Search ⌘ K

Docs API Reference

GETTING STARTED
Overview
Quickstart
Models
OpenAI Compatibility
Rate Limits
Templates
API Reference

CORE FEATURES
Text Generation
Speech to Text
Text to Speech
OCR and Image Recognition
Reasoning
Content Moderation
Structured Outputs
Prompt Caching

TOOLS & INTEGRATIONS
Tool Use
Integrations Catalog
Coding with Groq

COMPOUND (AGENTIC AI)
Overview
Systems
Use Cases

GUIDES
Prompting Guide
Assistant Message Prefilling

SERVICE TIERS
Service Tiers
Performance Tier
Flex Processing
Batch Processing

Text Generation

Copy page

Generating text with Groq's Chat Completions API enables you to have natural, conversational interactions with Groq's large language models. It processes a series of messages and generates human-like responses that can be used for various applications including conversational agents, content generation, task automation, and generating structured data outputs like JSON for your applications.

Chat Completions

Chat completions allow your applications to have dynamic interactions with Groq's models. You can send messages that include user inputs and system instructions, and receive responses that match the conversational context.

Chat models can handle both multi-turn discussions (conversations with multiple back-and-forth exchanges) and single-turn tasks where you need just one response.

For details about all available parameters, visit the API reference page.

Getting Started with Groq SDK

To start using Groq's Chat Completions API, you'll need to install the Groq SDK and set up your API key.

Python JavaScript

shell

```shell
npm install --save groq-sdk
```

Performing a Basic Chat Completion

The example below shows how to send messages to the model and receive a complete response. The messages array should include user queries and can optionally include system messages to guide the model's behavior.

JavaScript

```javascript
import Groq from "groq-sdk";

const groq = new Groq();

export async function main() {
  const completion = await getGroqChatCompletion();
  console.log(completion.choices[0]?.message?.content || "");
}

export const getGroqChatCompletion = async () => {
  return groq.chat.completions.create({
    messages: [
      // Set an optional system message. This sets the behavior of the
```

```
        // assistant and can be used to provide specific instructions for
        // how it should behave throughout the conversation.
        {
          role: "system",
          content: "You are a helpful assistant.",
        },
        // Set a user message for the assistant to respond to.
        {
          role: "user",
          content: "Explain the importance of fast language models",
        },
      ],
      model: "openai/gpt-oss-20b",
    });
  };

main();
```

Streaming allows you to receive and process the model's response token by token as it's being generated. This creates a more interactive experience and can reduce perceived latency.

To stream a completion, set the parameter `stream=true` and process the response chunks as they arrive.

JavaScript

```javascript
import Groq from "groq-sdk";

const groq = new Groq();

export async function main() {
  const stream = await getGroqChatStream();
  for await (const chunk of stream) {
    // Print the completion returned by the LLM.
    process.stdout.write(chunk.choices[0]?.delta?.content || "");
  }
}

export async function getGroqChatStream() {
  return groq.chat.completions.create({
    //
    // Required parameters
    //
    messages: [
      // Set an optional system message. This sets the behavior of the
      // assistant and can be used to provide specific instructions for
      // how it should behave throughout the conversation.
      {
        role: "system",
        content: "You are a helpful assistant.",
      },
      // Set a user message for the assistant to respond to.
      {
        role: "user",
        content: "Explain the importance of fast language models",
      },
    ],

    // The language model which will generate the completion.
    model: "openai/gpt-oss-20b",

```

JavaScript

```javascript
import Groq from "groq-sdk";
                                                                    m completions.
const groq = new Groq();                                            become deterministic
```

```javascript

export async function main() {
  const stream = await getGroqChatStream();
  for await (const chunk of stream) {
    // Print the completion returned by the LLM.
    process.stdout.write(chunk.choices[0]?.delta?.content || "");
  }
}

export async function getGroqChatStream() {
  return groq.chat.completions.create({
    //
    // Required parameters
    //
    messages: [
      // Set an optional system message. This sets the behavior of the
      // assistant and can be used to provide specific instructions for
      // how it should behave throughout the conversation.
      {
        role: "system",
        content: "You are a helpful assistant.",
      },
      // Set a user message for the assistant to respond to.
      {
        role: "user",
        content:
          "Start at 1 and count to 10.  Separate each number with a comma and a space",
      },
    ],

    // The language model which will generate the completion.
    model: "llama-3.3-70b-versatile",

    //
    // Optional parameters
    //

    // Controls randomness: lowering results in less random completions.
    // As the temperature approaches zero, the model will become deterministic
    // and repetitive.
    temperature: 0.5,

    // The maximum number of tokens to generate. Requests can use up to
    // 2048 tokens shared between prompt and completion.
    max_completion_tokens: 1024,

    // Controls diversity via nucleus sampling: 0.5 means half of all
    // likelihood-weighted options are considered.
    top_p: 1,

    // A stop sequence is a predefined or user-specified text string that
    // signals an AI to stop generating content, ensuring its responses
    // remain focused and concise. Examples include punctuation marks and
    // markers like "[end]".
    //
    // For this example, we will use ", 6" so that the llm stops counting at 5.
    // If multiple stop values are needed, an array of string may be passed,
    // stop: [", 6", ", six", ", Six"]
    stop: ", 6",

    // If set, partial message deltas will be sent.
    stream: true,
  });
}

main();
```

can use up to
half of all
ext string that
its responses
ation marks and