Package 'stt.api'

August 5, 2026

Title 'OpenAI' Compatible Speech-to-Text API Client

Version 0.3.1

Description A minimal-dependency R client for 'OpenAI'-compatible speech-to-text APIs (see <https://developers.openai.com/api/reference/resources/audio>) with optional local fallbacks. Supports 'OpenAI', local servers, and the 'whisper' package for local transcription.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/cornball-ai/stt.api

BugReports https://github.com/cornball-ai/stt.api/issues

Imports curl, jsonlite

Suggests tinytest, whisper

NeedsCompilation no

Author Troy Hernandez [aut, cre] (ORCID: <https://orcid.org/0009-0005-4248-604X>), cornball.ai [cph]

Maintainer Troy Hernandez <troy@cornball.ai>

Repository CRAN

Date/Publication 2026-08-04 22:40:02 UTC

Contents

clear_native_whisper_cache *Clear native whisper model cache*

Description

Removes cached native whisper models from memory. Call this to free GPU/RAM after batch processing is complete.

Usage

```
clear_native_whisper_cache()
```

Value

No return value, called for side effects (frees memory by removing cached models and triggers garbage collection).

Examples

```
clear_native_whisper_cache()
```

label_speakers *Fold Speaker Labels Into Caption Text*

Description

Diarized results carry speaker labels on `segments`, but subtitle tools read `data`, which is `from`/`to`/`text` only. The labels are therefore dropped on the way to a caption file. This folds them into the caption text so they survive:

Usage

```
label_speakers(x, sep = ": ", prefix = "", suffix = "")
```

Arguments

`x`	A result from `stt` with speaker labels, i.e. one produced with `response_format = "diarized_json"`.
`sep`	String placed between the label and the line. Defaults to `": "`.
`prefix`	String placed before the label, for styles like `prefix = "["`. Empty by default.
`suffix`	String placed after the label, for styles like `suffix = "]"`. Empty by default.

Details

```
HOUSTON: We copy you down, Eagle.
ARMSTRONG: Tranquility Base here. The Eagle has landed.
```

Only `data$text` changes. `segments` keeps its own untouched `text` and `speaker` columns, so the labels are still available separately, and the class and `"call_record"` attribute are preserved – the result still feeds `subtitles::whisper_to_srt()` and `whisper_to_ass()` directly.

Value

`x` with `data$text` relabelled. Segments whose speaker is missing are left alone rather than labelled `NA`, which happens when a provider matches only some speakers to the references given in `known_speakers`.

Karaoke

Pass `karaoke = FALSE` to `subtitles::whisper_to_ass()` for a diarized result. That argument defaults to `TRUE` and needs word-level timings, which OpenAI does not return alongside diarization – so the default errors on any diarized result, labelled or not. `whisper_to_srt()` is unaffected.

See Also

`stt` for `known_speakers`, which sets the labels this uses.

Examples

```
# The shape stt() returns for a diarized request.
x <- structure(
  list(
    segments = data.frame(
      start = c(0, 2.5), end = c(2.5, 5),
      text = c("We copy you down, Eagle.", "The Eagle has landed."),
      speaker = c("HOUSTON", "ARMSTRONG"),
      stringsAsFactors = FALSE),
    data = data.frame(
      from = c("00:00:00.000", "00:00:02.500"),
      to = c("00:00:02.500", "00:00:05.000"),
      text = c("We copy you down, Eagle.", "The Eagle has landed."),
      stringsAsFactors = FALSE)),
  class = c("stt_result", "whisper_transcription"))

label_speakers(x)$data$text
label_speakers(x, prefix = "[", suffix = "]", sep = " ")$data$text

## Not run:
# Into a caption file. karaoke = FALSE is required for diarized results.
subtitles::whisper_to_srt(label_speakers(x), "meeting.srt")
subtitles::whisper_to_ass(label_speakers(x), "meeting.ass",
                          karaoke = FALSE)

## End(Not run)
```

set_stt_base *Set the API Base URL*

Description

Sets the base URL for OpenAI-compatible STT endpoints.

Usage

```
set_stt_base(url)
```

Arguments

url	Character string. The base URL (e.g., "http://localhost:4123" or "https://api.openai.com").

Value

Invisibly returns the previous value.

Examples

```
set_stt_base("http://localhost:4123")
getOption("stt.api_base")
```

set_stt_key *Set the API Key*

Description

Sets the API key for hosted STT services (e.g., OpenAI). Local servers typically ignore this.

Usage

```
set_stt_key(key)
```

Arguments

key	Character string. The API key.

Value

Invisibly returns the previous value.

Examples

```
set_stt_key("test-key-123")
getOption("stt.api_key")
```

stt *Speech to Text*

Description

Convert an audio file to text using a local whisper backend or an OpenAI-compatible API.

Usage

```
stt(
  file,
  model = NULL,
  language = NULL,
  response_format = c("json", "text", "verbose_json", "diarized_json"),
  backend = c("auto", "whisper", "openai"),
  source = c("auto", "api", "package"),
  prompt = NULL,
  chunking_strategy = NULL,
  known_speakers = NULL
)
```

Arguments

`file`	Path to the audio file to convert.
`model`	Model name to use for transcription. For API backends, this is passed directly (e.g., "whisper-1"). For whisper, this is the model size (e.g., "tiny", "base", "small", "medium", "large"). If NULL, uses the backend's default. Which model you pick decides what timing you can get back from OpenAI: "whisper-1" is the only one that returns word-level timings (with `response_format = "verbose_json"`), "gpt-4o-transcribe-diarize" returns speaker-labelled segments (with `response_format = "diarized_json"`), and the plain "gpt-4o-transcribe"/"gpt-4o-mini-transcribe" models accept only `response_format = "json"`, so they return no timing at all and the result is a plain list with no `data` component. A self-hosted `whisper::serve()` endpoint has no such restriction.
`language`	Language code (e.g., "en", "es", "fr"). Optional hint to improve transcription accuracy.
`response_format`	Response format for API backend. One of "text", "json", "verbose_json", or "diarized_json". Ignored for whisper backend, except that a diarizing request is an error there, whether it is diarizing by format or by model (see `model`). "diarized_json" is OpenAI's diarizing format: segments gain a `speaker` column, and word timings are not available with it.
`backend`	Which engine to use: "auto" (default), "whisper", or "openai". Auto mode tries whisper first, then the openai API (if configured), except for a diarizing request, which only OpenAI serves and so resolves straight to "openai". That covers `response_format = "diarized_json"` and also a `model` whose name marks it

as diarizing, since those models answer plain "json" and "text" as well. See `source` for *where* the engine runs.

`source` Where the engine runs: "auto" (default), "api" for an HTTP service (OpenAI, or a self-hosted whisper server; see `set_stt_base`), or "package" for the in-process whisper R package. "auto" runs whisper in-process and openai via the API, matching the previous behavior. Use `backend = "whisper", source = "api"` to reach a whisper `serve()` endpoint.

`prompt` Optional text to guide the transcription. For API backend, this is passed as initial_prompt to help with spelling of names, acronyms, or domain-specific terms. Ignored for whisper backend, and an error for a diarizing model: OpenAI refuses a prompt for those whichever `response_format` is requested, so this is rejected on the model as well as on `diarized_json`.

`chunking_strategy` Optional chunking strategy passed to the API, e.g. "auto". Defaults to "auto" for any request to a diarizing model (`response_format = "diarized_json"`, or a `model` whose name says so), because OpenAI refuses those for audio longer than 30 seconds when it is unset. The threshold is the audio duration, not the response format. Defaulted regardless of length, since the duration is not known without decoding the file. NULL for non-diarizing requests.

`known_speakers` Optional named character vector of audio files, at most four, giving a short reference clip per speaker. The *names* are offered to the provider as labels: segments it matches to a reference come back named, so `known_speakers = c(agent = "agent.wav", caller = "caller.wav")` can yield `segments$speaker` values of "agent" and "caller". Matching is best-effort and partial results are normal – speakers it cannot match keep a generic label, so expect a mix, and check what came back rather than assuming. Nor does supplying references re-cut the segmentation: speakers the model has already merged into one cluster (several people on one radio downlink, say) are not thereby separated. Each clip should contain only that speaker and run roughly 2 to 10 seconds; the files are read and sent inline, so keep them short. Requires `response_format = "diarized_json"` and is an error otherwise.

Value

A list with components:

text The transcribed text as a single string.

segments A data.frame of segments with timing info, or NULL. For `response_format = "diarized_json"` it carries an extra `speaker` column; that column is absent, not NA, for every other format.

words A data.frame of word-level timestamps (word, start, end), present only when the API returns word granularity (verbose_json, and on OpenAI only from "whisper-1"; see `model`); otherwise absent.

language The detected or specified language code.

backend The legacy execution route ("api" or "whisper"). This reports *where* the engine ran, not the engine itself; the resolved `backend`/`source` pair lives in the `"call_record"` attribute.

raw The raw response from the backend.

When the result has usable segments (start/end/text columns), it additionally carries the shape subtitle tooling expects: a data data.frame with from/to timestamp strings ("HH:MM:SS.mmm") and text, and class c("stt_result", "whisper_transcription"), so it feeds subtitles::whisper_to_srt() and subtitles::whisper_to_ass() directly. Note the API route returns segments only with response_format = "verbose_json" or "diarized_json". Results without usable segments are plain lists, as before.

The result also carries a "call_record" attribute (cornball_sidecar v1, as in xtx.api/tts.api): the resolved request, elapsed seconds, and a timestamp – provenance that rides with the transcription when callers serialize it.

Examples

```
## Not run:
# Using OpenAI API
set_stt_base("https://api.openai.com")
set_stt_key(Sys.getenv("OPENAI_API_KEY"))
result <- stt("speech.wav", model = "whisper-1")
result$text

# Speaker-labelled segments
result <- stt("meeting.wav", model = "gpt-4o-transcribe-diarize",
              response_format = "diarized_json")
result$segments[, c("start", "end", "speaker", "text")]

# ...with your own labels instead of generic ones
audio <- system.file("audio", package = "stt.api")
result <- stt(file.path(audio, "EagleHasLanded.mp3"),
              model = "gpt-4o-transcribe-diarize",
              response_format = "diarized_json",
              known_speakers = c(
                  Armstrong = file.path(audio, "ref_armstrong.mp3"),
                  Houston   = file.path(audio, "ref_houston.mp3")))
unique(result$segments$speaker)

# Using a self-hosted whisper serve() endpoint
set_stt_base("http://troy-g5:7809")
result <- stt("speech.wav", backend = "whisper", source = "api")

# In-process whisper package
result <- stt("speech.wav", backend = "whisper", source = "package")

## End(Not run)
```

---

stt_health *Check STT Backend Health*

Description

Checks whether a transcription backend is available and working.

Usage

```
stt_health()
```

Value

A list with components:

ok Logical. TRUE if a backend is available.

backend Character. The available backend ("api" or "whisper"), or NULL if none available.

message Character. Status message with details.

Examples

```
## Not run:
h <- stt_health()
if (h$ok) {
  message("STT ready via ", h$backend)
}

## End(Not run)
```

Index