

# Package: whisper (via r-universe)

May 18, 2026

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

**Version** 0.3.0

**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

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

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

**Imports** torch, av, jsonlite, hfhub, safetensors, stats, utils

**Suggests** tinytest

**Config/pak/sysreqs** cmake make libavfilter-dev libuv1-dev libssl-dev

**Repository** <https://cornball-ai.r-universe.dev>

**Date/Publication** 2026-03-19 15:49:53 UTC

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

**RemoteRef** HEAD

**RemoteSha** 1a34db6b99ba8e7570334ae2e7dd1dcfa2f7115c

## Contents

apply_bpe . . . . .	3
apply_timestamp_rules . . . . .	4
audio_duration . . . . .	4

audio_to_mel . . . . .	5
beam_search_decode . . . . .	5
build_byte_decoder . . . . .	6
byte_to_token . . . . .	7
clean_text . . . . .	7
compression_ratio . . . . .	8
compute_stft . . . . .	8
compute_word_timestamps . . . . .	9
copy_if_exists . . . . .	9
create_decoder . . . . .	10
create_encoder . . . . .	10
create_mel_filterbank_fallback . . . . .	11
decode_bpe_bytes . . . . .	11
decode_timestamp . . . . .	12
decode_with_fallback . . . . .	12
detect_language . . . . .	13
detect_language_from_mel . . . . .	14
detect_language_from_pipeline . . . . .	15
download_tokenizer_files . . . . .	15
download_whisper_model . . . . .	16
dtw_align . . . . .	16
ensure_tokenizer_files . . . . .	17
expand_kv_cache . . . . .	17
extract_segments . . . . .	18
forced_decode . . . . .	18
get_initial_tokens . . . . .	19
get_model_path . . . . .	19
get_weights_path . . . . .	20
greedy_decode . . . . .	20
group_into_words . . . . .	21
hz_to_mel . . . . .	21
is_timestamp_token . . . . .	22
list_downloaded_models . . . . .	22
list_whisper_models . . . . .	23
load_audio . . . . .	23
load_decoder_weights . . . . .	24
load_encoder_weights . . . . .	24
load_mel_filterbank . . . . .	25
load_whisper_model . . . . .	25
load_whisper_weights . . . . .	26
medfilt1 . . . . .	26
mel_to_hz . . . . .	27
model_exists . . . . .	27
pad_or_trim . . . . .	28
parse_device . . . . .	28
parse_dtype . . . . .	29
rearrange_kv_cache . . . . .	29
sample_decode . . . . .	30

split\_audio . . . . . 30

tokenizer\_decode . . . . . 31

tokenizer\_encode . . . . . 31

transcribe . . . . . 32

transcribe\_chunk . . . . . 33

transcribe\_long . . . . . 34

whisper\_attention . . . . . 35

whisper\_config . . . . . 36

whisper\_decoder . . . . . 36

whisper\_decoder\_layer . . . . . 37

whisper\_device . . . . . 37

whisper\_dtype . . . . . 38

whisper\_encoder . . . . . 38

whisper\_encoder\_layer . . . . . 39

whisper\_lang\_from\_id . . . . . 39

whisper\_lang\_token . . . . . 40

whisper\_language\_table . . . . . 40

whisper\_model . . . . . 41

whisper\_pipeline . . . . . 41

WHISPER\_SAMPLE\_RATE . . . . . 42

whisper\_special\_tokens . . . . . 42

whisper\_tokenizer . . . . . 43

**Index** **44**

apply\_bpe *Apply BPE Merges*

**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` *Apply Timestamp Token Rules*

---

**Description**

Enforce Whisper timestamp generation constraints on logits.

**Usage**

```
apply_timestamp_rules(logits, generated, special, sample_begin)
```

**Arguments**

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

**Value**

Modified logits tensor

---

`audio_duration` *Get Audio Duration*

---

**Description**

Get Audio Duration

**Usage**

```
audio_duration(file)
```

**Arguments**

<code>file</code>	Path to audio file
-------------------	--------------------

**Value**

Duration in seconds

---

audio_to_mel	<i>Convert Audio to Mel Spectrogram</i>
--------------	---

---

### 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

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

### Value

torch tensor of shape (1, n\_mels, 3000) for 30s audio

### Examples

```
# 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 = 448L, 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     *Build Reverse Byte Decoder*

---

**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

*Word-Level Timestamp Alignment*


---

### Description

DTW-based alignment of tokens to audio frames using cross-attention weights. Compute Word-Level Timestamps 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 *Copy Weight if Exists*


---

### 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	<i>Create Encoder from Config</i>
----------------	-----------------------------------

---

**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      *Decode BPE Bytes Back to Text*

---

**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	<i>Decode with Temperature Fallback</i>
----------------------	---

---

**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 = 448L,
                    timestamps = FALSE, word_timestamps = FALSE,
                    compression_ratio_threshold = 2.4, logprob_threshold = -1,
                    length_penalty = 1, patience = Inf, 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
length_penalty	Length penalty for beam search
patience	Patience factor for beam search
device	Device

**Value**

List with tokens, cross\_attn\_weights, sum\_logprob, n\_tokens

---

detect_language	<i>Language Detection</i>
-----------------	---------------------------

---

**Description**

Detect the spoken language in an audio file using Whisper. Detect Language 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**

<code>pipe</code>	A <code>whisper_pipeline</code> object
<code>file</code>	Path to audio file, or numeric vector of audio samples
<code>top_k</code>	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**

<code>model</code>	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

*DTW Alignment*

---

### 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

*Ensure Tokenizer Files are Downloaded*

---

**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

*Expand KV Cache for Beam Search*

---

**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      *Get Initial Decoder Tokens*

---

**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      *Get Model Cache Path*

---

**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 = 448L, 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      *Check if Token is Timestamp*

---

**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	<i>List Available Models</i>
---------------------	------------------------------

---

**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    *Load Decoder Weights*

---

**Description**

Load Decoder Weights

**Usage**

```
load_decoder_weights(decoder, weights)
```

**Arguments**

decoder	WhisperDecoder module
weights	Named list of tensors

---

load\_encoder\_weights    *Load Encoder Weights*

---

**Description**

Load Encoder Weights

**Usage**

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

**Arguments**

encoder	WhisperEncoder module
weights	Named list of tensors

---

load\_mel\_filterbank     *Load Pre-computed Mel Filterbank*

---

**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     *Load Whisper Model*

---

**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 *Load Weights from Safetensors*

---

**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 *1D Median Filter*

---

**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

---

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

---

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 = 448L, 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

---

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)
```

**Arguments**

text	Character string to encode
vocab	Vocabulary mapping (token -> id)
merge_ranks	Merge ranking for BPE

**Value**

Integer vector of token IDs

---

transcribe	<i>Transcribe Audio</i>
------------	-------------------------

---

**Description**

Transcribe speech from an audio file using Whisper. For repeated transcription, use [whisper\\_pipeline\(\)](#) to load the model once.

**Usage**

```
transcribe(file, model = "tiny", language = NULL, task = "transcribe",
           timestamps = FALSE, word_timestamps = FALSE, beam_size = 1L,
           temperatures = 0, best_of = 1L, compression_ratio_threshold = 2.4,
           logprob_threshold = -1, length_penalty = 1, patience = Inf,
           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).
device	Device: "auto", "cpu", "cuda"
dtype	Data type: "auto", "float16", "float32"
verbose	Print progress messages

**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 = 0,
                 best_of = 1L, compression_ratio_threshold = 2.4,
                 logprob_threshold = -1, length_penalty = 1, patience = Inf,
                 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.
length_penalty	Length penalty exponent for beam search.
patience	Patience factor for beam search.
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 = 0, best_of = 1L,
                compression_ratio_threshold = 2.4, logprob_threshold = -1,
                length_penalty = 1, patience = Inf, 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.
device	Device
dtype	Dtype
verbose	Verbose

**Value**

Combined transcription result

---

whisper\_attention      *Whisper Encoder*

---

**Description**

Transformer encoder for processing mel spectrograms. 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 *Whisper Decoder*

---

### Description

Transformer decoder with cross-attention to encoder outputs. Decoder Layer 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 *Device and Dtype Management*

---

### Description

Utilities for managing torch devices and data types. Get Default Device 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.

**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 *Encoder Layer*

---

**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\_lang\_from\_id *Get Language Code from Token ID*

---

**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      *Get Language Token ID*

---

**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\_language\_table  
*Whisper Language Table*

---

**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_model	<i>Whisper Model</i>
---------------	----------------------

---

**Description**

Full Whisper model combining encoder and decoder. Whisper Model Module

**Usage**

```
whisper_model(config)
```

**Arguments**

config	Model configuration
--------	---------------------

---

whisper_pipeline	<i>Whisper Transcription</i>
------------------	------------------------------

---

**Description**

Main transcription API for Whisper. Create a Whisper Pipeline 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     *Audio Preprocessing for Whisper*

---

**Description**

Convert audio files to mel spectrograms for Whisper input. 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**

model                    Model name (default: "tiny")

**Value**

Named list of special token IDs

---

whisper_tokenizer	<i>Whisper BPE Tokenizer</i>
-------------------	------------------------------

---

**Description**

Byte-pair encoding tokenizer for Whisper models. Create Whisper Tokenizer Load or create a Whisper tokenizer from HuggingFace vocab files.

**Usage**

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

**Arguments**

model	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))
}
```

# Index

## \* datasets

- WHISPER\_SAMPLE\_RATE, 42
- apply\_bpe, 3
- apply\_timestamp\_rules, 4
- audio\_duration, 4
- audio\_to\_mel, 5
- beam\_search\_decode, 5
- build\_byte\_decoder, 6
- byte\_to\_token, 7
- clean\_text, 7
- compression\_ratio, 8
- compute\_stft, 8
- compute\_word\_timestamps, 9
- copy\_if\_exists, 9
- create\_decoder, 10
- create\_encoder, 10
- create\_mel\_filterbank\_fallback, 11
- decode\_bpe\_bytes, 11
- decode\_timestamp, 12
- decode\_with\_fallback, 12
- detect\_language, 13
- detect\_language\_from\_mel, 14
- detect\_language\_from\_pipeline, 15
- download\_tokenizer\_files, 15
- download\_whisper\_model, 16
- dtw\_align, 16
- ensure\_tokenizer\_files, 17
- expand\_kv\_cache, 17
- extract\_segments, 18
- forced\_decode, 18
- get\_initial\_tokens, 19
- get\_model\_path, 19
- get\_weights\_path, 20
- greedy\_decode, 20
- group\_into\_words, 21
- hz\_to\_mel, 21
- is\_timestamp\_token, 22
- list\_downloaded\_models, 22
- list\_whisper\_models, 23
- load\_audio, 23
- load\_decoder\_weights, 24
- load\_encoder\_weights, 24
- load\_mel\_filterbank, 25
- load\_whisper\_model, 25
- load\_whisper\_weights, 26
- medfilt1, 26
- mel\_to\_hz, 27
- model\_exists, 27
- pad\_or\_trim, 28
- parse\_device, 28
- parse\_dtype, 29
- rearrange\_kv\_cache, 29
- sample\_decode, 30
- split\_audio, 30
- tokenizer\_decode, 31
- tokenizer\_encode, 31
- transcribe, 32
- transcribe\_chunk, 33
- transcribe\_long, 34
- whisper\_attention, 35
- whisper\_config, 36
- whisper\_decoder, 36
- whisper\_decoder\_layer, 37
- whisper\_device, 37
- whisper\_dtype, 38
- whisper\_encoder, 38

`whisper_encoder_layer`, [39](#)  
`whisper_lang_from_id`, [39](#)  
`whisper_lang_token`, [40](#)  
`whisper_language_table`, [40](#)  
`whisper_model`, [41](#)  
`whisper_pipeline`, [32](#), [41](#)  
`WHISPER_SAMPLE_RATE`, [42](#)  
`whisper_special_tokens`, [42](#)  
`whisper_tokenizer`, [43](#)