Documentation

Chat Completions

 \Diamond

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
- 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
    }
}
```

Python JavaScript

```
pip install groq
```

Performing a basic Chat Completion

```
1 from groq import Groq
 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
            {
                "role": "system",
14
                "content": "you are a helpful assistant."
15
16
            },
            # Set a user message for the assistant to respond to.
17
18
19
                "role": "user",
                "content": "Explain the importance of fast language models",
20
```

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.

```
# The language model which will generate the completion.
```

```
25
        model="llama3-8b-8192",
26
21 from#groq import Groq
        # Optional parameters
28
29
  clie#t = Groq()
34
35 stre#mCenthodestrahdomnesspleioweringeres(lts in less random completions.
30
        # As the temperature approaches zero, the model will become deterministic
        # Baduiepdtparameters
33
38
        #emperature=0.5,
39
        messages=[
30
        # The maximumoptimbealosytakenmetaagenerate.sRequebesbeanvise of the
37
        # 32#768siskens andrednbeewweedpromprowidecompletioninstructions for
        max #ohewsi1024ould behave throughout the conversation.
32
19
40
        # Controlsodeversitytemä, nucleus sampling: 0.5 means half of all
        # likelihoodtmetghtegooparens beepfohsaderedant."
45
40
        top \beta = 1,
43
            # Set a user message for the assistant to respond to.
```

```
48
                   # A $top sequence is a predefined or user-specified text string that
                   # signalsreheAI tosetop generating content, ensuring its responses
Performing a Chart Completion with a step saguence defend tenging emmodel and
                   # markers like "[end]".
     27
     22
                   $top=None,
     29
            from groq import Groq
     20
                   # Ifiesetangpageimodmeswagehdwltasgwiehate thetcompletion.
     23
            cliemodea@#baame3-8b-8192",
     24
     23
           chat#completion = client.chat.completions.create(
     28
           # Pr#nopthenedmpaeamentereturned by the LLM.
     29
            prin#(Requiredplatametehsices[0].message.content)
     30
     3⊈
                   #e€ongeol$ randomness: lowering results in less random completions.
     30
                   # As#tBettempepaionelapystemhmeszege, Thesmodes whelbbbaumordefetmenistic
     33
                   # an# aspesianteand can be used to provide specific instructions for
     34
                   temp#rhowret0should behave throughout the conversation.
     39
     34
                   # The maximum"mumbestem"tokens to generate. Requests can use up to
     37
                   # 2048 tökenteshäredybetweenaphedpfuåndssompåetion.
     38
                   max_tokens=1024,
     39
                          # Set a user message for the assistant to respond to.
     40
                   # Controls diversity via nucleus sampling: 0.5 means half of all
                   # likelihoodeweighted"options are considered.
Performing an Async Chat Completion top_p=1, "content": "Count to 10.
                                                                            Your response must begin with \"1, \". example: 1, 2, 3, ...",
Simply use the Async client to enable asyncio
     44
                   #,A stop sequence is a predefined or user-specified text string that
     43
                   # signals an AI to stop generating content, ensuring its responses
     44
                   # Thealangospeedoded whichsailExampleatenthudeomphetwohion marks and
     4$
           impomtwatkeriahike86f8nd2",
     48
                   stop=None,
            from#groq import AsyncGroq
     49
     20
                   # Offtsetalppatametmessage deltas will be sent.
     55
                   #tream=True,
            async def main():
     30
     5Z
                   #lcontrolAsศตล์domqess: lowering results in less random completions.
     58
           # Pr#nAsthbeinempmentaledeppmoachturmedobythbembdMl will become deterministic
     59
           for #hankcempletreame= await client.chat.completions.create(
     10
                   pemp#fatunk=0h5ices[0].delta.content, end="")
     14
                          # Required parameters
     38
                   # Th∉ maximum number of tokens to generate. Requests can use up to
     1,3
                   # 2048 មិខិន្តិខិត្ត shared between prompt and completion.
                   max toke株sSet24n optional system message. This sets the behavior of the
     14
                                 # assistant and can be used to provide specific instructions for
     35
                   # Contro#sh@\verts\@\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\delta\del
     46
     47
                   # likelihood-weighted options are considered.
     48
                                        "role": "system",
Streaming an Async Chat Completion
                   # A stop}sequence is a predefined or user-specified text string that
     25
                   # signal# SATA# #6estmpsgagerferntheoateitfaghstpingspendrtoponses
     48
                   # remain{focused and concise. Examples include punctuation marks and
     4∌
            impoftmarkeriolikeoleend user",
     48
                   # For this examplentwe wExplase theeimportan€eta€ fiamtstanguaganeodglat,5.
           from #gfqqmingofte Astop Gvalues are needed, an array of string may be passed,
     29
     38
                   # stop=[", 6", ", six", ", Six"]
     37
                   stop=", 6"
           async de# ๆกล่าโล่ก่guage model which will generate the completion.
     7
                   client = AsyncGroq()
```

```
28
           # Ifmodel=parama3-86s8ag2"deltas will be sent.
   50
           stream=Faaweit client.chat.completions.create(
   30
                # Optioned parameters
   38
       # Print #he completion returned by the LLM.
   32
       print(chatscogpsefion.choices[0].message.content)
   38
   34
                # Co#t6elsanaodtmoeas:systemimgsregeltThis sets thedbehavmopletfothe
   35
                # As#thesfemperatudecappbeauhed terprothdemodeliwillibetometions for
                # de#ehmwinistshoand behavetikeoughout the conversation.
   38
   38
                temp∉rature=0.5,
                        "role": "system",
   38
                # The maximumberyof aokeashedpfedeastestRequests can use up to
JSON Mode
                # 2048 tokens shared between prompt and completion.
   22
                max #oSetsa103er message for the assistant to respond to
   42
       from typingontpotioueversuspingalnucleus sampling: 0.5 means half of all
   <u>a</u>
       import j#ofikelihoodtwefghtedxphaiontherempontadeeedf fast language models",
   25
   2B
                top \beta=1,
       from pydantic import BaseModel
   <u>a</u>8
       from group Amport Gradience is a predefined or user-specified text string that
   25
   28
                # ThighalangungAImodeltwhighnwialingneontentheenouplegions responses
       groq = G#00@mainlamans8b-8nd2concise. Examples include punctuation marks and
   20
   50
                # markers like "[end]".
   52
                #top=None,
       # Data madeltenatematameterate
   19
       class Ingredient(Bpartodelmessage deltas will be sent.
   34
   <u>1</u>2
           name;tream=False,
   13B
           ງພລກ#i&&ntテ️ols randomness: lowering results in less random completions.
   38
           quan#iAy_thettemperataresapproaches zero, the model will become
   35
           # Pr#ndethemiompletiondretpetedibg.the LLM.
   38
           print(mbatatomp10t5on.choices[0].message.content)
   10
       class Recipe(BaseModel):
   18
       asyncising and a syncising number of tokens to generate. Requests can use up to
   49
           ingrad2048stokens[Ingredicativeen prompt and completion.
   49
           diremaionskehisi654r]
   23
   22
                # Controls diversity via nucleus sampling: 0.5 means half of all
   <u>2</u>3
       def get_#efiRefinoonewelghtedtoptionBeareeconsidered.
           chat_66mpletion = groq.chat.completions.create(
   48
   45
                messages=[
   48
                # A stop sequence is a predefined or user-specified text string that
   <u>4</u>3
                # signals rale it system generating content, ensuring its responses
   38
                # remain F80tte0t and Y08nerse Example database that coutevits marke sand JSON.\n"
                # markers Page then ison schema to the model. Pretty printing improves results.
   39
                stop=None, The JSON object must use the schema: {json.dumps(Recipe.model_json_schema(), indent
   39
                    },
   33
   32
                # If{\rm set}, partial message deltas will be sent.
                stream=Truele": "user",
   33
                        "content": f"Fetch a recipe for {recipe_name}",
   38
   35
                    },
   38
           # Print the incremental deltas returned by the LLM.
   33
           asyncodel="llama3n8bt8192",
   28
                FFTR€(@ħUĥR=@hoices[0].delta.content, end="")
                # Streaming is not supported in JSON mode
   39
   49
       asyncio. Fthe Amife se,
                # Enable JSON mode by setting the response format
   41
   42
                response format={"type": "json object"},
   43
           )
```

```
44
        return Recipe.model_validate_json(chat_completion.choices[0].message.content)
45
46
47 def print_recipe(recipe: Recipe):
       print("Recipe:", recipe.recipe_name)
48
49
       print("\nIngredients:")
50
       for ingredient in recipe.ingredients:
51
52
           print(
               f"- {ingredient.name}: {ingredient.quantity} {ingredient.quantity_unit or ''}"
53
54
           )
55
      print("\nDirections:")
       for step, direction in enumerate(recipe.directions, start=1):
56
           print(f"{step}. {direction}")
57
58
59
60 recipe = get_recipe("apple pie")
61 print_recipe(recipe)
```