XTTS-v2: Single-Speaker Fine-Tuning

Roberto Caamano, Giuseppe Di Roberto

***Abstract*—This paper presents the process and findings of fine-tuning XTTS-v2, a multilingual and multispeaker text-to-speech model, on single-speaker datasets. XTTS-v2 integrates a GPT-based decoder, discrete variational autoencoder (VQ-VAE), and HiFi-GAN vocoder to generate high-quality speech from text and reference audio inputs. By constructing custom datasets using both statistical and sentence-level chunking and enabling configuration strategies such as conditioning resampling and masked ground truth prompting, the model was used to clone speaker identity and prosody with minimal training data.**

Experiments were conducted across both public and personal voice recordings, ranging from 45 minutes to 4 hours in length. Results highlighted the model's sensitivity to dataset size and quality, with smaller datasets requiring many more training epochs. Subjective checkpoint evaluation proved more reliable than numerical loss metrics for measuring audio quality and speaker similarity. A user-facing demo was developed using Streamlit to simplify inference and provide real-time customization of output speech.

Video Presentation: https://youtu.be/3KKw0uV_KRs

GitHub Repository: https://github.com/Sliverwall/DS677_852_XTTS_V2_SINGLE_SPEAKER_FINE_TUNING.git

I. Introduction

Text-to-speech (TTS) synthesis has advanced in recent years, and is now capable of producing natural, human-like speech. Modern TTS use deep learning architectures, such as sequence-to-sequence and transformer models, to learn translation from text to waveform. Among these, the XTTS-v2 is a model that quantizes both text and audio into discrete code sequences, then use an autoregressive transformer to learn the mapping between them. XTTS-v2 [1] builds on much of the work of Tortoise [2], which is another GPT based TTS model. In version 2, XTTS introduces a PerceiverResampler for fixed-size conditioning, and adding an additional latent vector used to improve stability with varying audio input lengths and aid the model to better encode pitch and rhythm. In this work we set out to explore the fine-tuning behavior of XTTS-v2 on single-speaker datasets of varying sizes. Our objectives were to determine optimal training schedules (epochs vs. dataset size), and verify if the model could be fine-tuned to achieve high-quality voice cloning on a single-speaker. We conducted a series of fine-tuning runs on both large (2–4 hour) and small (0.5–1.5 hour) datasets, assessing number of epochs effect on the synthesized voice outputs. We evaluated each run through listening tests, selecting the best checkpoint based on perceptual quality rather than loss metrics. Our results show that larger datasets require fewer epochs to avoid over-fitting, while smaller datasets benefit from extended training.

II. Related Work

The development of XTTS-v2 draws upon earlier advancements in neural text-to-speech systems, particularly Tortoise TTS and its predecessor XTTS-v1.

Tortoise TTS combined a GPT-style decoder with a diffusion-based vocoder and a reference encoder to generate highly realistic speech. While Tortoise TTS achieves impressive naturalness in synthesized speech, it has notable limitations. The diffusion vocoder introduced slow inference speeds, and the system often required large amounts of high-quality data to fine-tune effectively.

XTTS-v1, developed by Coqui, builds upon the Tortoise architecture by replacing the diffusion vocoder with a faster HiFi-GAN decoder and introducing a discrete variational autoencoder (dVAE) to tokenize audio. A GPT-2 style decoder, conditioned on both text tokens and speaker embeddings, predicts these tokens. XTTS-v1 also supports multilingual voice cloning and introduced dual language model heads for computing joint loss over both text and audio. Despite these advances, it still suffered from instability on long conditioning sequences, overfitting on small datasets, and limited expressiveness.

XTTS-v2 introduces significant architectural and functional improvements over its predecessor. Most notably, it introduces a Perceiver Resampler, which compresses conditioning inputs into a fixed-length latent representation. It also implements a masked ground truth prompting strategy to improve prosody and early token alignment. XTTS-v2 also standardizes audio output at a 24 kHz sample rate for improved voice fidelity and expands support for a total of 17 languages. Together, these enhancements make XTTS-v2 significantly more expressive, generalizable, and fine-tune friendly than its predecessors.

III. XTTS-v2 Model Architecture

The XTTS-v2 model transforms text and short audio prompts into a final waveform through various components. The text is first tokenized into subwords via BPE while a short training sample audio clip (which will be used as the target here) is converted into discrete codes by the VQ-VAE. Parallel to this, the reference audio is fed into the Conditioning Encoder, and the resulting embeddings are compressed to a fixed size by a Perceiver Resampler. A single GPT-2 then takes both the text token sequence and the resampled audio embeddings to predict a sequence of discrete audio codes. Also outputted by the GPT-2 are primary and auxiliary language modeling heads. Finally, a HiFi-GAN decodes those predicted codes back into a high-fidelity waveform, with an optional Speaker Encoder ensuring the synthesized voice retains its target speaker identity. When the model is trained via a single-speaker (i.e. there is only one speaker the model must perform as), the Speaker Encoder is not used.

A. *Byte-Pair Encoding*

Byte-Pair Encoding (BPE) [3] starts with a large set of individual characters and iteratively merges the most frequent

adjacent pairs into new "subword" units, building a compact vocabulary. This allows the tokenizer to represent common words as single tokens while still decomposing rare or unseen words into meaningful subword pieces. This keeps sequence lengths manageable for the GPT-2. During Fine-tuning the BPE is not adjusted. The BPE information is stored in a vocab.json file in the XTTS project.

B. Vector Quantised-Variational AutoEncoder (VQ-VAE)

The VQ-VAE encodes frames of the input audio through a convolutional encoder, producing continuous latent vectors. Each latent vector is then quantized to the nearest entry (nearest-neighbor lookup) in a fixed codebook of learned embedding vectors, giving a discrete code index for each frame. During training, a decoder reconstructs the original audio features from those discrete codes. The result is a sequence of audio tokens that represent prosody, rhythm, and phonetic content.

C. Conditioning Encoder

The Conditioning Encoder takes the VQ-VAE's discrete audio tokens from the speaker reference audio as input and applies convolutional layers to extract higher-level embeddings. The output is a fixed-length embedding per example in the batch that encodes the speaker reference audio's prosody and speaker features. By separating prosody and speaker content, the model can have better control over both what is said and how it is said.

D. Perceiver Resampler

The Perceiver Resampler takes as input the conditioning embeddings from the Conditioning Encoder and applies cross-attention then projects them into a fixed-size latent representation independent of the input length. It uses learned queries to focus on the most informative aspects of the prosody and speaker signals. The output is the final conditional embedding used by the GPT-2. This additional conditioning embedding was not used in the original XTTS-v1 model, and is an improvement found in XTTS-v2.

E. GPT

The GPT module in XTTS-v2 is a causal Transformer decoder based on the GPT-2 architecture [9]. It serves as the core generative component of the model and is responsible for predicting the sequence of discrete audio tokens that will ultimately be synthesized into waveform audio. Unlike standard autoregressive language models, this GPT is adapted to handle both linguistic content and acoustic structure by conditioning on both text and speaker prosody.

During training, the input to the GPT consists of three main components: a sequence of text tokens generated by the BPE tokenizer, a sequence of ground truth audio tokens obtained from the VQ-VAE, and a set of conditioning latents extracted by the Perceiver Resampler.

The text and ground truth audio tokens are embedded into continuous vector representations via learned embedding layers and are each augmented with their respective learned positional encodings. The conditioning latents, derived from the speaker reference audio, are then prepended to the embedded token sequence. This enables the GPT to condition its predictions on the reference speaker's audio throughout the generation process.

Internally, the GPT consists of stacked Transformer decoder blocks, each composed of the following components:

*1) **Multi-Head Self-Attention**:* The GPT uses causal self-attention, meaning each token attends only to all previous tokens including the conditioning latents. This allows the model to learn rich, context-aware representations. Multi-Head Self-Attention splits the input into multiple attention heads, each attending to different types of information simultaneously.

*2) **Feed-Forward Network (FFN)**:* A two-layer FFN follows each attention block. It projects the hidden dimension to a larger intermediate size, applies non-linearity, and projects back to the original dimension. This helps the model learn non-linear transformations and abstract patterns.

*3) **Layer Normalization and Residual Connections**:* Each attention and feed-forward block is wrapped with residual connections and layer normalization, ensuring stable training and allowing gradients to flow effectively across deep layers.

In the default XTTS-v2 configuration, the GPT consists of 30 Transformer decoder layers, each with a hidden size of 1024 and 16 attention heads, providing sufficient capacity to model both linguistic and acoustic dependencies across long sequences.

During training, the GPT uses teacher forcing: both text and ground truth tokens are fed as input, and the model is trained to predict the next token at each time step. To stabilize generation, XTTS-v2 uses a masked ground truth prompt strategy in which a configurable number of initial ground truth audio tokens are fed into the model as context but are excluded from the loss computation. This avoids penalizing the model for mismatches in early generation where it may not have enough contextual understanding of the input sequence. Additionally, this approach aids in aligning prosody and rhythm early in the audio token stream.

F. Dual LM Heads

The GPT module includes two distinct output projection heads: one for predicting text tokens and another for audio tokens. These are referred to as dual language modeling (LM) heads, and are used to apply supervision separately on the text and audio portions of the sequence during training. However, the text token prediction is not used in fine-tuning, so the text head is functionally irrelevant.

Each head is implemented as a simple linear projection from the model's hidden size (typically 1024) to its respective vocabulary size. The text head maps to the subword vocabulary defined by the BPE tokenizer, while the mel head maps to the discrete audio token vocabulary defined by the VQ-VAE codebook (typically 1026 entries including special tokens). During training, every token in the sequence is passed through both heads, but only the relevant head contributes to the loss depending on whether the token is part of the text or audio segment.

A cross-entropy loss is applied separately to the predictions from each head. Crucially, the two loss components are weighted unequally to prioritize accurate audio generation. By default, the audio loss is weighted as 1.0, while the text loss is weighted as 0.01. This reflects that during inference, the text tokens are already provided, and the model's primary goal is to generate their corresponding audio tokens.

G. *HiFi-GAN Decoder*

The final stage in the XTTS-v2 pipeline is a neural vocoder based on HiFi-GAN [8], which synthesizes waveform audio from the decoded acoustic representations produced by the GPT. This vocoder includes both a Speaker Encoder and a HiFi-GAN Generator. Together, they ensure the output speech matches not only the textual content, but also aligns with the intended speaker characteristics. After pre-training, the decoder can take as input a sequence of acoustic tokens, and optionally a speaker reference audio, and synthesize high-quality speech in the target voice.

*1) **Speaker Encoder**:* The speaker encoder is a ResNet-based model that extracts speaker-specific embeddings from the reference audio. These embeddings represent speaker-specific features, such as pitch, tone, and speaking style. The output is a fixed 512-dimensional vector that is broadcast and added to feature maps at each upsampling layer, allowing the generator to synthesize waveforms consistent with the reference speaker's identity.

This component is essential during inference when synthesizing audio for new speakers and ensures speaker identity is preserved across generated utterances. It is also used during pre-training to compute the Speaker Consistency Loss (SCL), which forces the speaker embedding extracted from the generated audio to match that of the reference audio.

*2) **HiFi-GAN Generator**:* The generator module is responsible for converting intermediate acoustic representations into raw waveform audio. It takes as input the latent features produced by the GPT, as well as the speaker embedding from the speaker encoder.

Internally, the generator consists of a series of transposed convolutional layers that progressively upsample the input features. Each convolutional layer is followed by residual blocks that refine the output and support in capturing complex audio patterns.

During pre-training, the generator is trained using an adversarial loss, spectral reconstruction loss, and waveform reconstruction loss. Adversarial loss encourages the generator to produce audio that is indistinguishable from real human speech. This is achieved through the use of multi-scale discriminators, which assess the realism of generated waveforms at different temporal resolutions, capturing both fine-grained details and long-range structure.

In parallel, a spectral reconstruction loss is applied by comparing the mel spectrograms of the generated and ground truth waveforms. This ensures that the generator accurately reproduces the frequency content and timbral qualities of the original audio. Additionally, a waveform reconstruction loss operates directly in the time domain, penalizing sample-level deviations between the predicted and true audio signals.

Once pre-training is complete, the HiFi-GAN generator's weights are frozen during GPT fine-tuning. This ensures that the waveform synthesis process remains stable while the GPT learns to produce better audio token sequences.

IV. Dataset

The datasets used for fine-tuning XTTS-v2 were sourced from both publicly available celebrity recordings and personally recorded voice samples. To create training datasets from these audio sources, two data generation pipelines were used: one based on statistical chunking and the other on sentence-level segmentation. Both pipelines produced audio–text pairs in LJSpeech format (`<filename>|<text>|<normalized text>`) compatible with XTTS.

The celebrity datasets included speech samples from Tom Hanks [4] (approximately 4 hours) and Benedict Cumberbatch [5] (approximately 2 hours), extracted from YouTube audio books. For the personal voice datasets, original recordings were captured locally and concatenated if needed, ranging between 45 minutes to 1.5 hours of speech. All audio files were normalized to -20 dBFS to ensure consistent loudness and resampled to the 24 kHz, mono and 16-bit PCM format for training, although XTTS also supports 16 kHz.

Two chunking strategies were used, each with distinct strengths:

*1) **Statistically Sampled Chunking**:* This method samples chunk durations from a normal distribution, constrained between 3 and 11.6 seconds. This introduces natural variation in chunk length and avoids repetitive segment boundaries. Chunks are extracted sequentially from the full audio stream and exported as individual WAV files.

After chunking, each audio segment is transcribed using OpenAI's Whisper Large-v3 [7], which returns timestamped text for each chunk. The resulting text is then post-processed to remove samples exceeding 250 characters or containing speech-to-text mismatches (using z-score outlier detection on characters per second).

*2) **Sentence-Level Chunking**:* To achieve more linguistically meaningful segmentation, sentence-level chunking was implemented using WhisperX [6] for transcription and word-level alignment. The resulting text is split into sentences using natural language tokenization. Sentences were then grouped incrementally to form chunks, constrained again between 3 and 11.6 seconds.

Each valid chunk is exported as a WAV file, and its corresponding transcript is stored in LJSpeech format. This method yielded cleaner boundaries and better alignment between textual and acoustic information.

V. Fine-tuning

Fine-tuning XTTS-v2 allowed us to adapt the pretrained model to generate personalized speech outputs using datasets as small as around 45 minutes. Each fine-tuning run was performed using the base XTTS-v2 checkpoint provided by Coqui as the initialization point. The checkpoint includes pretrained weights for various components of the model: the

discrete variational autoencoder (stored in dvae.pth), and the GPT decoder and HiFi-GAN vocoder (included in model.pth).

A. *Configuration and Hyperparameters*

Throughout fine-tuning experiments, configuration and hyperparameter values were adjusted for each run based on speaker identity, dataset size, available hardware, and goals.

The BaseDataSetConfig specifies how the dataset is loaded and formatted. In our case, it used the LJSpeech format, referencing a single metadata.csv file that maps each WAV file to its transcript. At the audio configuration level, the XttsAudioConfig used a sample_rate and dVAE_sample_rate of either 16kHz or 24 kHz.

Within the GPTArgs model configuration, we limited reference audio lengths by setting max_conditioning_length to 132,300 samples (6 seconds) and min_conditioning_length to 66,150 (3 seconds). The max_wav_len parameter dictates the full length of sample audio processed during training and is capped at 255,995 samples (11.6 seconds). To convert number of audio samples in terms of time, they must be divided by the sample rate (Hz).

XTTS uses a discrete audio token space of 1,026 entries: 1,024 learned audio codes and two special tokens, [START_AUDIO] and [STOP_AUDIO], at indices 1,024 and 1,025. During training, these tokens are embedded and passed through the transformer along with text embeddings and prepended conditioning latents.

We enabled gpt_use_perceiver_resampler to compress the highly-dimensional conditioning encoder embeddings into a fixed-length representation before feeding them into the GPT decoder. Additionally, we enabled gpt_use_masking_gt_prompt_approach to activate the masked ground truth prompting strategy.

For the optimizer and training loop (GPTTrainerConfig), the model was optimized using AdamW, with most parameters left as default. A MultiStepLR learning rate scheduler was used in longer training runs to decay the learning rate over time, though its impact was limited in shorter finetunes. We used a batch size of 3 and gradient accumulation of 84 steps, simulating a large effective batch size of 252 as recommended by Coqui for more efficient training.

Collectively, these hyperparameters and configurations were adjusted to suit the size and structure of each speaker dataset and to maximize speaker similarity, prosody retention, and synthesis stability across training conditions.

B. *Training*

While fine-tuning XTTS-v2, we used a single speaker dataset to ensure the model was tuned to one particular speaker. To create the full set of speakers, we employed the base XTTS-v2 model provided by the original authors of XTTS-v2 as the initial checkpoint model for each run. This approach allowed us to build a multitude of datasets from both audio-books and personal sampling without having to integrate them into a single dataset. Since we used both language model heads during training, 3 loss metrics are computed during training. The loss_text_ce is the cross-entropy loss that measures how well the GPT-2 transformer predicted the next text token in the input sequence, and the loss_mel_ce, which is the cross-entropy loss that measures how well the GPT is predicting the next discrete audio token. Unlike many other tasks done by neural networks, monitoring if the model is over-fitting during training cannot be done well through loss metrics alone. Since the kind of overfitting we needed to avoid is in the voice quality itself, not necessarily in the next token prediction, a more manual method was required. From checking each checkpoint step, we could decide the best version of the model throughout its training to save. Doing this allowed us to preserve most of our samples for training, only delegating two percent of the training set to validation.

C. *Experiments*

We conducted a series of fine-tuning runs for both our celebrity and personally voiced datasets. The celebrity datasets were derived from larger samples than our personally voiced samples. This difference was 2 to 4 hours for our celebrity samples each, to about half an hour to an hour for the personally voiced samples. For each of these kinds of samples, we did not vary many parameters. The most impactful difference we observed between training on larger and smaller datasets was how prone each of them were to over-fitting.

On the large celebrity datasets, we observed fairly rapid convergence to a good sounding voice. The celebrity models achieved their best sounding voice only after 2 epochs. Typically after 2 epochs, the voice became a generic North American sounding voice, losing much of the likeness of the target. Meanwhile, the smaller personal datasets did not achieve their best quality until between 10 and 40 epochs.

We also compared using 16 kHz versus 24 kHz sampling rate during training. At 16 kHz, the synthesized voices tended to sound unnaturally deep and flat. Switching to 24 kHz during training increased voice richness and likeness. All final models, including the small-dataset runs, used a 24 kHz sample rate for both conditioning and output.

VI. Inference

Using XTTS-v2 for inference can be done with or without having done fine-tuning. From the TTS package, the XttsConfig and Xtts modules are needed. Inference begins by loading the model's configuration and checkpoint. The configuration object can initialized by loading the config.json(which defines audio sample rates, token lengths, and hyperparameters) into the XttsConfig loader then initialize the Xtts model from that config.

Once the configuration object is called and loaded with its content, the model can be initialized using the Xtts.init_from_config() method. From here, call the model's load_checkpoint method, setting it to a folder containing the needed files: model.pth (weights), vocab.json (BPE vocabulary), and speakers_xtts.pth (speaker embeddings). After loading these components, set the model into evaluation mode and moving it to a specific device (GPU or CPU).

Next, the conditional latent codes are needed. This can be retrieved by using the model method

get_conditioning_latents(), while passing in the path to the speaker reference audio file (should be between 4 to 6 seconds long). This step outputs two latent vectors: a GPT conditioning latent (gpt_cond_latent) from the Perceiver Resampler, and a speaker identity vector (speaker_embedding) from the Speaker Encoder. If not using a fine-tuned model, the speaker embedding will be primarily used to ensure the output aligns with the speaker reference input. Passing multiple speaker reference inputs is possible here.

With the model loaded and the latents computed, the model is ready for inference. Call the inference() method, passing as arguments the text input to convert to audio, the conditional latent codes, and the speaker embedding. Specifying optional parameters such as temperature, speed, and sampling thresholds to shape the prosody and creativity of the output is also possible. Finally, convert the resulting waveform array into a tensor and write it out at the model's output sample rate.

Before running inference on longer text input, it is ideal to break sentences into discrete units to avoid having the model perform inference at an audio length not used during training (between 3 and 12 seconds). This can be done by splitting the sentences, performing inference on each individual one, then concatenating the audio segments into a single audio output.

In our demo, this process has been wrapped into a Streamlit app. The process of loading pre-trained models, providing speaker reference audio, and performing inference on a given text can be done without use of the command line interface.

VII. Novel Future Applications

XTTS-v2's ability to generate high-fidelity, speaker-consistent speech from minimal data allows a range of new use cases beyond traditional TTS. One particularly impactful application is in augmentative and alternative communication (AAC) for stroke survivors that experience a loss in speech. By fine-tuning XTTS-v2 on about half an hour of a patient's own recordings, XTTS-v2 can deliver personalized synthetic voices that return an individual's unique voice. Since many people have at least half an hour of continuous or discontinuous audio recordings of their own voice, this method vastly improve the patient's connection to a synthetic voice aid compared to generic TTS voices.

Another exciting frontier is in giving large-language-model (LLM) agents customizable voices. This can create value by improving the likability or, when using a celebrity trained model, branding of virtual assistants, customer service bots, and AI companions. Beyond healthcare and LLMs, XTTS-v2 can be used for media experiences. For example, authors not backed by big publishers can self-publish with a well-voiced audiobook attachment for little cost.

These applications demonstrate how a fine-tuned XTTS model can create valuable impact in a wide range of contexts.

VIII. Conclusion

This paper explored the fine-tuning of XTTS-v2, a multilingual and multispeaker text-to-speech model, on single-speaker datasets. Using pretrained components, including the GPT-based decoder, VQ-VAE and HiFi-GAN vocoder, we were able to generate high-quality speech outputs that effectively cloned speaker identity and prosodic features. This was achieved through strategies such as conditioning resampling, token-level masking, and audio-text alignment to optimize model behavior across different speaker profiles.

The results across multiple datasets displayed the sensitivity of XTTS-v2 to both dataset size and quality. Larger datasets typically led to faster convergence but were more prone to overfitting beyond a small number of epochs, resulting in generic-sounding output. In contrast, smaller datasets required more training epochs to stabilize and benefited more from chunk alignment and parameter tuning.

Throughout this work, we found that subjective checkpoint evaluation was more reliable than loss metrics in assessing voice fidelity and likeness, emphasizing the importance of monitoring during training. Inference procedures further demonstrated the model's flexibility in handling different speaker embeddings and prosodic conditions through a user-friendly UI that supports real-time customization.

Ultimately, XTTS-v2 proved to be flexible and high-performing for voice cloning and speaker adaptation when paired with carefully constructed datasets and configurations. With appropriate preprocessing and tuning, it can be fine-tuned to new speakers with minimal data, supporting practical applications in personalized speech synthesis, including accessibility tools, voice assistants, and multilingual content localization.

Acknowledgment

We would like to thank the Coqui AI team for releasing the XTTS-v2 model and for providing clear, thorough documentation. Their GitHub codebase and fine-tuning guide were invaluable in helping us understand the model's architecture and develop our fine-tuning workflow.

References

[1] A. Eremenko, T. Helgaker, B. Mehri, and A. Behr, "XTTS: Scalable, high-quality, multilingual text-to-speech," *arXiv preprint arXiv:2406.04904*, 2024.

[2] J. Betker, "Better speech synthesis through scaling," *arXiv preprint arXiv:2305.07243*, 2023.

[3] P. Gage, "A new algorithm for data compression," *C Users Journal, vol. 12, no. 2, pp. 23–38*, 1994.

[4] Free Latest Audiobooks, "The Terminal by Tom Hanks – Audiobook Part 1," *YouTube, Apr. 8, 2024. [Online]. Available: https://www.youtube.com/watch?v=BQ79Fi6-NKc&t=8810s*

[5] Just Free Audiobooks, "Benedict Cumberbatch Reads Sherlock Holmes: The Hound of the Baskervilles – Full Audiobook," *YouTube, Jan. 10, 2024. [Online]. Available: https://www.youtube.com/watch?v=84VVFgGmeSA*

[6] M. Bian, "WhisperX: Extended Automatic Speech Recognition with Word-Level Timestamps," *GitHub repository, 2023. [Online]. Available: https://github.com/m-bain/whisperx*

[7] A. Radford et al., "Robust speech recognition via large-scale weak supervision," *arXiv preprint arXiv:2212.04356*, 2022.

[8] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis," *arXiv preprint arXiv:2010.05646*, 2020.

[9] A. Radford et al., "Language Models are Unsupervised Multitask Learners," *OpenAI Technical Report*, 2019.