Package: chatterbox (via r-universe)

June 26, 2026

Title Text-to-Speech Using the 'Chatterbox' Engine

Version 0.2.1

Date 2026-06-21

Description A native R 'torch' port of the 'Chatterbox' text-to-speech engine <https://github.com/resemble-ai/chatterbox>. Provides speech synthesis with voice cloning; model weights are downloaded from 'HuggingFace' <https://huggingface.co/> via the 'hfhub' package.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/cornball-ai/chatterbox

BugReports https://github.com/cornball-ai/chatterbox/issues

Depends R (>= 4.0.0)

Imports torch (>= 0.17.0), tuneR, jsonlite, hfhub, safetensors (>= 0.1.1)

Suggests tinytest

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

Repository https://cornball-ai.r-universe.dev

Date/Publication 2026-06-24 01:03:13 UTC

RemoteUrl https://github.com/cornball-ai/chatterbox

RemoteRef HEAD

RemoteSha da02971d3a14c1bdf9ec6af560233ba7f6adedeb

Contents

`apply_llama3_rope_scaling` *Apply Llama3-style RoPE scaling*

Description

Apply Llama3-style RoPE scaling

Usage

```
apply_llama3_rope_scaling(inv_freq, scaling, dim)
```

Arguments

inv_freq	Inverse frequencies
scaling	Scaling configuration
dim	Dimension

Value

Scaled inverse frequencies

---

apply_rotary_emb_s3 *Apply rotary position embeddings*

Description

Apply rotary position embeddings

Usage

```
apply_rotary_emb_s3(xq, xk, freqs_cis)
```

Arguments

xq	Query tensor
xk	Key tensor
freqs_cis	Precomputed frequencies

Value

List with rotated q and k

---

apply_rotary_pos_emb *Apply rotary position embeddings to Q and K*

Description

Apply rotary position embeddings to Q and K

Usage

```
apply_rotary_pos_emb(q, k, cos, sin, position_ids)
```

Arguments

q	Query tensor (batch, heads, seq, head_dim)
k	Key tensor (batch, heads, seq, head_dim)
cos	Cosine cache
sin	Sine cache
position_ids	Position indices

Value

List with rotated q and k

---

attention_block *Attention block for perceiver*

Description

Attention block for perceiver

Usage

```
attention_block(embed_dim = 1024, num_heads = 4)
```

Arguments

embed_dim	Embedding dimension (default 1024)
num_heads	Number of attention heads (default 4)

Value

nn_module

---

basic_res_block *Basic residual block for FCM*

Description

Basic residual block for FCM

Usage

```
basic_res_block(in_planes, planes, stride = 1)
```

Arguments

in_planes	Input channels
planes	Output channels
stride	Stride for downsampling

Value

nn_module

---

basic_transformer_block *Basic transformer block*

Description

Basic transformer block

Usage

```
basic_transformer_block(dim, num_heads = 8L)
```

Arguments

dim	Hidden dimension
num_heads	Number of attention heads

Value

nn_module

---

cam_dense_tdnn_block *CAM Dense TDNN Block (multiple layers with dense connections)*

Description

CAM Dense TDNN Block (multiple layers with dense connections)

Usage

```
cam_dense_tdnn_block(num_layers, in_channels, out_channels, bn_channels,
                     kernel_size, dilation = 1)
```

Arguments

num_layers	Number of layers
in_channels	Input channels
out_channels	Output channels per layer
bn_channels	Bottleneck channels
kernel_size	Kernel size
dilation	Dilation

Value

nn_module

---

cam_dense_tdnn_layer *CAM Dense TDNN Layer*

Description

CAM Dense TDNN Layer

Usage

```
cam_dense_tdnn_layer(in_channels, out_channels, bn_channels, kernel_size,
                     dilation = 1)
```

Arguments

in_channels	Input channels
out_channels	Output channels
bn_channels	Bottleneck channels
kernel_size	Kernel size
dilation	Dilation

Value

nn_module

cam_layer *CAM (Context-Aware Masking) Layer*

Description

CAM (Context-Aware Masking) Layer

Usage

```
cam_layer(bn_channels, out_channels, kernel_size, stride = 1, padding = 0,
          dilation = 1, reduction = 2)
```

Arguments

bn_channels	Bottleneck channels
out_channels	Output channels
kernel_size	Kernel size
stride	Stride
padding	Padding
dilation	Dilation
reduction	Channel reduction factor

Value

nn_module

campplus *CAMPPlus speaker encoder*

Description

CAMPPlus speaker encoder

Usage

```
campplus(feat_dim = 80, embedding_size = 192, growth_rate = 32,
         init_channels = 128)
```

Arguments

feat_dim	Input feature dimension (default 80)
embedding_size	Output embedding size (default 192)
growth_rate	Dense block growth rate (default 32)
init_channels	Initial TDNN channels (default 128)

Value

nn_module

---

causal_block1d *Causal Block 1D - CausalConv + LayerNorm + Mish*

Description

Causal Block 1D - CausalConv + LayerNorm + Mish

Usage

```
causal_block1d(in_channels, out_channels, kernel_size = 3L)
```

Arguments

in_channels	Input channels
out_channels	Output channels
kernel_size	Kernel size

Value

nn_module

---

causal_cfm *Causal Conditional Flow Matching*

Description

Causal Conditional Flow Matching

Usage

```
causal_cfm(in_channels = 320, out_channels = 80, spk_emb_dim = 80,
           meanflow = FALSE)
```

Arguments

in_channels	Input channels (x + mu + spks + cond)
out_channels	Output channels (mel bins)
spk_emb_dim	Speaker embedding dimension
meanflow	Logical. Use mean-flow formulation. Default FALSE.

Value

nn_module

causal_conv1d *Causal Conv1d - pads left only*

Description

Causal Conv1d - pads left only

Usage

```
causal_conv1d(in_channels, out_channels, kernel_size, stride = 1L, dilation = 1L)
```

Arguments

in_channels	Input channels
out_channels	Output channels
kernel_size	Kernel size
stride	Stride (default 1)
dilation	Dilation (default 1)

Value

nn_module

causal_masked_diff_xvec *Causal Masked Diff with Xvector*

Description

Causal Masked Diff with Xvector

Usage

```
causal_masked_diff_xvec(vocab_size = 6561, input_size = 512, output_size = 80,
                        spk_embed_dim = 192, input_frame_rate = 25,
                        token_mel_ratio = 2, meanflow = FALSE)
```

Arguments

vocab_size	Speech token vocabulary size
input_size	Token embedding size
output_size	Mel bins
spk_embed_dim	Speaker embedding dimension
input_frame_rate	Input frame rate for audio processing
token_mel_ratio	Ratio of tokens to mel frames
meanflow	Logical. Use mean-flow formulation. Default FALSE.

Value

nn_module

---

causal_resnet_block1d *Causal ResNet Block 1D*

Description

Causal ResNet Block 1D

Usage

```
causal_resnet_block1d(in_channels, out_channels, time_embed_dim = 1024L)
```

Arguments

in_channels	Input channels
out_channels	Output channels
time_embed_dim	Time embedding dimension

Value

nn_module

cfm_attention *Self-attention for transformer block*

Description

Self-attention for transformer block

Usage

```
cfm_attention(dim, num_heads = 8L, head_dim = 64L)
```

Arguments

dim	Hidden dimension
num_heads	Number of attention heads
head_dim	Head dimension (default 64)

Value

nn_module

cfm_estimator *CFM Estimator (ConditionalDecoder)*

Description

UNet-style architecture with: - 1 down block (320 -> 256, with 4 transformer blocks) - 12 mid blocks (256 -> 256, each with 4 transformer blocks) - 1 up block (512 -> 256 with skip connection, 4 transformer blocks)

Usage

```
cfm_estimator(in_channels = 320L, out_channels = 80L, hidden_dim = 256L,
              num_mid_blocks = 12L, num_transformer_blocks = 4L,
              meanflow = FALSE)
```

Arguments

in_channels	Input channels (default 320 = x + mu + spks + cond)
out_channels	Output channels (default 80 = mel bins)
hidden_dim	Hidden dimension (default 256)
num_mid_blocks	Number of mid blocks (default 12)
num_transformer_blocks	Transformer blocks per layer (default 4)
meanflow	Logical. Use mean-flow formulation. Default FALSE.

Value

nn_module

`chatterbox` *Create (and load) a Chatterbox TTS model*

Description

Constructs the model object and, by default, loads the pretrained weights in the same call - the Python reference's `from_pretrained`/`from_local` do both at once. Pass `load = FALSE` for the bare object (e.g. to inspect it or test the not-loaded error paths), then load later with `load_chatterbox`.

Usage

```
chatterbox(device = "cpu", turbo = FALSE, load = TRUE, tune_gc = TRUE)
```

Arguments

`device`	Device to use ("cpu", "cuda", "mps", etc.)
`turbo`	Use turbo model (GPT-2 backbone, MeanFlow decoder). Default FALSE.
`load`	Load pretrained weights before returning. Default TRUE. Requires a prior download (`download_chatterbox_models`).
`tune_gc`	Tune torch's CUDA GC rates for faster inference (CUDA only, and only when unset). Persistent session side effect; default TRUE. See Details.

Details

When `tune_gc = TRUE` (the default) and `device` is CUDA, this raises torch's allocator GC floors before the first CUDA op. torch otherwise runs `gc()` on nearly every allocation once a model occupies more than 20\ `torch.cuda_allocator_reserved_rate` (the model footprint over VRAM) and `torch.threshold_call_gc`, only when they are unset, so an explicit setting always wins. This is a deliberate, persistent side effect (torch reads the rates later, at CUDA init); pass `tune_gc = FALSE` to skip it.

Value

Chatterbox TTS model object, loaded unless `load = FALSE`

Examples

```
## Not run:
# Construct and load the standard model on GPU
model <- chatterbox("cuda")

# Bare object without weights (load later with load_chatterbox())
model <- chatterbox("cuda", load = FALSE)

## End(Not run)
```

`chatterbox_gc_options` *Recommended torch garbage-collection settings for chatterbox*

Description

torch's allocators invoke a full R garbage collection based on settings that are read ONCE at torch startup. The default CUDA trigger (collections begin once torch reserves 20 percent of the card) sits below chatterbox's ~4.6 GB loaded footprint on most GPUs, which makes autoregressive inference collection-bound: ~91 percent of pure-R generation wall time is GC, and even compiled-loop backends are throttled by it (their allocations flow through the same allocator). With the trigger line above the model floor, pure R runs ~10x faster and the jit backend ~15x.

Usage

```
chatterbox_gc_options(vram_gb = NULL)
```

Arguments

`vram_gb`	Total GPU memory in GB. Default: detected via nvidia-smi, falling back to 16.

Details

Only one option matters for speed: `torch.cuda_allocator_reserved_rate`. Sweeps over 0.3-0.8 all give identical speed; the value just chooses how high the VRAM plateau sits. `torch.cuda_allocator_allocated_rate` (default 0.8) caps that plateau and can be lowered to ~0.6 on shared GPUs at no speed cost. The remaining settings (`torch.threshold_call_gc`, `torch.cuda_allocator_allocated_reserved_rate`) measured as not worth touching.

This helper does not (and cannot) change the settings for the current session: torch reads them at initialization, so they belong in your .Rprofile or at the very top of a script, before torch loads. Printing the returned object shows the exact snippet for this machine; the helper warns when torch is already initialized. Scripts can apply the values directly with `do.call(options, chatterbox_gc_options())` (again: before torch loads).

Rule of thumb for loops: collect once per utterance, not thousands of times inside it. `tts_chunked` does this automatically; in your own batch loops, call `gc()` after each `generate()`.

Value

A named list of the recommended `options()` values, classed `"chatterbox_gc_options"` so it prints as the full tuning advice for this machine.

Examples

```
chatterbox_gc_options(vram_gb = 16)
```

compute_rope_frequencies *Compute rotary position embeddings frequencies*

Description

Compute rotary position embeddings frequencies

Usage

```
compute_rope_frequencies(dim, max_seq_len, theta = 5e+05, scaling = NULL,
                         device = "cpu")
```

Arguments

dim	Dimension of embeddings
max_seq_len	Maximum sequence length
theta	Base frequency
scaling	Rope scaling configuration (optional)
device	Device to create tensors on

Value

List with cos and sin caches

compute_ve_mel *Compute mel spectrogram for voice encoder*

Description

Uses power spectrum (magnitude^2) without log compression, matching Python's mel_type="amp" and mel_power=2.0.

Usage

```
compute_ve_mel(wav, config = voice_encoder_config())
```

Arguments

wav	Audio samples (numeric vector)
config	Voice encoder config

Value

Mel spectrogram (batch, time, n_mels)

conformer_encoder_layer *Conformer Encoder Layer*

Description

Single conformer block with attention and feed-forward (no convolution).

Usage

```
conformer_encoder_layer(n_feat = 512, n_head = 8, n_ffn = 2048,
                        dropout_rate = 0.1)
```

Arguments

n_feat	Feature dimension
n_head	Number of attention heads
n_ffn	Feed-forward hidden dimension
dropout_rate	Dropout rate

Value

nn_module

conv_rnn_f0_predictor *Convolutional RNN F0 Predictor*

Description

Convolutional RNN F0 Predictor

Usage

```
conv_rnn_f0_predictor(in_channels = 80, cond_channels = 512)
```

Arguments

in_channels	Input channels (mel bins)
cond_channels	Hidden channels

Value

nn_module

create_kv_cache *Create pre-allocated KV cache*

Description

Create pre-allocated KV cache

Usage

```
create_kv_cache(batch_size, n_layers, n_heads, head_dim, max_len, device)
```

Arguments

batch_size	Batch size
n_layers	Number of transformer layers
n_heads	Number of attention heads
head_dim	Head dimension
max_len	Maximum sequence length
device	Device to allocate on

Value

List with k_cache, v_cache, valid_mask

create_mel_filterbank *Create mel filterbank*

Description

Create mel filterbank

Usage

```
create_mel_filterbank(sr, n_fft, n_mels, fmin = 0, fmax = NULL,
                      norm = "slaney", htk = FALSE)
```

Arguments

sr	Sample rate
n_fft	FFT size
n_mels	Number of mel bins
fmin	Minimum frequency
fmax	Maximum frequency
norm	Character. Normalization type. Default "slaney".
htk	Logical. Use HTK formula. Default FALSE.

Value

Mel filterbank matrix (n_mels x (n_fft/2 + 1))

`create_voice_embedding` *Create voice embedding from reference audio*

Description

Create voice embedding from reference audio

Usage

```
create_voice_embedding(model, audio, sample_rate = NULL, autocast = NULL,
                       norm_loudness = NULL)
```

Arguments

`model`	Chatterbox model
`audio`	Reference audio (file path, numeric vector, or torch tensor)
`sample_rate`	Sample rate of audio (if not a file)
`autocast`	Ignored (kept for API compatibility)
`norm_loudness`	Normalize the reference to -27 LUFS before conditioning (`normalize_loudness`). Default matches Python: `TRUE` for turbo models, `FALSE` for standard.

Value

Voice embedding that can be used for synthesis

Examples

```
## Not run:
model <- chatterbox("cuda")
voice <- create_voice_embedding(model, "reference_voice.wav")
res <- generate(model, "Reusing a cached voice.", voice)

## End(Not run)
```

dense_layer *Dense layer for final embedding*

Description

Dense layer for final embedding

Usage

```
dense_layer(in_channels, out_channels)
```

Arguments

in_channels	Input channels
out_channels	Output channels

Value

nn_module

download_chatterbox_models *Download Chatterbox Models from HuggingFace*

Description

Download all Chatterbox model files from HuggingFace. In interactive sessions, asks for user consent before downloading.

Usage

```
download_chatterbox_models(force = FALSE)
```

Arguments

force	Re-download even if files exist

Value

Named list of local file paths (invisibly)

Examples

```
## Not run:
# Download models (~2GB)
download_chatterbox_models()

## End(Not run)
```

download_chatterbox_turbo_models *Download Chatterbox Turbo Models from HuggingFace*

Description

Download all Chatterbox Turbo model files from HuggingFace. The turbo model uses a GPT-2 backbone and MeanFlow decoder for faster inference.

Usage

```
download_chatterbox_turbo_models(force = FALSE)
```

Arguments

force	Re-download even if files exist

Value

Named list of local file paths (invisibly)

Examples

```
## Not run:
download_chatterbox_turbo_models()

## End(Not run)
```

drop_invalid_tokens *Drop invalid speech tokens*

Description

Python parity: slice to the span after the first SOS and before the first EOS (s3tokenizer drop_invalid_tokens), then drop any remaining out-of-vocab ids (tts.py filters < SPEECH_VOCAB_SIZE on top). The old elementwise filter kept post-EOS garbage when generation hit the token cap with a mid-stream EOS.

Usage

```
drop_invalid_tokens(tokens)
```

Arguments

tokens	Token tensor or integer vector (0-indexed values)

Value

Filtered tokens, same type as input

---

espnet_rel_positional_encoding *Sinusoidal positional encoding (Espnet RelPositionalEncoding)*

Description

Creates sinusoidal positional embeddings for use with relative position attention. Includes scaling by sqrt(d_model) and adds positional embedding to the input.

Usage

```
espnet_rel_positional_encoding(d_model = 512, dropout_rate = 0.1, max_len = 5000)
```

Arguments

d_model	Model dimension
dropout_rate	Numeric. Dropout rate. Default 0.1.
max_len	Maximum sequence length

Value

nn_module

---

fcm_module *Factorized Convolutional Module (FCM)*

Description

Factorized Convolutional Module (FCM)

Usage

```
fcm_module(m_channels = 32, feat_dim = 80)
```

Arguments

m_channels	Number of channels
feat_dim	Input feature dimension (mel bins)

Value

nn_module

feed_forward *Feed-forward network for transformer Matches diffusers FeedForward: net = [GELU(proj), Dropout, Linear]*

Description

Feed-forward network for transformer Matches diffusers FeedForward: net = [GELU(proj), Dropout, Linear]

Usage

```
feed_forward(dim, hidden_dim = NULL)
```

Arguments

dim	Input dimension
hidden_dim	Hidden dimension (typically 4x dim)

Value

nn_module

fsmn_multi_head_attention *FSMN Multi-Head Attention*

Description

Multi-head attention with Frequency-domain Self-attention Memory Network

Usage

```
fsmn_multi_head_attention(n_state, n_head, kernel_size = 31L)
```

Arguments

n_state	Hidden dimension
n_head	Number of heads
kernel_size	FSMN kernel size (default 31)

Value

nn_module

fsq_codebook *FSQ Codebook module*

Description

FSQ Codebook module

Usage

```
fsq_codebook(dim, level = 3L)
```

Arguments

dim	Input dimension (n_audio_state)
level	Quantization level (default 3)

Value

nn_module

fsq_vector_quantization *FSQ Vector Quantization wrapper*

Description

FSQ Vector Quantization wrapper

Usage

```
fsq_vector_quantization(dim, codebook_size = 6561L)
```

Arguments

dim	Input dimension
codebook_size	Codebook size (must be 6561 = 3^8)

Value

nn_module

gelu_with_proj *GELU activation with projection (matches diffusers GELU structure)*

Description

GELU activation with projection (matches diffusers GELU structure)

Usage

```
gelu_with_proj(dim_in, dim_out)
```

Arguments

dim_in	Input dimension
dim_out	Output dimension

Value

nn_module

generate *Generate speech from text*

Description

Generate speech from text

Usage

```
generate(model, text, voice, exaggeration = 0.5, cfg_weight = 0.5,
         temperature = 0.8, top_p = 1, min_p = 0.05, autocast = NULL,
         traced = FALSE, backend = c("r", "jit"), top_k = 1000L,
         repetition_penalty = 1.2, normalize_text = FALSE,
         max_new_tokens = 1000L, max_cache_len = NULL, cfm_len = NULL,
         skip_vocoder = FALSE, output_path = NULL)
```

Arguments

model	Chatterbox model
text	Text to synthesize
voice	Voice embedding from create_voice_embedding() or path to reference audio
exaggeration	Emotion/expression exaggeration level (0-1, default 0.5)
cfg_weight	Classifier-free guidance weight (higher = more adherence to text, default 0.5)
temperature	Sampling temperature (default 0.8)

`top_p` Top-p (nucleus) sampling threshold. Default 1.0 (disabled), matching the Python reference.

`min_p` Minimum probability threshold relative to the most likely token (default 0.05, matching the Python reference). Standard model only.

`autocast` Use mixed precision (float16) on CUDA for faster inference. Default FALSE: the Python reference runs float32, and float16 output diverges slightly. Opt in for speed on tight VRAM.

`traced` Logical. Use JIT-traced inference. Default FALSE.

`backend` Character. Inference backend, either "r" or "jit". Default "r". The jit backend runs each token's full 30-layer forward as one TorchScript call (compiled once per session, in milliseconds, via `torch::jit_compile`): with tuned GC settings (see `chatterbox_gc_options`) it is the fastest native path (~11 ms/token long-form), auto-sizes its KV cache so generation always completes, and ships no compiled code. It replaced an equivalent C++ backend (within ~20\ that required linking against torch's private libraries.

`top_k` Integer. Top-k sampling parameter (turbo model only). Default 1000.

`repetition_penalty` Numeric. Repetition penalty. Default 1.2. Applied sign-dependently like HF transformers: positive logits are divided, negative ones multiplied.

`normalize_text` Logical. Apply the R-specific internal-caps mitigation (lowercase words with internal capitals). Default FALSE: it addressed a "first word then silence" failure that was actually the column-major/STFT bug, now fixed - with the model corrected, leaving caps intact also preserves intended emphasis (ALL CAPS reads as emphasis) and acronyms. Set TRUE only if specific text misbehaves. Punctuation normalization (whitespace collapse, first-letter capitalization, trailing period) always runs, matching the Python reference implementation.

`max_new_tokens` Maximum speech tokens to generate (default 1000, = 40 s of audio; the model's own ceiling is 4096).

`max_cache_len` KV cache positions for the jit and traced backends. Default NULL: jit auto-sizes so generation always fits (~1 MB VRAM per position); traced keeps its 350-position trace (a new size triggers a fresh ~50 s trace). Ignored by the pure-R backend, which has no pre-allocated cache.

`cfm_len` Optional explicit traced-CFM length (the padded mel sequence, = 640 + 2 * tokens). Default NULL: the standard traced path sizes it from the tokens actually generated, rounded up to the 250/500/1000 bucket ladder, so a slow speaker is covered without guessing from text length. Pass a value to pin it (e.g. to pre-trace a bucket). Ignored when not traced or for turbo.

`skip_vocoder` Logical. If TRUE, stop after flow matching and return the mel spectrogram instead of audio (Python 0.1.7's `skip_vocoder`). The result has a `mel` element (tensor, batch x 80 x frames; 50 frames/s) and no `audio`.

`output_path` Optional WAV path. When set, the audio is also written there (as a side effect) and the returned list gains a `path` element; the audio is still returned in full. Incompatible with `skip_vocoder` (no audio to write). Default NULL.

Value

List with elements:

audio Numeric vector of audio samples (omitted when `skip_vocoder = TRUE`, which returns `mel` instead)

sample_rate Sample rate in Hz

eos_found Logical. Whether the model emitted an end-of-speech token (TRUE) or hit the token cap (FALSE). FALSE often indicates garbage output and a need to retry or split the input.

n_tokens Number of speech tokens generated

audio_sec Audio duration in seconds

path Output file path (only when `output_path` is set)

Examples

```
## Not run:
model <- chatterbox("cuda")
res <- generate(model, "Hello world!", "reference_voice.wav")
write_audio(res$audio, res$sample_rate, "hello.wav")

# Fastest native path: TorchScript decode loop
res <- generate(model, "Hello world!", "reference_voice.wav",
                backend = "jit")

## End(Not run)
```

`generate_batch` *Generate speech for several texts with one batched synthesis pass*

Description

Runs T3 token generation per text (autoregressive, sequential), then synthesizes ALL utterances in a single batched S3Gen pass (one CFM solve and one vocoder call over the padded batch). Per-utterance results match single `generate` calls up to CFM noise handling - the fixed noise buffer means row i sees the same initial noise it would alone. Standard model only.

Usage

```
generate_batch(model, texts, voice, ...)
```

Arguments

`model`	Loaded chatterbox model (standard, not turbo)
`texts`	Character vector of texts to synthesize
`voice`	Shared voice: voice_embedding or reference audio path

`...` Arguments passed through to the T3 stage, as in `generate` (exaggeration, cfg_weight, temperature, top_p, min_p, backend, repetition_penalty, normalize_text, max_new_tokens, max_cache_len). `traced` and `autocast` affect the T3 stage only: the batched S3Gen synthesis always runs eager float32 (traced CFM is fixed at batch 1). The CFM trace-bucket sizing used by `generate` therefore does not apply here - batched S3Gen pads dynamically to the batch's longest utterance.

Value

List with one `generate`-style result per text (audio, sample_rate, eos_found, n_tokens, audio_sec)

Examples

```
## Not run:
model <- chatterbox("cuda")
res <- generate_batch(model,
                      c("First sentence.", "Second sentence."),
                      "reference_voice.wav")
write_audio(res[[1]]$audio, res[[1]]$sample_rate, "first.wav")

## End(Not run)
```

`get_conv_padding` *Get padding for convolution*

Description

Get padding for convolution

Usage

```
get_conv_padding(kernel_size, dilation = 1)
```

Arguments

`kernel_size`	Kernel size
`dilation`	Dilation rate

Value

Padding size

get_traced_layers *Get or create traced layers for cached inference*

Description

Get or create traced layers for cached inference

Usage

```
get_traced_layers(model, max_cache_len = 350L)
```

Arguments

model	T3 model
max_cache_len	Maximum cache length

Value

List of traced layer modules

gpt2_attention *GPT-2 Attention (combined QKV projection)*

Description

GPT-2 Attention (combined QKV projection)

Usage

```
gpt2_attention(config)
```

Arguments

config	GPT-2 config

Value

nn_module

gpt2_block *GPT-2 Transformer Block*

Description

GPT-2 Transformer Block

Usage

```
gpt2_block(config)
```

Arguments

config	GPT-2 config

Value

nn_module

gpt2_config *GPT-2 Model Configuration*

Description

GPT-2 Model Configuration

Usage

```
gpt2_config()
```

Value

List with GPT-2 medium config

gpt2_layer_norm *GPT-2 Layer Normalization*

Description

GPT-2 Layer Normalization

Usage

```
gpt2_layer_norm(hidden_size, eps = 1e-05)
```

Arguments

hidden_size	Dimension
eps	Epsilon

Value

nn_module

gpt2_mlp *GPT-2 MLP (GELU activation)*

Description

GPT-2 MLP (GELU activation)

Usage

```
gpt2_mlp(config)
```

Arguments

config	GPT-2 config

Value

nn_module

gpt2_model *GPT-2 Model (transformer backbone)*

Description

GPT-2 Model (transformer backbone)

Usage

```
gpt2_model(config = NULL)
```

Arguments

config	GPT-2 configuration

Value

nn_module

hifigan_resblock *HiFiGAN Residual Block*

Description

HiFiGAN Residual Block

Usage

```
hifigan_resblock(channels = 512, kernel_size = 3, dilations = c(1, 3, 5))
```

Arguments

channels	Number of channels
kernel_size	Kernel size
dilations	List of dilation rates

Value

nn_module

`hift_generator` *HiFTNet Generator*

Description

Neural Source Filter + ISTFTNet Reference: https://arxiv.org/abs/2309.09493

Usage

```
hift_generator(in_channels = 80, base_channels = 512, nb_harmonics = 8,
               sampling_rate = 22050, nsf_alpha = 0.1, nsf_sigma = 0.003,
               nsf_voiced_threshold = 10, upsample_rates = c(8, 8),
               upsample_kernel_sizes = c(16, 16), istft_n_fft = 16,
               istft_hop_len = 4, resblock_kernel_sizes = c(3, 7, 11),
              resblock_dilation_sizes = list(c(1, 3, 5), c(1, 3, 5), c(1, 3, 5)),
               source_resblock_kernel_sizes = c(7, 11),
               source_resblock_dilation_sizes = list(c(1, 3, 5), c(1, 3, 5)),
               lrelu_slope = 0.1, audio_limit = 0.99)
```

Arguments

`in_channels`	Input mel channels
`base_channels`	Base channel count
`nb_harmonics`	Number of harmonics for source filter
`sampling_rate`	Output sample rate
`nsf_alpha`	NSF sine amplitude
`nsf_sigma`	NSF noise std
`nsf_voiced_threshold`	F0 voiced threshold
`upsample_rates`	Upsampling rates
`upsample_kernel_sizes`	Upsampling kernel sizes
`istft_n_fft`	ISTFT FFT size
`istft_hop_len`	ISTFT hop length
`resblock_kernel_sizes`	ResBlock kernel sizes
`resblock_dilation_sizes`	ResBlock dilations
`source_resblock_kernel_sizes`	Source resblock kernels
`source_resblock_dilation_sizes`	Source resblock dilations
`lrelu_slope`	LeakyReLU slope
`audio_limit`	Output clipping limit

Value

nn_module

---

init_cache_from_first *Initialize cache with first token K/V values*

Description

Initialize cache with first token K/V values

Usage

```
init_cache_from_first(cache, past_key_values)
```

Arguments

cache	Cache list from create_kv_cache
past_key_values	List of K/V from first forward pass

Value

Updated cache (and seq_len as attribute)

---

integrated_loudness *Integrated loudness (ITU-R BS.1770-4)*

Description

Measures the integrated gated loudness of a mono signal in LUFS, matching pyloudnorm's K-weighting meter (the measurement Python chatterbox turbo applies to reference audio).

Usage

```
integrated_loudness(samples, sample_rate)
```

Arguments

samples	Numeric vector of mono audio samples.
sample_rate	Sample rate in Hz.

Value

Loudness in LUFS (`-Inf` for silence or when no block passes the gates).

Examples

```
samples <- sin(2 * pi * 440 * seq(0, 1, length.out = 48000))
integrated_loudness(samples, 48000)
```

`is_loaded` *Check if model is loaded*

Description

Check if model is loaded

Usage

```
is_loaded(model)
```

Arguments

`model`	Chatterbox model

Value

TRUE if model is loaded

`learned_position_embeddings` *Learned position embeddings module*

Description

Learned position embeddings module

Usage

```
learned_position_embeddings(seq_len, model_dim, init_std = 0.02)
```

Arguments

`seq_len`	Maximum sequence length
`model_dim`	Embedding dimension
`init_std`	Initialization standard deviation

Value

nn_module

linear_no_subsampling *Linear No Subsampling layer*

Description

Projects input to model dimension with layer norm and positional encoding.

Usage

```
linear_no_subsampling(input_dim = 512, output_dim = 512, dropout_rate = 0.1)
```

Arguments

input_dim	Input dimension
output_dim	Output dimension
dropout_rate	Dropout rate

Value

nn_module

llama_attention *Llama attention module*

Description

Llama attention module

Usage

```
llama_attention(config, layer_idx)
```

Arguments

config	Model configuration
layer_idx	Layer index

Value

nn_module

llama_config_520m *Create Llama 520M configuration*

Description

Create Llama 520M configuration

Usage

```
llama_config_520m()
```

Value

List with model configuration

llama_decoder_layer *Llama decoder layer*

Description

Llama decoder layer

Usage

```
llama_decoder_layer(config, layer_idx)
```

Arguments

config	Model configuration
layer_idx	Layer index

Value

nn_module

llama_mlp *Llama MLP module*

Description

Llama MLP module

Usage

```
llama_mlp(config)
```

Arguments

config	Model configuration

Value

nn_module

llama_model *Llama model (decoder only)*

Description

Llama model (decoder only)

Usage

```
llama_model(config = NULL)
```

Arguments

config	Model configuration (default: 520M)

Value

nn_module

llama_rms_norm *RMS Normalization module*

Description

RMS Normalization module

Usage

```
llama_rms_norm(hidden_size, eps = 1e-05)
```

Arguments

hidden_size	Dimension to normalize
eps	Epsilon for numerical stability

Value

nn_module

load_chatterbox *Load Chatterbox model weights*

Description

Load pretrained weights for all model components. Requires prior download via download_chatterbox_models. Idempotent: an already-loaded model is returned unchanged, so chatterbox(load = TRUE) followed by a stray load_chatterbox() does not reload.

Usage

```
load_chatterbox(model)
```

Arguments

model	Chatterbox model object

Value

Chatterbox model with loaded weights

Examples

```
## Not run:
model <- chatterbox("cuda", load = FALSE)
model <- load_chatterbox(model)

## End(Not run)
```

load_chatterbox_turbo *Load Chatterbox Turbo model weights*

Description

Loads the turbo variant (GPT-2 backbone, MeanFlow decoder). Requires prior download via download_chatterbox_turbo_models.

Usage

```
load_chatterbox_turbo(model)
```

Arguments

model	Chatterbox model object (with turbo=TRUE)

Value

Chatterbox model with loaded weights

Examples

```
## Not run:
model <- chatterbox("cuda", turbo = TRUE, load = FALSE)
model <- load_chatterbox_turbo(model)

## End(Not run)
```

load_conformer_encoder_weights *Load Conformer Encoder weights*

Description

Load Conformer Encoder weights

Usage

```
load_conformer_encoder_weights(model, state_dict, prefix = "flow.encoder.")
```

Arguments

model	Conformer encoder module
state_dict	State dictionary
prefix	Key prefix (e.g., "flow.encoder.")

Value

The `model` module, with weights copied in from `state_dict`.

`load_llama_weights` *Load weights from safetensors into Llama model*

Description

Load weights from safetensors into Llama model

Usage

```
load_llama_weights(model, state_dict, prefix = "model.")
```

Arguments

`model`	LlamaModel instance
`state_dict`	Named list of tensors from safetensors
`prefix`	Prefix to strip from weight names (default: "model.")

Value

Model with loaded weights

`load_t3_turbo_weights` *Load T3 turbo weights from safetensors*

Description

Load T3 turbo weights from safetensors

Usage

```
load_t3_turbo_weights(model, state_dict)
```

Arguments

`model`	T3 turbo model
`state_dict`	Named list of tensors

Value

Model with loaded weights

load_t3_weights *Load T3 weights from safetensors*

Description

Load T3 weights from safetensors

Usage

```
load_t3_weights(model, state_dict)
```

Arguments

model	T3 model
state_dict	Named list of tensors

Value

Model with loaded weights

load_tokenizer *Load tokenizer from JSON file (internal)*

Description

Load tokenizer from JSON file (internal)

Usage

```
load_tokenizer(vocab_path)
```

Arguments

vocab_path	Path to tokenizer.json

Value

Tokenizer object (list)

load_voice_embedding *Load a voice embedding from disk*

Description

Load a voice embedding from disk

Usage

```
load_voice_embedding(path, device = "cpu")
```

Arguments

path	File written by save_voice_embedding
device	Device to load tensors to (default "cpu"; use the model's device, e.g. "cuda", for generation)

Value

A voice_embedding object

Examples

```
## Not run:
voice <- load_voice_embedding("narrator.voice", device = "cuda")
model <- chatterbox("cuda")
res <- generate(model, "Loaded a saved voice.", voice)

## End(Not run)
```

load_voice_encoder_weights *Load voice encoder weights from safetensors*

Description

Load voice encoder weights from safetensors

Usage

```
load_voice_encoder_weights(model, state_dict)
```

Arguments

model	Voice encoder model
state_dict	Named list of tensors

Value

Model with loaded weights

---

make_non_pad_mask_s3 *Create non-padding mask*

Description

Create non-padding mask

Usage

```
make_non_pad_mask_s3(lengths, max_len)
```

Arguments

lengths	Tensor of sequence lengths
max_len	Maximum sequence length

Value

Boolean mask tensor (TRUE for valid positions)

---

make_pad_mask *Create padding mask*

Description

Create padding mask

Usage

```
make_pad_mask(lengths, max_len = NULL)
```

Arguments

lengths	Sequence lengths
max_len	Maximum length

Value

Boolean mask (TRUE for padded positions)

mask_to_bias *Convert mask to attention bias*

Description

Convert mask to attention bias

Usage

```
mask_to_bias(mask, dtype)
```

Arguments

mask	Boolean mask
dtype	Target dtype

Value

Attention bias tensor

mish_activation *Mish activation*

Description

Mish activation

Usage

```
mish_activation()
```

Value

nn_module

models_available *Check if Models are Downloaded*

Description

Check if Models are Downloaded

Usage

```
models_available()
```

Value

TRUE if all model files exist locally

Examples

```
models_available()
```

normalize_loudness *Normalize audio to a target loudness*

Description

Applies a constant gain so the signal measures `target_lufs` integrated loudness. Mirrors Python chatterbox turbo's `norm_loudness()`: when the gain is non-finite or non-positive (e.g. silence), the input is returned unchanged.

Usage

```
normalize_loudness(samples, sample_rate, target_lufs = -27)
```

Arguments

`samples`	Numeric vector of mono audio samples.
`sample_rate`	Sample rate in Hz.
`target_lufs`	Target integrated loudness (default -27, the Python turbo conditioning default).

Value

Gain-adjusted samples.

Examples

```
samples <- sin(2 * pi * 440 * seq(0, 1, length.out = 48000))
norm <- normalize_loudness(samples, 48000, target_lufs = -23)
```

normalize_tts_text *Normalize text for TTS*

Description

The single normalization entry point. Applies, in order: the R-specific internal-caps mitigation (`normalize_internal_caps`), then punctuation normalization (`punc_norm`: whitespace collapse, first-letter capitalization, uncommon-punctuation rewrite, trailing period). `punc_norm` is the Python-parity piece; the caps step is R-only and can be turned off.

Usage

```
normalize_tts_text(text, caps = TRUE, punctuation = TRUE)
```

Arguments

`text`	Character scalar.
`caps`	Apply the internal-caps mitigation. Default TRUE.
`punctuation`	Apply punctuation normalization. Default TRUE.

Value

Normalized text.

Examples

```
normalize_tts_text("hello   world")
```

pad_audio_for_tokenizer *Pad audio to multiple of token rate*

Description

Pad audio to multiple of token rate

Usage

```
pad_audio_for_tokenizer(wav, sr)
```

Arguments

`wav`	Audio samples
`sr`	Sample rate

Value

Padded audio

perceiver_resampler *Perceiver resampler for conditioning compression*

Description

Perceiver resampler for conditioning compression

Usage

```
perceiver_resampler(num_query_tokens = 32, embed_dim = 1024, num_heads = 4)
```

Arguments

num_query_tokens	Number of query tokens (default 32)
embed_dim	Embedding dimension (default 1024)
num_heads	Number of attention heads (default 4)

Value

nn_module

positionwise_feedforward *Positionwise Feed Forward*

Description

Two-layer feed-forward network with SiLU activation.

Usage

```
positionwise_feedforward(n_feat = 512, n_ffn = 2048, dropout_rate = 0.1)
```

Arguments

n_feat	Input/output dimension
n_ffn	Hidden dimension
dropout_rate	Dropout rate

Value

nn_module

pre_lookahead_layer *Pre-Lookahead Layer*

Description

Two causal convolutions with residual connection for look-ahead.

Usage

```
pre_lookahead_layer(channels = 512, pre_lookahead_len = 3)
```

Arguments

channels	Number of channels
pre_lookahead_len	Look-ahead length (kernel size - 1 for conv1)

Value

nn_module

precompute_freqs_cis *Precompute rotary position embedding frequencies*

Description

Precompute rotary position embedding frequencies

Usage

```
precompute_freqs_cis(dim, end, theta = 10000)
```

Arguments

dim	Dimension (head_dim)
end	Maximum sequence length
theta	Base frequency

Value

Complex frequency tensor

print.chatterbox *Print method for chatterbox*

Description

Print method for chatterbox

Usage

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

Arguments

x	Chatterbox model
...	Ignored

Value

x, invisibly. Called for the side effect of printing a summary of the model to the console.

print.chatterbox_gc_options *Print method for chatterbox_gc_options*

Description

Print method for chatterbox_gc_options

Usage

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

Arguments

x	Object from chatterbox_gc_options
...	Ignored

Value

x, invisibly

print.voice_embedding *Print method for voice_embedding*

Description

Print method for voice_embedding

Usage

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

Arguments

x	Voice embedding
...	Ignored

Value

x, invisibly. Called for the side effect of printing the embedding's shape and sample rate to the console.

punc_norm *Normalize punctuation for TTS*

Description

Normalize punctuation for TTS

Usage

```
punc_norm(text)
```

Arguments

text	Input text

Value

Normalized text

quick_tts *Quick TTS - one-line text-to-speech*

Description

Loads model if needed and generates speech. Convenient for quick tests.

Usage

```
quick_tts(text, reference_audio, output_path = NULL, device = "cpu",
          autocast = NULL, turbo = FALSE)
```

Arguments

text	Text to synthesize
reference_audio	Path to reference audio file
output_path	Optional output file path. If NULL, returns audio data.
device	Device to use
autocast	Use mixed precision (float16) on CUDA (default TRUE on CUDA)
turbo	Logical. Use turbo architecture. Default FALSE.

Value

The generate result list (audio, sample_rate, ...). When output_path is set the audio is also written there (the list gains a path element) and the list is returned invisibly so the audio vector does not print.

Examples

```
## Not run:
quick_tts("Hello!", "reference_voice.wav", "out.wav")

## End(Not run)
```

read_audio *Read audio file*

Description

Read audio file

Usage

```
read_audio(path)
```

Arguments

path	Path to audio file (WAV or MP3 format)

Value

List with samples (numeric vector normalized to \[-1, 1\]) and sr (sample rate)

Examples

```
tmp <- file.path(tempdir(), "tone.wav")
write_audio(sin(2 * pi * 440 * seq(0, 1, length.out = 24000)), 24000, tmp)
a <- read_audio(tmp)
str(a)  # list(samples = ..., sr = ...)
```

---

reflection_pad1d *Reflection padding for 1D (nn_reflection_pad1d equivalent)*

Description

Reflection padding for 1D (nn_reflection_pad1d equivalent)

Usage

```
reflection_pad1d(padding)
```

Arguments

padding	Integer vector c(left, right) for padding

Value

nn_module

---

rel_position_attention *Relative Position Multi-Headed Attention*

Description

Multi-head attention with relative positional encodings.

Usage

```
rel_position_attention(n_head = 8, n_feat = 512, dropout_rate = 0.1)
```

Arguments

n_head	Number of attention heads
n_feat	Feature dimension
dropout_rate	Dropout rate

Value

nn_module

---

resample_audio *Resample audio*

Description

Resample audio

Usage

```
resample_audio(samples, from_sr, to_sr)
```

Arguments

samples	Numeric vector of audio samples
from_sr	Source sample rate
to_sr	Target sample rate

Value

Resampled audio samples

Examples

```
## Not run:
# Windowed-sinc resampling runs on torch, so it needs libtorch installed
tone <- sin(2 * pi * 440 * seq(0, 1, length.out = 24000))
tone_16k <- resample_audio(tone, 24000, 16000)

## End(Not run)
```

rotate_half *Rotate half of the tensor for RoPE*

Description

Rotate half of the tensor for RoPE

Usage

```
rotate_half(x)
```

Arguments

x	Input tensor

Value

Rotated tensor

s3_audio_encoder *S3 Audio Encoder V2*

Description

S3 Audio Encoder V2

Usage

```
s3_audio_encoder(n_mels, n_state, n_head, n_layer, stride = 2L)
```

Arguments

n_mels	Number of mel bins
n_state	Hidden dimension
n_head	Number of attention heads
n_layer	Number of transformer layers
stride	Convolution stride (default 2)

Value

nn_module

s3_log_mel_spectrogram

Compute log mel spectrogram for S3Tokenizer

Description

Compute log mel spectrogram for S3Tokenizer

Usage

```
s3_log_mel_spectrogram(audio, mel_filters, window, n_fft = 400, device = "cpu")
```

Arguments

audio	Audio tensor (batch, samples)
mel_filters	Pre-computed mel filterbank
window	Hann window
n_fft	FFT size (default 400)
device	Device

Value

Log mel spectrogram (batch, n_mels, time)

s3_multi_head_attention

Multi-Head Attention base module

Description

Multi-Head Attention base module

Usage

```
s3_multi_head_attention(n_state, n_head)
```

Arguments

n_state	Hidden dimension
n_head	Number of heads

Value

nn_module

s3_residual_attention_block *Residual attention block*

Description

Residual attention block

Usage

```
s3_residual_attention_block(n_state, n_head, kernel_size = 31L)
```

Arguments

n_state	Hidden dimension
n_head	Number of heads
kernel_size	FSMN kernel size

Value

nn_module

s3_tokenizer *S3Tokenizer V2 module*

Description

S3Tokenizer V2 module

Usage

```
s3_tokenizer(config = NULL)
```

Arguments

config	Configuration list (default from s3_tokenizer_config())

Value

nn_module

s3_tokenizer_config *S3Tokenizer model configuration*

Description

S3Tokenizer model configuration

Usage

```
s3_tokenizer_config(n_mels = 128, n_audio_state = 1280, n_audio_head = 20,
                    n_audio_layer = 6, n_codebook_size = 6561)
```

Arguments

n_mels	Number of mel bins (default 128)
n_audio_state	Hidden state dimension (default 1280)
n_audio_head	Number of attention heads (default 20)
n_audio_layer	Number of transformer layers (default 6)
n_codebook_size	Codebook size (default 6561 = 3^8)

Value

Configuration list

s3gen *S3Gen Token to Waveform*

Description

S3Gen Token to Waveform

Usage

```
s3gen(meanflow = FALSE)
```

Arguments

meanflow	Logical. Use mean-flow formulation. Default FALSE.

Value

nn_module

save_voice_embedding *Save a voice embedding to disk*

Description

Persists a prepared voice (the R analogue of Python 0.1.7's `Conditionals.save()`) so it can be reused across sessions without the reference audio or recomputation. Tensors are moved to CPU before saving; the format is `torch_save` (not compatible with Python's `.pt` conditionals).

Usage

```
save_voice_embedding(voice, path)
```

Arguments

`voice`	Voice embedding from `create_voice_embedding`
`path`	Output file path (suggested extension: .rds-like custom, e.g. "narrator.voice")

Value

`path`, invisibly

Examples

```
## Not run:
model <- chatterbox("cuda")
voice <- create_voice_embedding(model, "reference_voice.wav")
save_voice_embedding(voice, file.path(tempdir(), "narrator.voice"))

## End(Not run)
```

serve *Serve chatterbox over HTTP*

Description

Starts a blocking HTTP server that loads the chatterbox model once and answers OpenAI-compatible TTS requests. Intended as a drop-in replacement for the chatterbox TTS container: point an HTTP client (e.g. **tts.api**) at `http://<host>:<port>` and it serves the same endpoints.

Usage

```
serve(port = 7810L, device = "cuda", voices_dir = NULL, turbo = FALSE,
      timeout = 300L, max_body = 10L * 1024L^2, warmup = TRUE)
```

Arguments

port	Integer. TCP port to listen on. Default 7810.
device	Character. Torch device for the model ("cuda", "cpu", "mps").
voices_dir	Character. Directory of voice reference files. Defaults to the TTS_VOICES_DIR env var, then ~/.cornball/voices.
turbo	Logical. Serve the Chatterbox Turbo model.
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 10 MB.
warmup	Logical. Run one short synthesis at startup to trigger the one-time JIT tracing, so the first client request isn't slow. Default TRUE.

Details

Endpoints:

- GET /health - liveness probe, returns {"status":"ok"}.
- GET /v1/audio/voices - lists voice names in voices_dir.
- POST /v1/audio/speech - body {input, voice, response_format, exaggeration, cfg_weight, temperature}; returns the synthesized audio bytes. voice is a voice-library name (resolved against voices_dir) or a path to a reference audio file.

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("chatterbox.service", package = "chatterbox").

Value

Does not return normally; runs until interrupted.

Examples

```
## Not run:
# OpenAI-compatible TTS server on port 7810
serve(port = 7810L, device = "cuda")

## End(Not run)
```

sine_gen *Sine Generator*

Description

Generates sine waveforms from F0 for source-filter synthesis

Usage

```
sine_gen(sample_rate, harmonic_num = 0, sine_amp = 0.1, noise_std = 0.003,
         voiced_threshold = 0)
```

Arguments

sample_rate	Sampling rate in Hz
harmonic_num	Number of harmonics
sine_amp	Sine amplitude
noise_std	Noise standard deviation
voiced_threshold	F0 threshold for voiced/unvoiced

Value

nn_module

sinusoidal_pos_emb *Sinusoidal positional embedding for timesteps*

Description

Sinusoidal positional embedding for timesteps

Usage

```
sinusoidal_pos_emb(dim = 320L)
```

Arguments

dim	Output dimension

Value

nn_module

snake_activation *Snake activation function*

Description

Sine-based periodic activation: x + 1/a * sin^2(ax) Reference: https://arxiv.org/abs/2006.08195

Usage

```
snake_activation(in_features, alpha_trainable = TRUE, alpha_logscale = FALSE)
```

Arguments

in_features	Number of input channels
alpha_trainable	Whether alpha is trainable
alpha_logscale	Whether to use log scale for alpha

Value

nn_module

source_module_hn_nsf *Source Module for Neural Source Filter*

Description

Source Module for Neural Source Filter

Usage

```
source_module_hn_nsf(sample_rate, upsample_scale, harmonic_num = 0,
                     sine_amp = 0.1, add_noise_std = 0.003, voiced_threshold = 0)
```

Arguments

sample_rate	Sampling rate
upsample_scale	Upsampling factor
harmonic_num	Number of harmonics
sine_amp	Sine amplitude
add_noise_std	Noise std
voiced_threshold	Voiced threshold

Value

nn_module

statistics_pooling *Statistics pooling*

Description

Statistics pooling

Usage

```
statistics_pooling(x)
```

Arguments

x	Input tensor (batch, channels, time)

Value

Statistics tensor (batch, channels * 2)

t3_cond *Create T3 conditioning object*

Description

Create T3 conditioning object

Usage

```
t3_cond(speaker_emb, cond_prompt_speech_tokens = NULL,
        cond_prompt_speech_emb = NULL, emotion_adv = 0.5)
```

Arguments

speaker_emb	Speaker embedding tensor (B, 256)
cond_prompt_speech_tokens	Optional speech tokens for conditioning
cond_prompt_speech_emb	Optional pre-computed speech embeddings
emotion_adv	Emotion/exaggeration control (0-1)

Value

List representing T3Cond

t3_cond_enc *T3 conditioning encoder*

Description

T3 conditioning encoder

Usage

```
t3_cond_enc(config = NULL)
```

Arguments

config	T3 configuration

Value

nn_module

t3_cond_to_device *Move T3 conditioning to device*

Description

Move T3 conditioning to device

Usage

```
t3_cond_to_device(cond, device)
```

Arguments

cond	T3 conditioning object
device	Target device

Value

T3 conditioning on device

t3_config_english *Create T3 configuration (English-only)*

Description

Create T3 configuration (English-only)

Usage

```
t3_config_english()
```

Value

List with T3 configuration

t3_config_turbo *Create T3 turbo configuration (GPT-2 backbone)*

Description

Create T3 turbo configuration (GPT-2 backbone)

Usage

```
t3_config_turbo()
```

Value

List with T3 turbo configuration

t3_inference_traced *T3 inference with JIT tracing (optimized)*

Description

Uses jit_trace for ~8x faster per-token inference.

Usage

```
t3_inference_traced(model, cond, text_tokens, max_new_tokens = 1000,
                    temperature = 0.8, cfg_weight = 0.5, top_p = 1,
                  min_p = 0.05, repetition_penalty = 1.2, max_cache_len = 350L)
```

Arguments

model	T3 model
cond	Conditioning object
text_tokens	Text token tensor
max_new_tokens	Maximum tokens to generate
temperature	Sampling temperature
cfg_weight	Classifier-free guidance weight
top_p	Top-p sampling threshold
min_p	Min-p filtering threshold
repetition_penalty	Repetition penalty
max_cache_len	Maximum KV cache length

Value

Generated speech token tensor

---

t3_model *T3 Token-to-Token TTS model*

Description

T3 Token-to-Token TTS model

Usage

```
t3_model(config = NULL)
```

Arguments

config	T3 configuration

Value

nn_module

t3_model_turbo *T3 Token-to-Token TTS model (Turbo variant with GPT-2 backbone)*

Description

T3 Token-to-Token TTS model (Turbo variant with GPT-2 backbone)

Usage

```
t3_model_turbo(config = NULL)
```

Arguments

config	T3 turbo configuration

Value

nn_module

tdnn_layer *TDNN Layer*

Description

TDNN Layer

Usage

```
tdnn_layer(in_channels, out_channels, kernel_size, stride = 1, dilation = 1,
           padding = NULL)
```

Arguments

in_channels	Input channels
out_channels	Output channels
kernel_size	Kernel size
stride	Stride
dilation	Dilation
padding	Padding (default: computed from kernel_size and dilation)

Value

nn_module

timestep_embedding *Timestep embedding MLP*

Description

Timestep embedding MLP

Usage

```
timestep_embedding(in_channels = 320L, time_embed_dim = 1024L)
```

Arguments

in_channels	Input channels
time_embed_dim	Output dimension

Value

nn_module

tokenize_text *Encode text to token IDs using BPE*

Description

Mirrors the HF tokenizers pipeline used by the Python reference: added tokens ([SPACE], [laughter], [sigh], ...) are extracted first and map directly to their ids; BPE merges run on the remaining text, in priority order (first merge = highest priority).

Usage

```
tokenize_text(tokenizer, text)
```

Arguments

tokenizer	Tokenizer object
text	Input text

Value

Integer vector of token IDs

traceable_attention *Traceable attention module with pre-allocated KV cache*

Description

This module is designed to be traced with jit_trace. It uses: - Pre-allocated KV cache of fixed max size - Attention mask to indicate valid cache positions - Returns only output tensor (no lists/dicts)

Usage

```
traceable_attention(attn, max_cache_len = 300L)
```

Arguments

attn	Original llama_attention module
max_cache_len	Maximum cache length

Value

nn_module

traceable_decoder_layer *Traceable decoder layer with pre-allocated KV cache*

Description

Traceable decoder layer with pre-allocated KV cache

Usage

```
traceable_decoder_layer(layer, max_cache_len = 300L)
```

Arguments

layer	Original llama_decoder_layer
max_cache_len	Maximum cache length

Value

nn_module

traceable_kv_projector

Traceable K/V projection module

Description

Computes K and V projections with RoPE for a single layer. Returns concatenated K and V for easy unpacking.

Usage

```
traceable_kv_projector(layer)
```

Arguments

layer	Original llama_decoder_layer

Value

nn_module

traceable_transformer_cached

Traceable transformer for cached inference

Description

This wraps the full Llama model for traced cached inference. Uses pre-allocated KV cache for all layers.

Usage

```
traceable_transformer_cached(tfmr, max_cache_len = 300L)
```

Arguments

tfmr	Original llama_model
max_cache_len	Maximum cache length

Value

nn_module

traceable_transformer_first *Traceable transformer for first token (no cache)*

Description

Traceable transformer for first token (no cache)

Usage

```
traceable_transformer_first(tfmr)
```

Arguments

tfmr	Original llama_model

Value

nn_module

transit_layer *Transit layer (channel reduction)*

Description

Transit layer (channel reduction)

Usage

```
transit_layer(in_channels, out_channels, bias = FALSE)
```

Arguments

in_channels	Input channels
out_channels	Output channels
bias	Whether to use bias (default FALSE)

Value

nn_module

transpose_layer *Transpose layer for use in sequential*

Description

Transpose layer for use in sequential

Value

nn_module

tts_chunked *Generate speech for long text (the long-form policy layer)*

Description

Splits at sentence boundaries (oversized sentences subdivided at commas, then word-split as a last resort), resolves the voice once, and runs T3 on every chunk first so batching uses ACTUAL speech-token lengths rather than a character estimate. Chunks are then bucketed by their real length and synthesized within a per-card batch cap (sized from VRAM): a group of one takes the fast traced-CFM path, a group of several runs as one eager batched S3Gen solve. Audio is stitched in original order; garbage is collected at each batch boundary (see `chatterbox_gc_options`). Turbo has no batched path and is synthesized serially.

Usage

```
tts_chunked(model, text, voice, chunk_size = 200, max_batch = NULL, ...)
```

Arguments

`model`	Chatterbox model
`text`	Text to synthesize
`voice`	Voice embedding or path to reference audio (resolved once)
`chunk_size`	Maximum characters per chunk (default 200)
`max_batch`	Maximum chunks per batched solve. Default NULL: sized per card from VRAM. Set an integer to override.
`...`	Synthesis arguments forwarded to the T3 and S3Gen stages, as in `generate` (exaggeration, cfg_weight, temperature, backend, traced, normalize_text, max_new_tokens, ...)

Value

List with audio and sample_rate

Examples

```
## Not run:
model <- chatterbox("cuda")
res <- tts_chunked(model, long_text, "reference_voice.wav")
write_audio(res$audio, res$sample_rate, "long.wav")

## End(Not run)
```

`tts_to_file` *Generate speech and save to file*

Description

Thin convenience wrapper over `generate` with `output_path` set, kept for the file-summary return shape. New code can call `generate(..., output_path = path)` directly.

Usage

```
tts_to_file(model, text, voice, output_path, ...)
```

Arguments

`model`	Chatterbox model
`text`	Text to synthesize
`voice`	Voice embedding or path to reference audio
`output_path`	Output file path (WAV format)
`...`	Additional arguments passed to generate()

Value

Invisibly returns a list with elements: `path`, `eos_found`, `n_tokens`, `audio_sec`. When iterating over many texts, collect these into a data.frame to identify which inputs failed (`eos_found = FALSE`) and need reprocessing.

Examples

```
## Not run:
model <- chatterbox("cuda")
tts_to_file(model, "Hello world!", "reference_voice.wav", "out.wav")

## End(Not run)
```

turbo_models_available *Check if Turbo Models are Downloaded*

Description

Check if Turbo Models are Downloaded

Usage

```
turbo_models_available()
```

Value

TRUE if all turbo model files exist locally

Examples

```
turbo_models_available()
```

update_kv_cache *Update KV cache with new K/V values*

Description

Update KV cache with new K/V values

Usage

```
update_kv_cache(cache, layer_idx, new_k, new_v, position)
```

Arguments

cache	Cache list from create_kv_cache
layer_idx	Layer index (1-indexed)
new_k	New key tensor (batch, heads, 1, head_dim)
new_v	New value tensor (batch, heads, 1, head_dim)
position	Current position (0-indexed)

Value

The cache list, invisibly, with the new K/V written in place at position.

update_valid_mask *Update valid mask to include new position*

Description

Update valid mask to include new position

Usage

```
update_valid_mask(cache, position)
```

Arguments

cache	Cache list from create_kv_cache
position	Current position (0-indexed)

Value

The cache list, invisibly, with position marked valid in the attention mask.

upsample_1d *Upsample 1D*

Description

2x upsampling using interpolation + convolution.

Usage

```
upsample_1d(channels = 512, stride = 2L)
```

Arguments

channels	Number of channels
stride	Upsample factor

Value

nn_module

upsample_conformer_encoder

Upsample Conformer Encoder

Description

Upsample Conformer Encoder

Usage

```
upsample_conformer_encoder(input_size = 512, output_size = 512, num_blocks = 6)
```

Arguments

input_size	Input dimension
output_size	Output dimension
num_blocks	Number of conformer blocks

Value

nn_module

upsample_conformer_encoder_full

Upsample Conformer Encoder

Description

Full conformer encoder matching Python UpsampleConformerEncoder.

Usage

```
upsample_conformer_encoder_full(input_size = 512, output_size = 512,
                                num_blocks = 6, num_up_blocks = 4, n_head = 8,
                                n_ffn = 2048, dropout_rate = 0.1,
                                pre_lookahead_len = 3)
```

Arguments

input_size	Input dimension
output_size	Output dimension
num_blocks	Number of conformer blocks before upsample
num_up_blocks	Number of conformer blocks after upsample
n_head	Number of attention heads

n_ffn	Feed-forward hidden dimension
dropout_rate	Dropout rate
pre_lookahead_len	Look-ahead length

Value

nn_module

---

voice_convert *Convert speech to a target voice*

Description

Re-synthesizes audio so the same words and prosody come out in the target voice (Python chatterbox's ChatterboxVC). No text or T3 generation is involved: the source speech is tokenized directly (25 tokens/s) and S3Gen renders the tokens with the target speaker's conditioning, so the result follows the source's timing.

Usage

```
voice_convert(model, audio, voice, sample_rate = NULL)
```

Arguments

model	Loaded chatterbox model (standard, not turbo)
audio	Source speech (file path, numeric vector, or torch tensor)
voice	Target voice: a voice_embedding from create_voice_embedding (or load_voice_embedding), or a path to reference audio
sample_rate	Sample rate of audio (if not a file)

Value

List with audio (numeric vector), sample_rate (24000), and audio_sec, like generate

Examples

```
## Not run:
model <- chatterbox("cuda")
res <- voice_convert(model, "source_speech.wav", "target_voice.wav")
write_audio(res$audio, res$sample_rate, "converted.wav")

## End(Not run)
```

voice_encoder *Voice encoder module*

Description

Voice encoder module

Usage

```
voice_encoder(config = NULL)
```

Arguments

config	Voice encoder configuration

Value

nn_module

voice_encoder_config *Voice encoder configuration*

Description

Voice encoder configuration

Usage

```
voice_encoder_config()
```

Value

List with configuration parameters

`write_audio` *Write audio file*

Description

Write audio file

Usage

```
write_audio(samples, sr, path)
```

Arguments

`samples`	Numeric vector of audio samples (normalized to \[-1, 1\])
`sr`	Sample rate
`path`	Output path (WAV format)

Value

The output `path`, invisibly. Called for the side effect of writing a WAV file.

Examples

```
tmp <- file.path(tempdir(), "tone.wav")
write_audio(sin(2 * pi * 440 * seq(0, 1, length.out = 24000)), 24000, tmp)
```

Index