Bark 🐶

↑ Back to top

Bark is a multi-lingual TTS model created by Suno-AI. It can generate conversational speech as well as music and sound effects. It is architecturally very similar to Google's AudioLM. For more information, please refer to the Suno-AI's repo.

Acknowledgements

- 👑 Suno-AI for training and open-sourcing this model.
- 👑 gitmylo for finding the solution to the semantic token generation for voice clones and finetunes.
- 👑 serp-ai for controlled voice cloning.

Example Use

```
text = "Hello, my name is Manmay , how are you?"

from TTS.tts.configs.bark_config import BarkConfig
from TTS.tts.models.bark import Bark

config = BarkConfig()
model = Bark.init_from_config(config)
model.load_checkpoint(config, checkpoint_dir="path/to/model/dir/", eval=True)

# with random speaker
output_dict = model.synthesize(text, config, speaker_id="random", voice_dirs=None)

# cloning a speaker.
# It assumes that you have a speaker file in `bark_voices/speaker_n/speaker.wav` or `bark_voices/
output_dict = model.synthesize(text, config, speaker_id="ljspeech", voice_dirs="bark_voices/")
```

Using 🐸TTS API:

v: dev

```
from TTS.api import TTS

# Load the model to GPU
# Bark is really slow on CPU, so we recom[illegible]U.
tts = TTS("tts_models/multilingual/multi-dataset/bark", gpu=True)


# Cloning a new speaker
# This expects to find a mp3 or wav file like `bark_voices/new_speaker/speaker.wav`
# It computes the cloning values and stores in `bark_voices/new_speaker/speaker.npz`
tts.tts_to_file(text="Hello, my name is Manmay , how are you?",
                file_path="output.wav",
                voice_dir="bark_voices/",
                speaker="ljspeech")


# When you run it again it uses the stored values to generate the voice.
tts.tts_to_file(text="Hello, my name is Manmay , how are you?",
                file_path="output.wav",
                voice_dir="bark_voices/",
                speaker="ljspeech")


# random speaker
tts = TTS("tts_models/multilingual/multi-dataset/bark", gpu=True)
tts.tts_to_file("hello world", file_path="out.wav")
```

↑ Back to top

Using 🐸TTS Command line:

```
# cloning the `ljspeech` voice
tts --model_name  tts_models/multilingual/multi-dataset/bark \
--text "This is an example." \
--out_path "output.wav" \
--voice_dir bark_voices/ \
--speaker_idx "ljspeech" \
--progress_bar True

# Random voice generation
tts --model_name  tts_models/multilingual/multi-dataset/bark \
--text "This is an example." \
--out_path "output.wav" \
--progress_bar True
```

v: dev

Important resources & papers

↑ Back to top

- Original Repo: https://github.com/suno-ai/bark
- Cloning implementation: https://github.com/serp-ai/bark-with-voice-clone
- AudioLM: https://arxiv.org/abs/2209.03143

v: dev ▾

BarkConfig

↑ Back to top

```
class TTS.tts.configs.bark_config.BarkConfig(output_path='output', logger_uri=None,
    run_name='run', project_name=None, run_description='🐸Coqui trainer run.',
    print_step=25, plot_step=100, model_param_stats=False, wandb_entity=None,
    dashboard_logger='tensorboard', save_on_interrupt=True, log_model_step=None,
    save_step=10000, save_n_checkpoints=5, save_checkpoints=True, save_all_best=False,
    save_best_after=10000, target_loss=None, print_eval=False, test_delay_epochs=0,
    run_eval=True, run_eval_steps=None, distributed_backend='nccl',
    distributed_url='tcp://localhost:54321', mixed_precision=False, precision='fp16',
    epochs=1000, batch_size=32, eval_batch_size=16, grad_clip=0.0,
    scheduler_after_epoch=True, lr=0.001, optimizer='radam', optimizer_params=None,
    lr_scheduler=None, lr_scheduler_params=<factory>, use_grad_scaler=False,
    allow_tf32=False, cudnn_enable=True, cudnn_deterministic=False, cudnn_benchmark=False,
    training_seed=54321, model='bark', num_loader_workers=0, num_eval_loader_workers=0,
    use_noise_augment=False, audio=<factory>, use_phonemes=False, phonemizer=None,
    phoneme_language=None, compute_input_seq_cache=False, text_cleaner=None,
    enable_eos_bos_chars=False, test_sentences_file='', phoneme_cache_path=None,
    characters=None, add_blank=False, batch_group_size=0, loss_masking=None,
    min_audio_len=1, max_audio_len=inf, min_text_len=1, max_text_len=inf,
    compute_f0=False, compute_energy=False, compute_linear_spec=False,
    precompute_num_workers=0, start_by_longest=False, shuffle=False, drop_last=False,
    datasets=<factory>, test_sentences=<factory>, eval_split_max_size=None,
    eval_split_size=0.01, use_speaker_weighted_sampler=False,
    speaker_weighted_sampler_alpha=1.0, use_language_weighted_sampler=False,
    language_weighted_sampler_alpha=1.0, use_length_weighted_sampler=False,
    length_weighted_sampler_alpha=1.0, num_chars=0, semantic_config=<factory>,
    fine_config=<factory>, coarse_config=<factory>, CONTEXT_WINDOW_SIZE=1024,
    SEMANTIC_RATE_HZ=49.9, SEMANTIC_VOCAB_SIZE=10000, CODEBOOK_SIZE=1024,
    N_COARSE_CODEBOOKS=2, N_FINE_CODEBOOKS=8, COARSE_RATE_HZ=75, SAMPLE_RATE=24000,
    USE_SMALLER_MODELS=False, TEXT_ENCODING_OFFSET=10048, SEMANTIC_PAD_TOKEN=10000,
    TEXT_PAD_TOKEN=129595, SEMANTIC_INFER_TOKEN=129599, COARSE_SEMANTIC_PAD_TOKEN=12048,
    COARSE_INFER_TOKEN=12050, REMOTE_MODEL_PATHS=None, LOCAL_MODEL_PATHS=None,
    SMALL_REMOTE_MODEL_PATHS=None, CACHE_DIR='/home/docs/.local/share/tts/suno/bark_v0',
    DEF_SPEAKER_DIR='/home/docs/.local/share/tts/bark_v0/speakers')            [source]
```

v: dev ▾

Bark TTS configuration

PARAMETERS:

- **model** (*str*) – model name that reg ↑ Back to top del.
- **audio** (*BarkAudioConfig*) – audio configuration. Defaults to BarkAudioConfig().
- **num_chars** (*int*) – number of characters in the alphabet. Defaults to 0.
- **semantic_config** (*GPTConfig*) – semantic configuration. Defaults to GPTConfig().
- **fine_config** (*FineGPTConfig*) – fine configuration. Defaults to FineGPTConfig().
- **coarse_config** (*GPTConfig*) – coarse configuration. Defaults to GPTConfig().
- **CONTEXT_WINDOW_SIZE** (*int*) – GPT context window size. Defaults to 1024.
- **SEMANTIC_RATE_HZ** (*float*) – semantic tokens rate in Hz. Defaults to 49.9.
- **SEMANTIC_VOCAB_SIZE** (*int*) – semantic vocabulary size. Defaults to 10_000.
- **CODEBOOK_SIZE** (*int*) – encodec codebook size. Defaults to 1024.
- **N_COARSE_CODEBOOKS** (*int*) – number of coarse codebooks. Defaults to 2.
- **N_FINE_CODEBOOKS** (*int*) – number of fine codebooks. Defaults to 8.
- **COARSE_RATE_HZ** (*int*) – coarse tokens rate in Hz. Defaults to 75.
- **SAMPLE_RATE** (*int*) – sample rate. Defaults to 24_000.
- **USE_SMALLER_MODELS** (*bool*) – use smaller models. Defaults to False.
- **TEXT_ENCODING_OFFSET** (*int*) – text encoding offset. Defaults to 10_048.
- **SEMANTIC_PAD_TOKEN** (*int*) – semantic pad token. Defaults to 10_000.
- **TEXT_PAD_TOKEN** (*[type]*) – text pad token. Defaults to 10_048.
- **TEXT_EOS_TOKEN** (*[type]*) – text end of sentence token. Defaults to 10_049.
- **TEXT_SOS_TOKEN** (*[type]*) – text start of sentence token. Defaults to 10_050.
- **SEMANTIC_INFER_TOKEN** (*int*) – semantic infer token. Defaults to 10_051.
- **COARSE_SEMANTIC_PAD_TOKEN** (*int*) – coarse semantic pad token. Defaults to 12_048.
- **COARSE_INFER_TOKEN** (*int*) – coarse infer token. Defaults to 12_050.
- **REMOTE_BASE_URL** (*[type]*) – remote base url. Defaults to "https://huggingface.co/erogol/bark/tree".
- **REMOTE_MODEL_PATHS** (*Dict*) – remote model paths. Defaults to None.
- **LOCAL_MODEL_PATHS** (*Dict*) – local model paths. Defaults to None.
- **SMALL_REMOTE_MODEL_PATHS** (*Dict*) – small remote model paths. Defaults to None.
- **CACHE_DIR** (*str*) – local cache directory. Defaults to get_user_data_dir().
- **DEF_SPEAKER_DIR** (*str*) – default speaker directory to stoke speaker values for voice cloning. Defaults to get_user_data_dir().

v: dev

BarkArgs

↑ Back to top

Bark Model

```
class TTS.tts.models.bark.Bark(config, tokenizer=BertTokenizer(name_or_path='bert-base-
    multilingual-cased', vocab_size=119547, model_max_length=512, is_fast=False,
    padding_side='right', truncation_side='right', special_tokens={'unk_token': '[UNK]',
```

v: dev

```
'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token':
'[MASK]'}, clean_up_tokenization_spaces=True))
```
[source]

```
generate_audio(text, history_prompt              temp=0.7, waveform_temp=0.7, base=None,
    allow_early_stop=True, **kwargs)
```
[source]

↑ Back to top

Generate audio array from input text.

PARAMETERS:

- **text** – text to be turned into audio
- **history_prompt** – history choice for audio cloning
- **text_temp** – generation temperature (1.0 more diverse, 0.0 more conservative)
- **waveform_temp** – generation temperature (1.0 more diverse, 0.0 more conservative)

RETURNS:

numpy audio array at sample frequency 24khz

```
generate_voice(audio, speaker_id, voice_dir)
```
[source]

Generate a voice from the given audio and text.

PARAMETERS:

- **audio** (*str*) – Path to the audio file.
- **speaker_id** (*str*) – Speaker name.
- **voice_dir** (*str*) – Path to the directory to save the generate voice.

```
load_checkpoint(config, checkpoint_dir, text_model_path=None, coarse_model_path=None,
    fine_model_path=None, eval=False, strict=True, **kwargs)
```
[source]

Load a model checkpoints from a directory. This model is with multiple checkpoint files and it expects to have all the files to be under the given *checkpoint_dir* with the rigth names. If

v: dev ▾

eval is True, set the model to eval mode.

PARAMETERS:

↑ Back to top

- **config** (*TortoiseConfig*) – The
- **checkpoint_dir** (*str*) – The directory where the checkpoints are stored.
- **ar_checkpoint_path** (*str, optional*) – The path to the autoregressive checkpoint. Defaults to None.
- **diff_checkpoint_path** (*str, optional*) – The path to the diffusion checkpoint. Defaults to None.
- **clvp_checkpoint_path** (*str, optional*) – The path to the CLVP checkpoint. Defaults to None.
- **vocoder_checkpoint_path** (*str, optional*) – The path to the vocoder checkpoint. Defaults to None.
- **eval** (*bool, optional*) – Whether to set the model to eval mode. Defaults to False.
- **strict** (*bool, optional*) – Whether to load the model strictly. Defaults to True.

`semantic_to_waveform(semantic_tokens, history_prompt=None, temp=0.7, base=None)[source]`

Generate audio array from semantic input.

PARAMETERS:

- **semantic_tokens** – semantic token output from *text_to_semantic*
- **history_prompt** – history choice for audio cloning
- **temp** – generation temperature (1.0 more diverse, 0.0 more conservative)

RETURNS:

numpy audio array at sample frequency 24khz

`synthesize(text, config, speaker_id='random', voice_dirs=None, **kwargs)` `[source]`

v: dev

Synthesize speech with the given input text.

↑ Back to top

PARAMETERS:

- **text** (*str*) – Input text.
- **config** (*BarkConfig*) – Config with inference parameters.
- **speaker_id** (*str*) – One of the available speaker names. If *random*, it generates a random speaker.
- **speaker_wav** (*str*) – Path to the speaker audio file for cloning a new voice. It is cloned and saved in *voice_dirs* with the name *speaker_id*. Defaults to None.
- **voice_dirs** (*List[str]*) – List of paths that host reference audio files for speakers. Defaults to None.
- ****kwargs** – Model specific inference settings used by *generate_audio()* and `TTS.tts.layers.bark.inference_funcs.generate_text_semantic().

RETURNS:

A dictionary of the output values with *wav* as output waveform, *deterministic_seed* as seed used at inference, *text_input* as text token IDs after tokenizer, *voice_samples* as samples used for cloning, *conditioning_latents* as latents used at inference.

```
text_to_semantic(text, history_prompt=None, temp=0.7, base=None,
    allow_early_stop=True, **kwargs)
```

[source]

Generate semantic array from text.

PARAMETERS:

- **text** – text to be turned into audio
- **history_prompt** – history choice for audio cloning
- **temp** – generation temperature (1.0 more diverse, 0.0 more conservative)

RETURNS:

numpy semantic array to be fed into *semantic_to_waveform*

v: dev