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

COMPOUND (AGENTIC AI)
Overview
Systems
Use Cases

GUIDES
Prompting Guide
Assistant Message Prefilling

SERVICE TIERS
Service Tiers
Performance Tier
Flex Processing
Batch Processing

ADVANCED
LoRA Inference

PRODUCTION READINESS
Production Checklist
Optimizing Latency
Security Onboarding
Prometheus Metrics

ACCOUNT AND CONSOLE
Spend Limits
Projects
Model Permissions

Speech to Text

Copy page

Groq API is designed to provide fast speech-to-text solution available, offering OpenAI-compatible endpoints that enable near-instant transcriptions and translations. With Groq API, you can integrate high-quality audio processing into your applications at speeds that rival human interaction.

API Endpoints

We support two endpoints:

ENDPOINT	USAGE	API ENDPOINT
Transcriptions	Convert audio to text	`https://api.groq.com/openai/v1/audio/transcriptions`
Translations	Translate audio to English text	`https://api.groq.com/openai/v1/audio/translations`

Supported Models

MODEL ID	MODEL	SUPPORTED LANGUAGE(S)	DESCRIPTION
`whisper-large-v3-turbo`	Whisper Large V3 Turbo	Multilingual	A fine-tuned version of a pruned Whisper Large V3 designed for fast, multilingual transcription tasks.
`whisper-large-v3`	Whisper Large V3	Multilingual	Provides state-of-the-art performance with high accuracy for multilingual transcription and translation tasks.

Which Whisper Model Should You Use?

Having more choices is great, but let's try to avoid decision paralysis by breaking down the tradeoffs between models to find the one most suitable for your applications:

- If your application is error-sensitive and requires multilingual support, use `whisper-large-v3` .
- If your application requires multilingual support and you need the best price for performance, use `whisper-large-v3-turbo` .

The following table breaks down the metrics for each model.

MODEL	COST PER HOUR	LANGUAGE SUPPORT	TRANSCRIPTION SUPPORT	TRANSLATION SUPPORT	REAL-TIME SPEED FACTOR	WORD ERROR RATE
`whisper-large-v3`	$0.111	Multilingual	Yes	Yes	189	10.3%
`whisper-large-v3-turbo`	$0.04	Multilingual	Yes	No	216	12%

Working with Audio Files

Audio File Limitations

Max File Size	25 MB (free tier), 100MB (dev tier)
Max Attachment File Size	25 MB. If you need to process larger files, use the `url` parameter to specify a url to the file instead.
Minimum File Length	0.01 seconds
Minimum Billed Length	10 seconds. If you submit a request less than this, you will still be billed for 10 seconds.
Supported File Types	Either a URL or a direct file upload for `flac`, `mp3`, `mp4`, `mpeg`, `mpga`, `m4a`, `ogg`, `wav`, `webm`

Single Audio Track	Only the first track will be transcribed for files with multiple audio tracks. (e.g. dubbed video)
Supported Response Formats	`json`, `verbose_json`, `text`
Supported Timestamp Granularities	`segment`, `word`

Audio Preprocessing

Our speech-to-text models will downsample audio to 16KHz mono before transcribing, which is optimal for speech recognition. This preprocessing can be performed client-side if your original file is extremely large and you want to make it smaller without a loss in quality (without chunking, Groq API speech-to-text endpoints accept up to 25MB for free tier and 100MB for dev tier). For lower latency, convert your files to `wav` format. When reducing file size, we recommend FLAC for lossless compression.

The following `ffmpeg` command can be used to reduce file size:

shell

```
ffmpeg \
  -i <your file> \
  -ar 16000 \
  -ac 1 \
  -map 0:a \
  -c:a flac \
  <output file name>.flac
```

Working with Larger Audio Files

For audio files that exceed our size limits or require more precise control over transcription, we recommend implementing audio chunking. This process involves:

1. Breaking the audio into smaller, overlapping segments
2. Processing each segment independently
3. Combining the results while handling overlapping

To learn more about this process and get code for your own implementation, see the complete audio chunking tutorial in our Groq API Cookbook.

Using the API

The following are request parameters you can use in your transcription and translation requests:

PARAMETER	TYPE	DEFAULT	DESCRIPTION
`file`	`string`	Required unless using `url` instead	The audio file object for direct upload to translate/transcribe.
`url`	`string`	Required unless using `file` instead	The audio URL to translate/transcribe (supports Base64URL).
`language`	`string`	Optional	The language of the input audio. Supplying the input language in ISO-639-1 (i.e. `en,` tr`) format will improve accuracy and latency. The translations endpoint only supports 'en' as a parameter option.
`model`	`string`	Required	ID of the model to use.
`prompt`	`string`	Optional	Prompt to guide the model's style or specify how to spell unfamiliar words. (limited to 224 tokens)
`response_format`	`string`	json	Define the output response format. Set to `verbose_json` to receive timestamps for audio segments. Set to `text` to return a text response.
			The temperature between 0 and 1. For

`temperature`	`float`	0	translations and transcriptions, we recommend the default value of 0.
`timestamp_granularities[]`	`array`	segment	The timestamp granularities to populate for this transcription. `response_format` must be set `verbose_json` to use timestamp granularities. Either or both of `word` and `segment` are supported. `segment` returns full metadata and `word` returns only word, start, and end timestamps. To get both word-level timestamps and full segment metadata, include both values in the array.

Example Usage of Transcription Endpoint

The transcription endpoint allows you to transcribe spoken words in audio or video files.

Python JavaScript curl

The Groq SDK package can be installed using the following command:

shell

```
pip install groq
```

The following code snippet demonstrates how to use Groq API to transcribe an audio file in Python:

Python

```python
import os
import json
from groq import Groq

# Initialize the Groq client
client = Groq()

# Specify the path to the audio file
filename = os.path.dirname(__file__) + "/YOUR_AUDIO.wav" # Replace with your aud

# Open the audio file
with open(filename, "rb") as file:
    # Create a transcription of the audio file
    transcription = client.audio.transcriptions.create(
      file=file, # Required audio file
      model="whisper-large-v3-turbo", # Required model to use for transcription
      prompt="Specify context or spelling",  # Optional
      response_format="verbose_json",  # Optional
      timestamp_granularities = ["word", "segment"], # Optional (must set respon
      language="en",  # Optional
      temperature=0.0  # Optional
    )
    # To print only the transcription text, you'd use print(transcription.text)
    print(json.dumps(transcription, indent=2, default=str))
```

Example Usage of Translation Endpoint

The translation endpoint allows you to translate spoken words in audio or video files to English.

Python JavaScript curl

The Groq SDK package can be installed using the following command:

shell

```
pip install groq
```

The following code snippet demonstrates how to use Groq API to translate an audio file in Python:

Python

```python
import os
from groq import Groq

# Initialize the Groq client
client = Groq()

# Specify the path to the audio file
filename = os.path.dirname(__file__) + "/sample_audio.m4a" # Replace with your a

# Open the audio file
with open(filename, "rb") as file:
    # Create a translation of the audio file
    translation = client.audio.translations.create(
      file=(filename, file.read()), # Required audio file
      model="whisper-large-v3", # Required model to use for translation
      prompt="Specify context or spelling",  # Optional
      language="en", # Optional ('en' only)
      response_format="json",  # Optional
      temperature=0.0  # Optional
    )
    # Print the translation text
    print(translation.text)
```

Understanding Metadata Fields

When working with Groq API, setting `response_format` to `verbose_json` outputs each segment of transcribed text with valuable metadata that helps us understand the quality and characteristics of our transcription, including `avg_logprob`, `compression_ratio`, and `no_speech_prob`.

This information can help us with debugging any transcription issues. Let's examine what this metadata tells us using a real example:

JSON


```json
{
  "id": 8,
  "seek": 3000,
  "start": 43.92,
  "end": 50.16,
  "text": " document that the functional specification that you started to read thr
  "tokens": [51061, 4166, 300, 264, 11745, 31256],
  "temperature": 0,
  "avg_logprob": -0.097569615,
  "compression_ratio": 1.6637554,
  "no_speech_prob": 0.012814695
}
```


As shown in the above example, we receive timing information as well as quality indicators. Let's gain a better understanding of what each field means:

- `id:8` : The 9th segment in the transcription (counting begins at 0)
- `seek` : Indicates where in the audio file this segment begins (3000 in this case)
- `start` and `end` timestamps: Tell us exactly when this segment occurs in the audio (43.92 to 50.16 seconds in our example)
- `avg_logprob` (Average Log Probability): -0.097569615 in our example indicates very high confidence. Values closer to 0 suggest better confidence, while more negative values (like -0.5 or lower) might indicate transcription issues.
- `no_speech_prob` (No Speech Probability): 0.0.012814695 is very low, suggesting this is definitely speech. Higher values (closer to 1) would indicate potential silence or non-speech audio.
- `compression_ratio` : 1.6637554 is a healthy value, indicating normal speech patterns. Unusual values (very high or low) might suggest issues with speech clarity or word boundaries.

Using Metadata for Debugging

When troubleshooting transcription issues, look for these patterns:

- Low Confidence Sections: If `avg_logprob` drops significantly (becomes more negative), check for background noise, multiple speakers talking simultaneously, unclear pronunciation, and strong accents. Consider cleaning up the audio in these sections or adjusting chunk sizes around problematic chunk boundaries.
- Non-Speech Detection: High `no_speech_prob` values might indicate silence periods that could be trimmed, background music or noise, or non-verbal sounds being misinterpreted as speech. Consider noise reduction when preprocessing.
- Unusual Speech Patterns: Unexpected `compression_ratio` values can reveal stuttering or word repetition, speaker talking unusually fast or slow, or audio quality issues affecting word separation.

Quality Thresholds and Regular Monitoring

We recommend setting acceptable ranges for each metadata value we reviewed above and flagging segments that fall outside these ranges to be able to identify and adjust preprocessing or chunking strategies for flagged sections.

By understanding and monitoring these metadata values, you can significantly improve your transcription quality and quickly identify potential issues in your audio processing pipeline.

Prompting Guidelines

The prompt parameter (max 224 tokens) helps provide context and maintain a consistent output style. Unlike chat completion prompts, these prompts only guide style and context, not specific actions.

Best Practices

- Provide relevant context about the audio content, such as the type of conversation, topic, or speakers involved.
- Use the same language as the language of the audio file.
- Steer the model's output by denoting proper spellings or emulate a specific writing style or tone.
- Keep the prompt concise and focused on stylistic guidance.

We can't wait to see what you build! 🚀