
36         # behavior of the
37         # persistent and repetitive
38         temperature=0.0, and can be used to provide specific instructions for
39         # how it should behave throughout the conversation.
40         {
41             # The maximum number of tokens to generate. Requests can use up to
42             # 2048 tokens shared between prompt and completion.
43             max_completion_tokens=1024,
44             # Set a user message for the assistant to respond to.
45             # Controls diversity via nucleus sampling: 0.5 means half of all
46             # likelihood-weighted options are considered.

```

JSON Mode

```

24     top_p=1,"content": "Explain the importance of fast language models",
25 from typing import List, Optional
26 import json
27 # A stop sequence is a predefined or user-specified text string that
28 # signals an AI to stop generating content, ensuring its responses
29 # remain focused and concise. Examples include punctuation marks and
30 # markers like "[end]",
31 from pydantic import BaseModel
32 from groq import Groq
33 stop=None,
34 groq = Groq()
35 # Optional parameter message deltas will be sent.
36 #stream=False,
37 # Data model for LLM to generate
38 class Ingredient(BaseModel):
39     """A single ingredient in a recipe. Lowering results in less random completions.
40     # If set to True, the completion will be returned by the LLM. The model will become
41     # more deterministic and helpful.
42     quantity: Optional[str]
43     """
44     async def run(main())
45
46     # The maximum number of tokens to generate. Requests can use up to
47     # 32768 tokens shared between prompt and completion.
48     class Recipe(BaseModel):
49         """A recipe object with ingredients and directions.
50         # Controls diversity via nucleus sampling: 0.5 means half of all
51         # likelihood-weighted options are considered.
52         top_p=1,
53     def get_recipe(recipe_name: str) -> Recipe:
54         """A stop sequence is a predefined or user-specified text string that
55         # signals an AI to stop generating content, ensuring its responses
56         # remain focused and concise. Examples include punctuation marks and
57         # markers like "[end]",
58         stop=None, content": "You are a recipe database that outputs recipes in JSON.\n"
59         # Pass the json schema to the model. Pretty printing improves results.
60         # If set to True, the completion will be returned by the LLM. The model will become
61         # more deterministic and helpful.
62         stream=True,
63         {
64             "role": "user",
65             # Print the completion returned by the LLM.
66             "content": f"Get the recipe for {recipe_name}",
67         }
68         async for chunk in stream:
69             print(chunk.choices[0].delta.content, end="")
70             model="llama3-70b-8192",
71     async def run(main())
72     # Streaming is not supported in JSON mode
73     stream=False,
74     # Enable JSON mode by setting the response format
75     response_format={"type": "json_object"},
76 )
77 return Recipe.model_validate_json(chat_completion.choices[0].message.content)
78
79 def print_recipe(recipe: Recipe):
80     print("Recipe:", recipe.recipe_name)
81
82     print("\nIngredients:")
83     for ingredient in recipe.ingredients:
84         print(
85             f"- {ingredient.name}: {ingredient.quantity} {ingredient.quantity_unit or ''}"
86         )
87     print("\nDirections:")
88     for step, direction in enumerate(recipe.directions, start=1):
89         print(f"{step}. {direction}")
90
91 recipe = get_recipe("apple pie")
92 print_recipe(recipe)

```