

Package ‘whisper’

August 3, 2026

Title Native R 'torch' Implementation of 'OpenAI' 'Whisper'

Version 0.5.1

Date 2026-07-31

Description Speech-to-text transcription using a native R 'torch' implementation of 'OpenAI' 'Whisper' model <<https://github.com/openai/whisper>>. Supports multiple model sizes from tiny (39M parameters) to large-v3 (1.5B parameters) with integrated download from 'HuggingFace' <<https://huggingface.co/>> via the 'hfhub' package. Provides automatic speech recognition with optional language detection and translation to English. Audio preprocessing, mel spectrogram computation, and transformer-based encoder-decoder inference are all implemented in R using the 'torch' package.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.5.0)

URL <https://github.com/cornball-ai/whisper>

BugReports <https://github.com/cornball-ai/whisper/issues>

Imports torch (>= 0.17.0), av, jsonlite, hfhub, safetensors, stats, utils

Suggests tinytest

NeedsCompilation no

Author Troy Hernandez [aut, cre] (ORCID: <<https://orcid.org/0009-0005-4248-604X>>), cornball.ai [cph], OpenAI [cph] (Whisper model architecture and mel filterbank data (MIT license))

Maintainer Troy Hernandez <troy@cornball.ai>

Repository CRAN

Date/Publication 2026-08-03 21:50:02 UTC

Contents

.model_sizes	4
apply_bpe	4
apply_timestamp_rules	5
audio_duration	5
audio_to_mel	6
beam_search_decode	7
build_byte_decoder	8
byte_to_token	8
clean_text	9
compression_ratio	9
compute_stft	10
compute_word_timestamps	10
copy_if_exists	11
create_decoder	11
create_encoder	12
create_mel_filterbank_fallback	12
decode_bpe_bytes	13
decode_timestamp	13
decode_with_fallback	14
detect_language	15
detect_language_from_mel	16
detect_language_from_pipeline	16
download_tokenizer_files	17
download_whisper_model	17
dtw_align	18
ensure_tokenizer_files	18
expand_kv_cache	19
extract_segments	19
forced_decode	20
get_initial_tokens	20
get_model_path	21
get_weights_path	21
greedy_decode	22
group_into_words	22
hz_to_mel	23
is_timestamp_token	23
list_downloaded_models	24
list_whisper_models	24
load_audio	25
load_decoder_weights	25
load_encoder_weights	26
load_mel_filterbank	26
load_whisper_model	27
load_whisper_weights	27
medfilt1	28
mel_to_hz	28

model_exists	29
pad_or_trim	29
parse_device	30
parse_dtype	30
print.whisper_resident	31
print.whisper_result	31
rearrange_kv_cache	32
resident_activate	32
resident_deactivate	33
resident_load	33
resident_status	34
resident_transcribe	35
resident_unload	36
sample_decode	36
serve	37
split_audio	38
tokenizer_decode	39
tokenizer_encode	39
transcribe	40
transcribe_chunk	41
transcribe_long	43
whisper_attention	44
whisper_config	44
whisper_decoder	45
whisper_decoder_layer	45
whisper_device	46
whisper_dtype	46
whisper_encoder	47
whisper_encoder_layer	47
whisper_language_table	48
whisper_lang_from_id	48
whisper_lang_token	49
whisper_model	49
whisper_pipeline	50
WHISPER_SAMPLE_RATE	50
whisper_special_tokens	51
whisper_tokenizer	51
whisper_tune_gc	52

.model_sizes	<i>Model Download Utilities</i>
--------------	---------------------------------

Description

Download Whisper models from HuggingFace using hfhub.

Usage

.model_sizes

Format

An object of class numeric of length 5.

apply_bpe	<i>Apply BPE Merges</i>
-----------	-------------------------

Description

Apply BPE Merges

Usage

apply_bpe(tokens, merge_ranks)

Arguments

- | | |
|-------------|--------------------------------|
| tokens | Character vector of tokens |
| merge_ranks | Named vector of merge rankings |

Value

Character vector after BPE merges

apply_timestamp_rules	<i>Apply Timestamp Token Rules</i>
-----------------------	------------------------------------

Description

Enforce Whisper timestamp generation constraints on logits.

Usage

apply_timestamp_rules(logits, generated, special, sample_begin)

Arguments

- logits Logit tensor (1, vocab) or (vocab)
- generated Integer vector of tokens generated so far
- special Special token IDs
- sample_begin Index where content tokens start in generated

Value

Modified logits tensor

audio_duration	<i>Get Audio Duration</i>
----------------	---------------------------

Description

Get Audio Duration

Usage

audio_duration(file)

Arguments

- file Path to audio file

Value

Duration in seconds

`audio_to_mel`*Convert Audio to Mel Spectrogram*

Description

Main preprocessing function that converts audio to the mel spectrogram format expected by Whisper.

Usage

```
audio_to_mel(file, n_mels = 80L, device = "auto", dtype = "auto")
```

Arguments

<code>file</code>	Path to audio file, or numeric vector of audio samples
<code>n_mels</code>	Number of mel bins (80 for most models, 128 for large-v3)
<code>device</code>	torch device for output tensor
<code>dtype</code>	torch dtype for output tensor

Value

torch tensor of shape (1, n_mels, 3000) for 30s audio

Examples

```
# Guarded like every other torch-touching example here. torch is an
# Import so it installs on a check machine, but its Lantern runtime is
# a separate post-install download that a build host will not have;
# unguarded, this errors under --run-donttest with "Lantern is not
# loaded".
if (torch::torch_is_installed()) {
  # Convert audio file to mel spectrogram
  audio_file <- system.file("audio", "jfk.mp3", package = "whisper")
  mel <- audio_to_mel(audio_file)
  dim(mel)
}
```

beam_search_decode	<i>Beam Search Decode</i>
--------------------	---------------------------

Description

Beam search decoding for Whisper. Maintains multiple hypotheses and selects the best one based on length-normalized log probability.

Usage

```
beam_search_decode(model, encoder_output, initial_tokens, tokenizer,
                    beam_size = 5L, max_length = 224L, timestamps = FALSE,
                    word_timestamps = FALSE, length_penalty = 1, patience = Inf,
                    device)
```

Arguments

model	WhisperModel
encoder_output	Encoder hidden states (batch=1)
initial_tokens	Initial token tensor (batch=1)
tokenizer	Tokenizer
beam_size	Number of beams
max_length	Maximum output length
timestamps	Whether to allow timestamp tokens
word_timestamps	Whether to collect cross-attention weights
length_penalty	Length penalty exponent
patience	Patience factor (stop after patience*beam_size finished)
device	Device

Value

List with tokens, cross_attn_weights, sum_logprob, n_tokens

build_byte_decoder	<i>Build Reverse Byte Decoder</i>
--------------------	-----------------------------------

Description

Inverts the GPT-2 byte-to-unicode mapping used by byte_to_token(). Cached after first call.

Usage

build_byte_decoder()

Value

Named character vector mapping unicode codepoint (as string) to raw byte value

byte_to_token	<i>Convert Byte to BPE Token</i>
---------------	----------------------------------

Description

GPT-2/Whisper uses a specific byte-to-unicode mapping.

Usage

byte_to_token(byte)

Arguments

byte Integer byte value (0-255)

Value

Character token

clean_text	<i>Clean Transcribed Text</i>
------------	-------------------------------

Description

Clean Transcribed Text

Usage

clean_text(text)

Arguments

text Raw decoded text

Value

Cleaned text

compression_ratio	<i>Compression Ratio</i>
-------------------	--------------------------

Description

Ratio of raw to compressed text size. High values indicate repetitive or hallucinated output.

Usage

compression_ratio(text)

Arguments

text Character string

Value

Numeric compression ratio

compute_stft	<i>Compute STFT Magnitude</i>
--------------	-------------------------------

Description

Compute STFT Magnitude

Usage

```
compute_stft(audio, n_fft = WHISPER_N_FFT, hop_length = WHISPER_HOP_LENGTH)
```

Arguments

audio	Numeric vector of audio samples
n_fft	FFT window size
hop_length	Hop length between frames

Value

Complex STFT matrix

compute_word_timestamps	<i>Compute Word-Level Timestamps</i>
-------------------------	--------------------------------------

Description

Use cross-attention weights and DTW alignment to assign timestamps to individual words.

Usage

```
compute_word_timestamps(tokens, cross_attn_weights, tokenizer, config,  
                        time_offset = 0, sample_begin = 4L)
```

Arguments

tokens	Integer vector of generated token IDs
cross_attn_weights	List of cross-attention weight tensors per decode step
tokenizer	Whisper tokenizer
config	Model configuration
time_offset	Time offset in seconds (for chunked audio)
sample_begin	Index where content tokens start in generated

Value

Data frame with word, start, end columns

copy_if_exists	<i>Copy Weight if Exists</i>
----------------	------------------------------

Description

Copy Weight if Exists

Usage

copy_if_exists(param, weights, name)

Arguments

param	Target parameter
weights	Weight dictionary
name	Weight name

create_decoder	<i>Create Decoder from Config</i>
----------------	-----------------------------------

Description

Create Decoder from Config

Usage

create_decoder(config)

Arguments

config	Model configuration from whisper_config()
--------	---

Value

WhisperDecoder module

create_encoder	Create Encoder from Config
Description	
Create Encoder from Config	
Usage	
create_encoder(config)	
Arguments	
config	Model configuration from whisper_config()
Value	
WhisperEncoder module	
create_mel_filterbank_fallback	Create Mel Filterbank (Fallback)

Description	
Create a mel filterbank matrix for converting STFT to mel spectrogram. Used when pre-computed filterbank is not available.	
Usage	
create_mel_filterbank_fallback(n_fft = WHISPER_N_FFT, n_mels = 80L, sample_rate = WHISPER_SAMPLE_RATE)	
Arguments	
n_fft	FFT size
n_mels	Number of mel bins
sample_rate	Audio sample rate
Value	
Mel filterbank matrix (n_mels x n_freqs)	

decode_bpe_bytes	<i>Decode BPE Bytes Back to Text</i>
------------------	--------------------------------------

Description

Reverses the GPT-2 byte-level encoding, converting unicode tokens back to raw UTF-8 bytes.

Usage

```
decode_bpe_bytes(text)
```

Arguments

text	Text with BPE byte tokens
------	---------------------------

Value

Decoded UTF-8 text

decode_timestamp	<i>Decode Timestamp Token</i>
------------------	-------------------------------

Description

Decode Timestamp Token

Usage

```
decode_timestamp(token_id, model = "tiny")
```

Arguments

token_id	Token ID
model	Model name for correct token IDs

Value

Time in seconds

decode_with_fallback *Decode with Temperature Fallback*

Description

Try decoding at progressively higher temperatures until quality thresholds are met. At temperature 0, uses beam search (or greedy if beam_size=1). At temperature > 0, uses sampling with best-of.

Usage

```
decode_with_fallback(model, encoder_output, initial_tokens, tokenizer,
                    temperatures = c(0, 0.2, 0.4, 0.6, 0.8, 1),
                    beam_size = 5L, best_of = 5L, max_length = 224L,
                    timestamps = FALSE, word_timestamps = FALSE,
                    compression_ratio_threshold = 2.4, logprob_threshold = -1,
                    no_speech_threshold = 0.6, length_penalty = 1,
                    patience = Inf, jit = TRUE, device)
```

Arguments

model	WhisperModel
encoder_output	Encoder hidden states
initial_tokens	Initial token tensor
tokenizer	Tokenizer
temperatures	Numeric vector of temperatures to try
beam_size	Number of beams for temp=0
best_of	Number of samples for temp>0
max_length	Maximum output length
timestamps	Whether to allow timestamp tokens
word_timestamps	Whether to collect cross-attention weights
compression_ratio_threshold	Max compression ratio
logprob_threshold	Min average log probability
no_speech_threshold	Skip a window as silence when its no-speech probability exceeds this and the decode is not confident.
length_penalty	Length penalty for beam search
patience	Patience factor for beam search
jit	Use the TorchScript greedy decode step on CUDA (default TRUE).
device	Device

Value

List with tokens, cross_attn_weights, sum_logprob, n_tokens

detect_language	<i>Detect Language</i>
-----------------	------------------------

Description

Identify the spoken language in an audio file. Uses Whisper's decoder to predict the most likely language token from the first 30 seconds of audio.

Usage

```
detect_language(file, model = "tiny", device = "auto", dtype = "auto",
               top_k = 5L, download = TRUE, verbose = TRUE)
```

Arguments

file	Path to audio file (WAV, MP3, etc.)
model	Model name: "tiny", "base", "small", "medium", "large-v3"
device	Device: "auto", "cpu", "cuda"
dtype	Data type: "auto", "float16", "float32"
top_k	Number of top language probabilities to return (default: 5)
download	If TRUE and model not present, prompt to download.
verbose	Print loading messages.

Value

List with language (two-letter code) and probabilities (named numeric vector of top-k language probs).

Examples

```
if (model_exists("tiny")) {
  audio_file <- system.file("audio", "jfk.mp3", package = "whisper")
  result <- detect_language(audio_file)
  result$language
  result$probabilities
}
```

detect_language_from_mel

Detect Language from Mel Spectrogram

Description

Core detection logic. Feed SOT token to decoder, read language logits.

Usage

```
detect_language_from_mel(model, mel, config, device, top_k = 5L)
```

Arguments

model	WhisperModel
mel	Mel spectrogram tensor
config	Model config
device	torch device
top_k	Number of top probabilities to return

Value

List with language code and probabilities

detect_language_from_pipeline

Detect Language from Pipeline

Description

Internal function that runs language detection using a pre-loaded pipeline.

Usage

```
detect_language_from_pipeline(pipe, file, top_k = 5L)
```

Arguments

pipe	A whisper_pipeline object
file	Path to audio file, or numeric vector of audio samples
top_k	Number of top probabilities to return

Value

List with language code and probabilities

download_tokenizer_files

Download Tokenizer Files from HuggingFace

Description

Download Tokenizer Files from HuggingFace

Usage

```
download_tokenizer_files(model)
```

Arguments

model	Model name
-------	------------

download_whisper_model

Download Model from HuggingFace

Description

Download Whisper model weights and tokenizer files from HuggingFace. In interactive sessions, asks for user consent before downloading.

Usage

```
download_whisper_model(model = "tiny", force = FALSE)
```

Arguments

model	Model name: "tiny", "base", "small", "medium", "large-v3"
force	Re-download even if exists

Value

Path to model directory (invisibly)

Examples

```
if (interactive()) {
  # Download tiny model (smallest, ~150MB)
  download_whisper_model("tiny")

  # Download larger model for better accuracy
  download_whisper_model("small")
}
```

dtw_align	<i>DTW Alignment</i>
-----------	----------------------

Description

Standard dynamic time warping on a cost matrix.

Usage

dtw_align(cost)

Arguments

cost Numeric matrix (n_tokens x n_frames)

Value

Integer matrix with 2 columns (token_idx, frame_idx), 1-indexed

ensure_tokenizer_files	<i>Ensure Tokenizer Files are Downloaded</i>
------------------------	--

Description

Ensure Tokenizer Files are Downloaded

Usage

ensure_tokenizer_files(model)

Arguments

model Model name

Value

Path to vocab directory (directory containing vocab.json)

expand_kv_cache	<i>Expand KV Cache for Beam Search</i>
-----------------	--

Description

Replicate batch=1 KV cache to batch=beam_size.

Usage

```
expand_kv_cache(kv_cache, beam_size)
```

Arguments

- kv_cache List of per-layer KV caches (batch=1)
- beam_size Number of beams

Value

Expanded KV cache (batch=beam_size)

extract_segments	<i>Extract Segments with Timestamps</i>
------------------	---

Description

Extract Segments with Timestamps

Usage

```
extract_segments(tokens, tokenizer, time_offset = 0)
```

Arguments

- tokens Token IDs
- tokenizer Tokenizer
- time_offset Offset in seconds for chunk processing

Value

Data frame with start, end, text

forced_decode	<i>Forced Decode</i>
---------------	----------------------

Description

Teacher-forcing decode: feed known token sequence one at a time, collecting cross-attention weights. Used by beam search when word_timestamps is needed.

Usage

```
forced_decode(model, encoder_output, token_ids, device)
```

Arguments

model	WhisperModel
encoder_output	Encoder hidden states
token_ids	Integer vector of all token IDs (including initial)
device	Device

Value

List of cross-attention weight lists (one per content step)

get_initial_tokens	<i>Get Initial Decoder Tokens</i>
--------------------	-----------------------------------

Description

Build the initial token sequence for decoder input.

Usage

```
get_initial_tokens(language = "en", task = "transcribe", model = "tiny",  
                  timestamps = FALSE)
```

Arguments

language	Two-letter language code or NULL for auto
task	"transcribe" or "translate"
model	Model name for correct special token IDs
timestamps	Whether to include timestamps (internal use)

Value

Integer vector of initial token IDs

get_model_path	<i>Get Model Cache Path</i>
----------------	-----------------------------

Description

Get Model Cache Path

Usage

```
get_model_path(model)
```

Arguments

model	Model name
-------	------------

Value

Path to model directory in hfhub cache

get_weights_path	<i>Get Path to Model Weights</i>
------------------	----------------------------------

Description

Get Path to Model Weights

Usage

```
get_weights_path(model)
```

Arguments

model	Model name
-------	------------

Value

Path to safetensors file

greedy_decode	<i>Greedy Decoding</i>
---------------	------------------------

Description

Greedy Decoding

Usage

```
greedy_decode(model, encoder_output, initial_tokens, tokenizer,
              max_length = 224L, timestamps = FALSE, word_timestamps = FALSE,
              device)
```

Arguments

model	WhisperModel
encoder_output	Encoder hidden states
initial_tokens	Initial token tensor
tokenizer	Tokenizer
max_length	Maximum output length
timestamps	Whether to allow timestamp tokens
word_timestamps	Whether to collect cross-attention weights
device	Device

Value

Integer vector of generated tokens, or list with tokens and cross_attn_weights when word_timestamps is TRUE

group_into_words	<i>Group Subword Tokens into Words</i>
------------------	--

Description

Merge BPE subword tokens into whole words with timestamps.

Usage

```
group_into_words(token_ids, starts, ends, tokenizer)
```

Arguments

token_ids	Integer vector of text token IDs
starts	Numeric vector of token start times
ends	Numeric vector of token end times
tokenizer	Whisper tokenizer

Value

Data frame with word, start, end columns

hz_to_mel	<i>Convert Hz to Mel Scale</i>
-----------	--------------------------------

Description

Convert Hz to Mel Scale

Usage

hz_to_mel(hz)

Arguments

hz	Frequency in Hz
----	-----------------

Value

Frequency in mel scale

is_timestamp_token	<i>Check if Token is Timestamp</i>
--------------------	------------------------------------

Description

Check if Token is Timestamp

Usage

is_timestamp_token(token_id, model = "tiny")

Arguments

token_id	Token ID
model	Model name for correct token IDs

Value

TRUE if timestamp token

`list_downloaded_models`*List Downloaded Models*

Description

List Downloaded Models

Usage

```
list_downloaded_models()
```

Value

Character vector of downloaded model names

Examples

```
list_downloaded_models()
```

`list_whisper_models` *List Available Models*

Description

List Available Models

Usage

```
list_whisper_models()
```

Value

Character vector of model names

Examples

```
list_whisper_models()
```

load_audio	<i>Load and Preprocess Audio</i>
------------	----------------------------------

Description

Load audio from file, convert to mono, resample to 16kHz.

Usage

load_audio(file)

Arguments

file Path to audio file (WAV, MP3, etc.)

Value

Numeric vector of audio samples normalized to -1 to 1 range

Examples

```
# Load included sample audio
audio_file <- system.file("audio", "jfk.mp3", package = "whisper")
samples <- load_audio(audio_file)
length(samples)
range(samples)
```

load_decoder_weights	<i>Load Decoder Weights</i>
----------------------	-----------------------------

Description

Load Decoder Weights

Usage

load_decoder_weights(decoder, weights)

Arguments

decoder WhisperDecoder module
weights Named list of tensors

load_encoder_weights	<i>Load Encoder Weights</i>
----------------------	-----------------------------

Description

Load Encoder Weights

Usage

```
load_encoder_weights(encoder, weights)
```

Arguments

encoder	WhisperEncoder module
weights	Named list of tensors

load_mel_filterbank	<i>Load Pre-computed Mel Filterbank</i>
---------------------	---

Description

Load the official Whisper mel filterbank from bundled CSV file.

Usage

```
load_mel_filterbank(n_mels = 80L)
```

Arguments

n_mels	Number of mel bins (80 or 128)
--------	--------------------------------

Value

Mel filterbank matrix (n_mels x n_freqs)

load_whisper_model	<i>Load Whisper Model</i>
--------------------	---------------------------

Description

Load a Whisper model with weights from HuggingFace.

Usage

```
load_whisper_model(model = "tiny", device = "auto", dtype = "auto",  
                   download = FALSE, verbose = TRUE)
```

Arguments

model	Model name: "tiny", "base", "small", "medium", "large-v3"
device	Device to load model on ("auto", "cpu", "cuda")
dtype	Data type ("auto", "float16", "float32")
download	If TRUE and model not present, prompt to download
verbose	Print loading messages

Value

WhisperModel module

Examples

```
# Load tiny model (requires prior download)  
if (model_exists("tiny")) {  
  model <- load_whisper_model("tiny")  
}
```

load_whisper_weights	<i>Load Weights from Safetensors</i>
----------------------	--------------------------------------

Description

Load Weights from Safetensors

Usage

```
load_whisper_weights(model, weights_path, verbose = TRUE)
```

Arguments

model	WhisperModel module
weights_path	Path to safetensors file
verbose	Print loading messages

medfilt1	<i>1D Median Filter</i>
----------	-------------------------

Description

Apply a sliding median filter to a numeric vector.

Usage

```
medfilt1(x, width = 7L)
```

Arguments

x	Numeric vector
width	Filter width (must be odd)

Value

Filtered numeric vector of same length

mel_to_hz	<i>Convert Mel Scale to Hz</i>
-----------	--------------------------------

Description

Convert Mel Scale to Hz

Usage

```
mel_to_hz(mel)
```

Arguments

mel	Frequency in mel scale
-----	------------------------

Value

Frequency in Hz

model_exists	<i>Check if Model is Downloaded</i>
--------------	-------------------------------------

Description

Check if Model is Downloaded

Usage

```
model_exists(model)
```

Arguments

model	Model name
-------	------------

Value

TRUE if model weights exist locally

Examples

```
model_exists("tiny")
model_exists("large-v3")
```

pad_or_trim	<i>Pad or Trim Audio to Fixed Length</i>
-------------	--

Description

Pad or Trim Audio to Fixed Length

Usage

```
pad_or_trim(audio, length = WHISPER_N_SAMPLES)
```

Arguments

audio	Numeric vector of audio samples
length	Target length in samples (default: 30s at 16kHz)

Value

Numeric vector of specified length

parse_device	<i>Parse Device Argument</i>
--------------	------------------------------

Description

Parse Device Argument

Usage

```
parse_device(device = "auto")
```

Arguments

device Character or torch device. "auto" uses GPU if available.

Value

torch device object

parse_dtype	<i>Parse Dtype Argument</i>
-------------	-----------------------------

Description

Parse Dtype Argument

Usage

```
parse_dtype(dtype = "auto", device = whisper_device())
```

Arguments

dtype Character or torch dtype. "auto" uses float16 on GPU, float32 on CPU.
device torch device (used for auto selection)

Value

torch dtype

```
print.whisper_resident
```

Print a Resident Handle

Description

Print a Resident Handle

Usage

```
## S3 method for class 'whisper_resident'  
print(x, ...)
```

Arguments

x	A 'whisper_resident' handle.
...	Ignored.

Value

'x', invisibly.

```
print.whisper_result
```

Print a Transcription Result

Description

Print a Transcription Result

Usage

```
## S3 method for class 'whisper_result'  
print(x, ...)
```

Arguments

x	A 'transcribe()' result.
...	Ignored.

Value

'x', invisibly.

rearrange_kv_cache	<i>Rearrange KV Cache by Beam Indices</i>
--------------------	---

Description

Reorder cached key-value tensors to match new beam ordering.

Usage

```
rearrange_kv_cache(kv_cache, beam_indices, device)
```

Arguments

kv_cache	List of per-layer KV caches
beam_indices	Integer tensor of beam indices (1-indexed)
device	Device

Value

Reordered KV cache

resident_activate	<i>Activate a Resident Model (Create the GPU Representation)</i>
-------------------	--

Description

Copies the pinned host weights to the target device in one synchronous pass (non-blocking per-tensor copies, then a device synchronize). The transition is transactional: on any failure – including a partial move after an out-of-memory error – every tensor is rebound to its pinned host copy, the CUDA allocation is released, and the handle returns to "inactive". A rollback that cannot be verified leaves the handle "broken", where only [resident_status()] and [resident_unload()] operate.

Usage

```
resident_activate(res)
```

Arguments

res	A ‘whisper_resident’ handle from [resident_load()].
-----	---

Value

The handle, invisibly. No-op when already active.

resident_deactivate	<i>Deactivate a Resident Model (Destroy the GPU Representation)</i>
---------------------	---

Description

Rebinds every tensor to its pinned host copy, releases the orphaned CUDA storage, and verifies the restored state tensor-by-tensor. The pinned weights remain in host memory, so a later [resident_activate()] is a DMA copy, not a disk reload. Refuses while a transcription is in flight.

Usage

```
resident_deactivate(res, release = TRUE)
```

Arguments

res	A ‘whisper_resident’ handle.
release	Return the allocator’s blocks to the driver (default TRUE). FALSE keeps them pooled for a fast next activation; the weights are freed either way, and ‘gpu_bytes’ goes to zero in both cases (the retained pool is process overhead, attributable to no model).

Details

‘release’ decides who gets the freed VRAM, and it is worth about an order of magnitude. With ‘release = TRUE’ (the default) the CUDA caching allocator hands its blocks back to the driver, so the memory is visible as free to other processes – but the next activation must re-acquire every block from the driver, which on a small card measured ~9 ms per tensor (medium fp32, 2.85 GB across 948 tensors: 9.2 s, 0.31 GB/s). With ‘release = FALSE’ the blocks stay in this process’s pool for the next model to reuse, and the same activation took 0.86 s (3.29 GB/s), matching raw pinned-DMA bandwidth on that card. Use ‘FALSE’ when one process hosts every model and switches between them; use the default when a different process needs the card.

Value

The handle, invisibly. No-op when already inactive.

resident_load	<i>Load a Model as a Resident (Pinned Host Weights)</i>
---------------	---

Description

Loads a Whisper model and retains its canonical weights as page-locked (pinned) CPU tensors, so the GPU representation can be created and destroyed repeatedly without reloading from disk. [resident_activate()] copies the weights to the GPU (a fast DMA transfer from pinned memory); [resident_deactivate()] frees the GPU copy and repoints the model at the pinned host storage. The handle starts inactive.

Usage

```
resident_load(model = "tiny", device = "cuda", dtype = "auto", download = TRUE,
              verbose = TRUE)
```

Arguments

model	Model name: "tiny", "base", "small", "medium", "large-v3"
device	Target CUDA device for activation (default "cuda"). Residency requires CUDA; pinned host memory exists to feed it. A bare "cuda" is resolved to the current device's explicit index at load time, and the handle stays bound to that exact device ("cuda:N") for every later transition and synchronize.
dtype	"auto" (default), "float16", or "float32"; resolved against 'device'.
download	Download the model if not cached (default TRUE).
verbose	Print progress messages.

Details

The dtype is resolved against the **target** device with the same rules as [whisper_pipeline()]: float16 on CUDA, except GPUs with broken fp16 (GTX 16-series), which get float32. The pinned copies are stored at the resolved dtype, so activation moves exactly the bytes inference needs.

Value

A 'whisper_resident' handle (an environment). Fields of interest via [resident_status()]: state, byte counts, and a content identity (weights sha256, HF repo and snapshot revision, resolved dtype).

Examples

```
if (torch::torch_is_installed() && torch::cuda_is_available() &&
    model_exists("tiny")) {
  res <- resident_load("tiny")
  resident_activate(res)
  audio <- system.file("audio", "jfk.mp3", package = "whisper")
  resident_transcribe(res, audio, timestamps = TRUE)
  resident_deactivate(res) # VRAM freed, weights stay pinned in RAM
  resident_activate(res) # fast: DMA copy, no disk
  resident_unload(res)
}
```

resident_status	<i>Status of a Resident Model</i>
-----------------	-----------------------------------

Description

Callable in every state, including "broken" and "unloaded".

Usage

```
resident_status(res)
```

Arguments

`res` A ‘whisper_resident’ handle.

Value

A list: model, state, in_flight, device (the exact bound device, e.g. "cuda:0"), dtype, pinned_bytes, gpu_bytes (logical sums over the tensor manifest, not allocator statistics; in the "broken" state this is the observed on-GPU manifest bytes, or NA when unknowable), identity (weights sha256, HF repo/revision, resolved dtype), path (diagnostic), last_error.

resident_transcribe	<i>Transcribe with a Resident Model</i>
---------------------	---

Description

Runs [transcribe()] through the resident handle’s pipeline. Requires the model to be active; marks the handle in-flight for the duration so a concurrent deactivation is refused.

Usage

```
resident_transcribe(res, file, ...)
```

Arguments

`res` A ‘whisper_resident’ handle, currently active.

`file` Path to audio file (WAV, MP3, MP4, ...).

`...` Passed to the pipeline transcriber: ‘language’, ‘task’, ‘timestamps’, ‘word_timestamps’, ‘beam_size’, ‘temperatures’, ... (see [transcribe()]).

Value

The transcription result, same shape as [transcribe()].

resident_unload	<i>Unload a Resident Model (Drop Pinned and GPU State)</i>
-----------------	--

Description

Releases both representations: any GPU tensors and the pinned host copies. Permitted from every state except mid-flight – including "broken", where it is the recovery path (dropping the module and the pinned list releases the storage that could not be verified). The handle becomes "unloaded", which is terminal.

Usage

```
resident_unload(res)
```

Arguments

res	A 'whisper_resident' handle.
-----	------------------------------

Value

The handle, invisibly.

sample_decode	<i>Sample Decode</i>
---------------	----------------------

Description

Temperature-scaled sampling decode. Fork of greedy_decode that uses categorical sampling instead of argmax.

Usage

```
sample_decode(model, encoder_output, initial_tokens, tokenizer,
              temperature = 0.6, max_length = 224L, timestamps = FALSE,
              word_timestamps = FALSE, device)
```

Arguments

model	WhisperModel
encoder_output	Encoder hidden states
initial_tokens	Initial token tensor (batch=1)
tokenizer	Tokenizer
temperature	Sampling temperature (must be > 0)
max_length	Maximum output length

timestamps	Whether to allow timestamp tokens
word_timestamps	Whether to collect cross-attention weights
device	Device

Value

List with tokens, cross_attn_weights, sum_logprob, n_tokens

serve	<i>Serve whisper over HTTP</i>
-------	--------------------------------

Description

Starts a blocking HTTP server that loads a whisper model once and answers OpenAI-compatible speech-to-text requests. Intended as a drop-in for the OpenAI transcription API or a Whisper container: point an HTTP client (e.g. **stt.api** via `set_stt_base()`) at `http://<host>:<port>` and it serves the same endpoint.

Usage

```
serve(port = 7809L, model = "large-v3", device = "cuda", dtype = "auto",
      timeout = 300L, max_body = 100L * 1024L^2, warmup = TRUE)
```

Arguments

port	Integer. TCP port to listen on. Default 7809 (the cornball serve range is 7809-7829; chatterbox sits on 7810).
model	Model name to load and keep resident (the request's model field is ignored). Default "large-v3".
device	Character. Torch device ("cuda", "cpu", "mps").
dtype	Compute dtype ("auto", "float16", "float32").
timeout	Integer. Per-connection I/O timeout in seconds (guards against stalled clients). Default 300.
max_body	Integer. Maximum request body size in bytes. Default 100 MB (audio uploads are larger than JSON bodies).
warmup	Logical. Transcribe a short bundled clip at startup to compile the decode step and prime the allocator, so the first client request isn't slow. Default TRUE.

Details

Endpoints:

- GET /health - liveness probe, returns {"status": "ok", "model": ...}.
- POST /v1/audio/transcriptions - multipart/form-data with fields file (required audio upload), language, response_format (json (default), text, or verbose_json), temperature, and timestamp_granularities[]. Returns the transcription. verbose_json adds segments with start/end times; adding timestamp_granularities[]=word also returns words with per-word start/end times.
- POST /v1/audio/translations - same, but translates to English (Whisper’s translate task).

The server is single-threaded and runs until interrupted. Run it under a process supervisor (systemd, a container CMD, tmux) for persistence; an example systemd unit ships with the package: `system.file("whisper.service", package = "whisper")`. It is designed to sit alongside a chatbox TTS server as a second always-on process on the same GPU (each has its own CUDA context). On CUDA it tunes torch’s allocator GC (see [whisper_tune_gc](#)) before loading and uses the TorchScript greedy decode step.

Value

Does not return normally; runs until interrupted.

split_audio	<i>Split Long Audio into Chunks</i>
-------------	-------------------------------------

Description

Split audio longer than 30 seconds into overlapping chunks.

Usage

```
split_audio(file, chunk_length = 30, overlap = 1)
```

Arguments

file	Path to audio file
chunk_length	Chunk length in seconds
overlap	Overlap between chunks in seconds

Value

List of audio chunks (numeric vectors)

tokenizer_decode	<i>Decode Token IDs to Text</i>
------------------	---------------------------------

Description

Decode Token IDs to Text

Usage

```
tokenizer_decode(ids, id_to_token, special_tokens)
```

Arguments

ids	Integer vector of token IDs
id_to_token	Mapping from ID to token
special_tokens	Special token info

Value

Character string

tokenizer_encode	<i>Encode Text to Token IDs</i>
------------------	---------------------------------

Description

Encode Text to Token IDs

Usage

```
tokenizer_encode(text, vocab, merge_ranks, eot_fallback = NULL)
```

Arguments

text	Character string to encode
vocab	Vocabulary mapping (token -> id)
merge_ranks	Merge ranking for BPE
eot_fallback	End-of-text id for any token not found in vocab. Like the Python reference (tiktoken), < endoftext > is a special token kept out of the BPE vocab; some vocab.json files (large-v3) omit it entirely, so the id is supplied from the special-token table.

Value

Integer vector of token IDs

transcribe

*Transcribe Audio***Description**

Transcribe speech from an audio file using Whisper.

Usage

```
transcribe(file, model = "tiny", language = NULL, task = "transcribe",
           timestamps = FALSE, word_timestamps = FALSE, beam_size = 1L,
           temperatures = c(0, 0.2, 0.4, 0.6, 0.8, 1), best_of = 1L,
           compression_ratio_threshold = 2.4, logprob_threshold = -1,
           length_penalty = 1, patience = Inf, jit = TRUE, device = "auto",
           dtype = "auto", verbose = TRUE)
```

Arguments

file	Path to audio file (WAV, MP3, etc.)
model	Model name: "tiny", "base", "small", "medium", "large-v3"
language	Language code (e.g., "en", "es"), or NULL (default) for auto-detection from the audio.
task	"transcribe" or "translate" (translate to English)
timestamps	If TRUE, return segment-level timestamps
word_timestamps	If TRUE, return word-level timestamps (implies timestamps)
beam_size	Number of beams for beam search (1 = greedy, default)
temperatures	Numeric vector of temperatures to try. 0 uses beam search or greedy; values > 0 use sampling. Multiple values enable fallback.
best_of	Number of samples per temperature > 0, keeping the best.
compression_ratio_threshold	Max compression ratio before fallback.
logprob_threshold	Min average log probability before fallback.
length_penalty	Length penalty exponent for beam search scoring.
patience	Patience factor for beam search (stop after patience*beam_size).
jit	On CUDA, run decoding through a TorchScript decode step (default TRUE), covering both greedy and word-timestamp runs. Token-for-token equivalent to the eager path but avoids the per-op R dispatch floor. No effect on CPU or beam search, which use the eager decoder.
device	Device: "auto", "cpu", "cuda"
dtype	Data type: "auto", "float16", "float32"
verbose	Print progress messages

Details

For repeated transcription, use `whisper_pipeline()` to load the model once.

Value

List with text, language, and metadata. When `timestamps=TRUE`, includes segments data.frame with start, end, text columns. When `word_timestamps=TRUE`, includes words data.frame with word, start, end columns.

Examples

```
if (model_exists("tiny")) {
  audio_file <- system.file("audio", "jfk.mp3", package = "whisper")

  # Auto-detect language (default)
  result <- transcribe(audio_file, model = "tiny")
  result$language # "en"
  result$text

  # Explicit language
  result <- transcribe(audio_file, model = "tiny", language = "en")

  # With timestamps
  result <- transcribe(audio_file, model = "tiny", timestamps = TRUE)
  result$segments

  # Translate Spanish audio to English
  spanish_file <- system.file("audio", "allende.mp3", package = "whisper")
  result <- transcribe(spanish_file, model = "tiny",
                       language = "es", task = "translate")
  result$text
}
```

transcribe_chunk	<i>Transcribe Single Chunk</i>
------------------	--------------------------------

Description

Transcribe Single Chunk

Usage

```
transcribe_chunk(file, model, tokenizer, config, language = NULL,
                 task = "transcribe", timestamps = FALSE,
                 word_timestamps = FALSE, beam_size = 1L,
                 temperatures = c(0, 0.2, 0.4, 0.6, 0.8, 1), best_of = 1L,
                 compression_ratio_threshold = 2.4, logprob_threshold = -1,
                 no_speech_threshold = 0.6, length_penalty = 1, patience = Inf,
                 jit = TRUE, time_offset = 0, device, dtype, verbose = TRUE)
```

Arguments

file	Audio file or mel spectrogram
model	WhisperModel
tokenizer	Tokenizer
config	Model config
language	Language code
task	Task type
timestamps	Return segment-level timestamps.
word_timestamps	Return word-level timestamps.
beam_size	Number of beams for beam search.
temperatures	Numeric vector of temperatures for fallback.
best_of	Number of samples per temperature > 0.
compression_ratio_threshold	Max compression ratio before fallback.
logprob_threshold	Min average log probability before fallback.
no_speech_threshold	Skip a window as silence when its no-speech probability exceeds this and the decode is not confident.
length_penalty	Length penalty exponent for beam search.
patience	Patience factor for beam search.
jit	Use the TorchScript greedy decode step on CUDA.
time_offset	Time offset in seconds for chunk processing.
device	Device
dtype	Dtype
verbose	Verbose output

Value

Transcription result

transcribe_long	<i>Transcribe Long Audio</i>
-----------------	------------------------------

Description

Process audio longer than 30 seconds in chunks.

Usage

```
transcribe_long(file, model, tokenizer, config, language, task,
                timestamps = FALSE, word_timestamps = FALSE, beam_size = 1L,
                temperatures = c(0, 0.2, 0.4, 0.6, 0.8, 1), best_of = 1L,
                compression_ratio_threshold = 2.4, logprob_threshold = -1,
                length_penalty = 1, patience = Inf, jit = TRUE, device, dtype,
                verbose)
```

Arguments

file	Audio file
model	WhisperModel
tokenizer	Tokenizer
config	Model config
language	Language
task	Task
timestamps	Return segment-level timestamps.
word_timestamps	Return word-level timestamps.
beam_size	Number of beams for beam search.
temperatures	Numeric vector of temperatures for fallback.
best_of	Number of samples per temperature > 0.
compression_ratio_threshold	Max compression ratio before fallback.
logprob_threshold	Min average log probability before fallback.
length_penalty	Length penalty exponent for beam search.
patience	Patience factor for beam search.
jit	Use the TorchScript greedy decode step on CUDA.
device	Device
dtype	Dtype
verbose	Verbose

Value

Combined transcription result

whisper_attention	<i>Multi-Head Self-Attention</i>
-------------------	----------------------------------

Description

Multi-Head Self-Attention

Usage

```
whisper_attention(n_state, n_head)
```

Arguments

n_state	Hidden dimension
n_head	Number of attention heads

whisper_config	<i>Whisper Model Configurations</i>
----------------	-------------------------------------

Description

Get configuration for a Whisper model variant.

Usage

```
whisper_config(model = "tiny")
```

Arguments

model	Character. Model name: "tiny", "base", "small", "medium", "large-v3"
-------	--

Value

List with model configuration parameters

Examples

```
# Get tiny model configuration
cfg <- whisper_config("tiny")
cfg$n_mels
cfg$n_audio_layer

# Compare model sizes
whisper_config("tiny")$n_text_layer
whisper_config("large-v3")$n_text_layer
```

whisper_decoder	<i>Text Decoder</i>
-----------------	---------------------

Description

Full Whisper decoder: token embedding + positional embedding + transformer layers.

Usage

```
whisper_decoder(n_vocab, n_ctx, n_state, n_head, n_layer)
```

Arguments

n_vocab	Vocabulary size
n_ctx	Maximum context length
n_state	Hidden dimension
n_head	Number of attention heads
n_layer	Number of transformer layers

whisper_decoder_layer	<i>Decoder Layer</i>
-----------------------	----------------------

Description

Pre-norm transformer decoder layer with self-attention and cross-attention.

Usage

```
whisper_decoder_layer(n_state, n_head)
```

Arguments

n_state	Hidden dimension
n_head	Number of attention heads

whisper_device	<i>Get Default Device</i>
----------------	---------------------------

Description

Returns CUDA device if available, otherwise CPU.

Usage

```
whisper_device()
```

Value

torch device object

Examples

```
if (torch::torch_is_installed()) {  
  device <- whisper_device()  
  device$type  
}
```

whisper_dtype	<i>Get Default Dtype</i>
---------------	--------------------------

Description

Returns float16 on CUDA, float32 on CPU. Exception: the GTX 16-series (TU116/TU117, e.g. GTX 1660/1650) computes float16 incorrectly and produces NaN output, so float32 is used on those cards. Pass an explicit dtype = "float16" to override.

Usage

```
whisper_dtype(device = whisper_device())
```

Arguments

device	torch device
--------	--------------

Value

torch dtype

Examples

```
if (torch::torch_is_installed()) {  
  dtype <- whisper_dtype()  
  dtype  
}
```

whisper_encoder	<i>Audio Encoder</i>
-----------------	----------------------

Description

Full Whisper encoder: Conv stem + positional encoding + transformer layers.

Usage

```
whisper_encoder(n_mels, n_ctx, n_state, n_head, n_layer)
```

Arguments

n_mels	Number of mel spectrogram bins
n_ctx	Maximum context length (1500 for 30s audio)
n_state	Hidden dimension
n_head	Number of attention heads
n_layer	Number of transformer layers

whisper_encoder_layer	<i>Encoder Layer</i>
-----------------------	----------------------

Description

Pre-norm transformer encoder layer.

Usage

```
whisper_encoder_layer(n_state, n_head)
```

Arguments

n_state	Hidden dimension
n_head	Number of attention heads

whisper_language_table
<i>Whisper Language Table</i>

Description

Returns the named integer vector mapping language codes to offsets.

Usage

whisper_language_table()

Value

Named integer vector (language code -> offset from 50259)

whisper_lang_from_id	<i>Get Language Code from Token ID</i>
----------------------	--

Description

Reverse lookup: convert a language token ID back to a two-letter code.

Usage

whisper_lang_from_id(token_id)

Arguments

token_id Integer token ID (e.g., 50259 for English)

Value

Two-letter language code

whisper_lang_token	<i>Get Language Token ID</i>
--------------------	------------------------------

Description

Get Language Token ID

Usage

```
whisper_lang_token(lang = "en", model = "tiny")
```

Arguments

- lang Two-letter language code (e.g., "en", "es", "fr")
- model Model name for correct token IDs

Value

Token ID for the language

whisper_model	<i>Whisper Model Module</i>
---------------	-----------------------------

Description

Whisper Model Module

Usage

```
whisper_model(config)
```

Arguments

- config Model configuration

whisper_pipeline	Create a Whisper Pipeline
------------------	---------------------------

Description

Load the model, tokenizer, and config once. Call \$transcribe() repeatedly without reloading.

Usage

```
whisper_pipeline(model = "tiny", device = "auto", dtype = "auto",
                  download = TRUE, verbose = TRUE)
```

Arguments

model	Model name: "tiny", "base", "small", "medium", "large-v3"
device	Device: "auto", "cpu", "cuda"
dtype	Data type: "auto", "float16", "float32"
download	If TRUE and model not present, prompt to download.
verbose	Print loading messages.

Value

A whisper_pipeline object with a \$transcribe() method.

Examples

```
if (model_exists("tiny")) {
  pipe <- whisper_pipeline("tiny")
  pipe$transcribe(system.file("audio", "jfk.mp3", package = "whisper"))
}
```

WHISPER_SAMPLE_RATE	Whisper Audio Constants
---------------------	-------------------------

Description

Whisper Audio Constants

Usage

```
WHISPER_SAMPLE_RATE
```

Format

An object of class integer of length 1.

`whisper_special_tokens`*Special Token IDs*

Description

Get special token IDs for a Whisper model. Token IDs differ between model variants (e.g., large-v3 has extra language tokens).

Usage

```
whisper_special_tokens(model = "tiny")
```

Arguments

<code>model</code>	Model name (default: "tiny")
--------------------	------------------------------

Value

Named list of special token IDs

`whisper_tokenizer`*Create Whisper Tokenizer*

Description

Load or create a Whisper tokenizer from HuggingFace vocab files.

Usage

```
whisper_tokenizer(model = "tiny")
```

Arguments

<code>model</code>	Model name for vocab lookup
--------------------	-----------------------------

Value

Tokenizer object (list with encode/decode functions)

Examples

```
# Load tokenizer (requires prior model download)
if (model_exists("tiny")) {
  tok <- whisper_tokenizer("tiny")
  tok$encode("Hello world")
  tok$decode(c(50258, 50259, 50359, 50363))
}
```

whisper_tune_gc

Tune torch's CUDA garbage collection for whisper inference

Description

Opt-in performance helper. torch's CUDA allocator invokes R's `gc()` on nearly every allocation once a loaded model occupies more than 20% GPU memory (its default `torch.cuda_allocator_reserved_rate` floor), which can dominate inference time for the larger whisper models. This raises the floor to the model's footprint as a fraction of VRAM (clamped to at least 0.2, so models already under the default are unaffected) and lifts `torch.threshold_call_gc` off its 4 GB default.

Usage

```
whisper_tune_gc(model = "large-v3", device = "auto", dtype = "auto",
                footprint_gb = NULL)
```

Arguments

<code>model</code>	Whisper model name, used to estimate the footprint when <code>footprint_gb</code> is <code>NULL</code> .
<code>device</code>	Device, as accepted by load_whisper_model .
<code>dtype</code>	Compute dtype; determines bytes per parameter.
<code>footprint_gb</code>	Optional explicit footprint in GB, overriding the per-model estimate (use for combined multi-model workloads).

Details

Call this *before* [load_whisper_model](#): torch reads the allocator rates once, at lazy CUDA initialization. It is a no-op on non-CUDA devices and only sets an option that is not already set, so an explicit `options(torch.cuda_allocator_reserved_rate = ...)` always wins.

Side effect: it sets session-global `torch.*` options that persist after the call - deliberately, since torch reads them later. The package never calls this for you; you invoke it.

Call it before anything initializes CUDA. torch reads the allocator rates exactly once, at CUDA init, so options set afterwards have no effect for the rest of the session. This function is careful not to initialize CUDA itself - it resolves the device and the dtype size from strings and `nvidia-smi` rather than from torch - but it cannot undo an init that already happened. In a process that hosts several models, call it first, before any of them load.

For several models resident on one GPU in the *same* R process, pass their combined size via `footprint_gb` so the single shared floor covers all of them.

Value

The reserved-rate that was set (invisibly), or `NULL` when nothing was set (non-CUDA device, or the option was already set).

Examples

```
if (torch::torch_is_installed()) {  
  # No-op off CUDA; returns NULL.  
  whisper_tune_gc("large-v3", device = "cpu")  
}  
  
## Not run:  
# On a GPU, call before loading so torch picks up the rate at CUDA init:  
whisper_tune_gc("large-v3", device = "cuda")  
model <- load_whisper_model("large-v3", device = "cuda")  
  
## End(Not run)
```

Index

- * **datasets**
 - .model_sizes, [4](#)
 - WHISPER_SAMPLE_RATE, [50](#)
- .model_sizes, [4](#)
- apply_bpe, [4](#)
- apply_timestamp_rules, [5](#)
- audio_duration, [5](#)
- audio_to_mel, [6](#)
- beam_search_decode, [7](#)
- build_byte_decoder, [8](#)
- byte_to_token, [8](#)
- clean_text, [9](#)
- compression_ratio, [9](#)
- compute_stft, [10](#)
- compute_word_timestamps, [10](#)
- copy_if_exists, [11](#)
- create_decoder, [11](#)
- create_encoder, [12](#)
- create_mel_filterbank_fallback, [12](#)
- decode_bpe_bytes, [13](#)
- decode_timestamp, [13](#)
- decode_with_fallback, [14](#)
- detect_language, [15](#)
- detect_language_from_mel, [16](#)
- detect_language_from_pipeline, [16](#)
- download_tokenizer_files, [17](#)
- download_whisper_model, [17](#)
- dtw_align, [18](#)
- ensure_tokenizer_files, [18](#)
- expand_kv_cache, [19](#)
- extract_segments, [19](#)
- forced_decode, [20](#)
- get_initial_tokens, [20](#)
- get_model_path, [21](#)
- get_weights_path, [21](#)
- greedy_decode, [22](#)
- group_into_words, [22](#)
- hz_to_mel, [23](#)
- is_timestamp_token, [23](#)
- list_downloaded_models, [24](#)
- list_whisper_models, [24](#)
- load_audio, [25](#)
- load_decoder_weights, [25](#)
- load_encoder_weights, [26](#)
- load_mel_filterbank, [26](#)
- load_whisper_model, [27](#), [52](#)
- load_whisper_weights, [27](#)
- medfilt1, [28](#)
- mel_to_hz, [28](#)
- model_exists, [29](#)
- pad_or_trim, [29](#)
- parse_device, [30](#)
- parse_dtype, [30](#)
- print.whisper_resident, [31](#)
- print.whisper_result, [31](#)
- rearrange_kv_cache, [32](#)
- resident_activate, [32](#)
- resident_deactivate, [33](#)
- resident_load, [33](#)
- resident_status, [34](#)
- resident_transcribe, [35](#)
- resident_unload, [36](#)
- sample_decode, [36](#)
- serve, [37](#)
- split_audio, [38](#)
- tokenizer_decode, [39](#)
- tokenizer_encode, [39](#)

transcribe, [40](#)
transcribe_chunk, [41](#)
transcribe_long, [43](#)

whisper_attention, [44](#)
whisper_config, [44](#)
whisper_decoder, [45](#)
whisper_decoder_layer, [45](#)
whisper_device, [46](#)
whisper_dtype, [46](#)
whisper_encoder, [47](#)
whisper_encoder_layer, [47](#)
whisper_lang_from_id, [48](#)
whisper_lang_token, [49](#)
whisper_language_table, [48](#)
whisper_model, [49](#)
whisper_pipeline, [41](#), [50](#)
WHISPER_SAMPLE_RATE, [50](#)
whisper_special_tokens, [51](#)
whisper_tokenizer, [51](#)
whisper_tune_gc, [38](#), [52](#)