PiVo: two English voices in 1.4 million inference parameters

A compact English text-to-speech model, vocoder included

Leon Lehmann

Empero Research

research@empero.org

Abstract

We asked ourselves: how many parameters does a voice need? Not just the acoustic model, but everything between a line of text and the waveform, vocoder included. PiVo is our current answer. It speaks English in two fixed synthetic voices, Pi and Vo, using 1,401,562 learned parameters in total: a 584,965-parameter acoustic model that predicts 20 vocoder features every 10 ms, and an 816,597-parameter vocoder derived from FARGAN that turns them into 16 kHz audio. That is roughly 1,200 times smaller than the 1.7-billion-parameter Qwen3-TTS teacher whose recordings it learned from. Trained on 8,628 teacher-generated utterances, PiVo is easy to understand: on 266 held-out utterances, Whisper transcribes it with word error rates of 0.863% on validation and 0.532% on test. Sounding natural is harder. Some sentences keep a metallic or robotic texture, and our attempt to fix this by adapting the vocoder made no consistent difference in 20 blind paired comparisons. We release the weights, ONNX graphs, a PyTorch-free runtime and a small WAV server. All evaluation text is in-domain, the listening review had one listener, and speed on phone-class ARM hardware is still unmeasured.

- **The whole learned stack is 1.40M parameters.** No teacher, speaker encoder, aligner or discriminator is needed at inference.
- **Intelligible is not the same as natural.** Held-out word error stays below 1%, yet some text still comes out metallic or robotic.
- **Our vocoder fix did not survive a blind test.** Adapting the vocoder cleaned up one reference example, but 20 fresh paired comparisons gave 18 ties and one win each way.
- **It runs on a plain CPU without PyTorch.** Four ONNX graphs and NumPy are enough, and the graphs match the PyTorch model to within 7×10^{-6}.

1 Why build a tiny voice

Our teacher model, Qwen3-TTS, has 1.7 billion parameters. We wanted to see how much of a voice survives when the entire synthesizer has to fit in 1–1.5 million, with a hard ceiling of five million, and when the count includes the vocoder rather than leaving it off the books. We fixed two English voices: Pi, a feminine voice, and Vo, a masculine one. The longer-term goal is offline, real-time speech on a phone-class ARM CPU. A small parameter count makes that plausible; it does not prove it, and we have not measured it yet.

The acoustic model takes its inspiration from Mariusz Kurman's Neuroblast project (Kurman, 2026). Our implementation is original and uses no Neuroblast weights. For the vocoder we chose FARGAN (Valin et al., 2024), because its explicit pitch conditioning and frame-by-frame generation are a natural fit for a very small budget. Qwen3-TTS (Hu et al., 2026) designed the two voices and recorded our training data; it plays no part at inference.

This report describes the model we selected, how we trained it and what we measured. We do not rank PiVo against other public TTS systems, because our measurements come from our own synthetic corpus, single training runs and one listener. Vocal events, other languages and voice cloning are outside the scope of this version.

2 Inference architecture

2.1 From text to acoustic features

A deterministic English frontend expands numbers, currencies, ordinals and ISO dates, then looks up stressed ARPAbet phones in the CMU pronouncing dictionary. When a word is not in the dictionary, the frontend refuses it instead of guessing, and the caller can supply a pronunciation. This keeps coverage gaps visible; it is not a learned grapheme-to-phoneme model. Short all-caps words are read as initialisms, with an exception for NASA, which keeps its dictionary pronunciation (Section 4.2).

The acoustic model is 96 wide, with feed-forward width 192, four query heads and two key/value heads. Its text encoder has six blocks that alternate grouped-query attention with dilated temporal convolution. Attention uses RMS normalization and rotary positions, and the feed-forward layers use gated SiLU activations. A learned two-entry embedding tells the encoder which voice to use. The token embedding reserves 128 entries, all of which are included in the parameter count.

A width-64 predictor estimates log durations for each phone. At inference we invert and round them, give PAD, BOS and EOS zero frames, and give every spoken phone at least one. Spaces and punctuation may receive zero frames. Each phone's encoding is then repeated for its duration. Four simple timing signals tell the decoder where it is inside a phone: relative progress, its sine and cosine, and the log duration. They are computed on the host and enter through a learned projection that is counted in the acoustic model.

Two causal attention blocks then predict 20 values for every 10 ms frame: 18 cepstral coefficients, a pitch value and a pitch-correlation feature. Attention looks back at most 64 frames, which keeps the decoder's key/value cache bounded. Only the first decoder block uses rotary positions. These 20 features are exactly what the Opus FARGAN vocoder expects, so we predict them directly rather than going through mel spectrograms.

2.2 From features to waveform

The vocoder looks at the previous, current and next feature frame and generates four 40-sample subframes for each 160-sample frame. Pitch feedback and recurrent state are explicit inputs and outputs. The host converts feature 18 into a pitch period in samples:

$$p = \mathrm{clip}_{[32,255]}\left(\mathrm{round}\left[256\,2^{-(f_{18}+1.5)}\right]\right). \tag{1}$$

The host also applies de-emphasis with coefficient 0.85 and carries that filter state from chunk to chunk. Every utterance starts from zero state.

We export four ONNX graphs: encoder, acoustic decoder, conditioning network and vocoder step. The encoder reads the whole input text first; after that, the Python runtime yields audio in 160-sample (10 ms) frames. Audio is therefore streamed, but text is not,

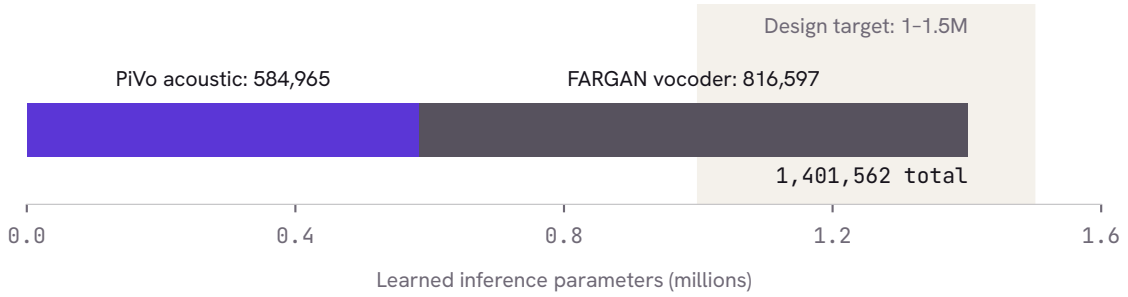


Figure 1. The complete stack fits the 1–1.5M target with the vocoder included. The acoustic model has 584,965 parameters and the FARGAN-derived vocoder 816,597, for 1,401,562 in total. The shaded band marks the design target. Parameter counts are not memory or speed measurements.

and we have not yet measured the delay to the first audio frame. The vocoder needs one frame of lookahead.

2.3 What we count

The two components hold 1,401,562 learned parameters (Fig. 1), or 5,606,248 bytes in FP32. The ONNX files are somewhat larger, because they also store graph structure, constants and calibration data. The dictionary and the rule-based frontend are not learned parameters, but they do ship with the runtime.

At the maximum attention history, the four decoder key/value arrays hold 48,384 bytes and the five vocoder state arrays another 3,344, for 51,728 bytes of model state. Encoded text, expanded features, buffers and the ONNX Runtime workspace come on top of that. We have not yet measured total process memory on ARM.

3 Data and training

3.1 Two synthetic voices

We designed both voices with the Qwen3-TTS VoiceDesign model and generated the recordings with the Qwen3-TTS Base model. Both model cards declare Apache-2.0. Pi and Vo are synthetic identities: no real speaker was recorded, and no speaker encoder is needed at inference. Teacher versions, voice descriptions and model-card hashes are recorded in the model repository.

The teacher generated 9,656 recordings, about 13.93 hours. After quality filtering we kept 8,628 utterances, which form 4,314 scripts read by both Pi and Vo, about 12.29 hours before trimming. The scripts combine our own writing with excerpts from Austen, Conan Doyle, Dickens, Carroll, Grahame and Montgomery. The release includes the model and our own demonstration sentences, but not the training audio.

We split the data into 8,362 training, 114 validation and 152 test utterances. Both readings of a script always land in the same split, so the 266 held-out utterances cover 133 distinct scripts. The held-out text comes from the same sources and the same teacher as the training data. It tells us how well PiVo learned its corpus, not how it handles arbitrary English or how it compares with human speech.

3.2 Alignment and trimming

Montreal Forced Aligner 3.4.3, with the English US ARPA 3.0.0 model, gives phone timings that match our frontend's pronunciations. A pinned native extractor computes the 20 features for each 10 ms frame. Its 80-sample analysis delay is compensated by padding the start of each target waveform and dropping the unmatched tail.

While auditing durations we found that leading silence was being counted as part of the first spoken phone. We therefore built an edge-corrected corpus that keeps the spoken region, three frames of margin and any energetic frames, trimming only at whole-frame boundaries so that features, waveforms and durations stay aligned. All 8,628 utterances and their split assignments survive, no spoken phone ends up with zero duration, and 3,877,299 frames remain, about 10.77 hours. Fixing the timing was worth doing on its own; it did not by itself make the voices sound better.

3.3 Supervised training and spectral refinement

The parent model minimizes a smooth-L1 loss on the acoustic features plus 0.1 times the squared error on log(1 + duration). We train with batch size eight, learning rate 3×10^{-4} and 200 warm-up updates. Optional cepstral-delta and voiced-pitch losses are switched off. The run stops at epoch 89; the checkpoint with the lowest validation loss comes from epoch 82, after 85,772 updates. Training took about 90 minutes (5,372.6 seconds).

We then refine the model for 1,200 updates with batch size four, 128-frame segments and learning rate 10^{-4}. A 110,018-parameter two-scale temporal critic sees the 18 cepstral channels and the voice identity. The generator keeps its supervised loss and adds adversarial and feature-matching terms, weighted 0.01 and 0.1. The encoder and duration predictor are frozen, and we verified that all 61 frozen tensors are unchanged afterwards. The critic is used only during training and adds nothing at inference. The checkpoint after 1,200 refinement updates is the model we release. It comes from a single run with seed 42; we have not studied variation across seeds.

4 Evaluation

4.1 Intelligibility and export accuracy

We compared 13 saved candidates on the same 114 validation and 152 test utterances. In the main setting, PiVo predicts both durations and features; settings that use reference features or reference durations serve as diagnostics. A pinned Whisper small.en model transcribes the output, and we report its raw word error rate without correcting recognizer mistakes. The references contain 1,738 words for validation and 2,254 for test, the same for both voices.

The selected model makes 15 word errors on validation (0.863%) and 12 on test (0.532%). Table 1 lists all 13 candidates. We chose the model before looking at test scores. Low word error means the words are clear; it says nothing about how natural the voices sound, and in our own listening the selected model still has a tinny, slightly robotic edge.

We also checked that the exported ONNX graphs behave like the PyTorch model. The largest absolute differences are 1.23×10^{-6} for the encoder, 6.68×10^{-6} for the decoder, 3.87×10^{-7} for the conditioning network and 2.98×10^{-7} for the vocoder step. Both voices synthesize with PyTorch imports blocked, and none of the selected model's 798 held-out waveforms (266 utterances in three settings) clips. These checks tell us the export is faithful, not that it sounds good.

Table 1. Many recipes reach low in-domain word error. All 13 candidates, evaluated on the same 266 held-out utterances. Parameter counts include the vocoder. WER is raw ASR word error in percent, over 1,738 validation and 2,254 test words. Candidates differ in data, preprocessing, training length and checkpoint selection; each is a single run, so small differences are not significant.

Candidate	Parameters (M)	Validation WER	Test WER
Frozen baseline	1.401178	2.647	1.775
Phase	1.401562	1.323	0.532
Phase + spectral	1.401562	1.381	0.710
1.5M phase	1.484746	1.093	0.710
1.5M + spectral	1.484746	1.036	0.621
Edge-corrected phase	1.401562	1.381	0.399
Edge-corrected + spectral	1.401562	1.266	0.399
Expanded phase	1.401562	0.978	0.311
Expanded 1.5M	1.484746	1.093	0.311
Expanded edge-corrected	1.401562	0.921	0.532
PiVo-1.4M (selected)	1.401562	0.863	0.532
Expanded + spectral	1.401562	1.036	0.399
Expanded 1.5M + spectral	1.484746	1.036	0.399

Table 2. The adapted vocoder brings no consistent improvement. One listener, ten texts in each of two voices, 20 blinded pairs. Three of the 18 ties were rated unusable in both versions. This review was run before the NASA pronunciation fix.

Voice	Baseline	Adapted	Baseline wins	Ties	Adapted wins
Pi	5.80	6.00	0	9	1
Vo	6.40	6.30	1	9	0
Both	6.10	6.15	1	18	1

4.2 Listening

To test a vocoder change (Section 5.1), we took ten new text excerpts from the web and rendered each in both voices, once with the original vocoder and once with an adapted one. Within each pair the acoustic features and timing were identical, so only the vocoder differed. One listener rated the 20 blinded, level-matched pairs on a 1–10 scale. The means in Table 2 summarize those ratings; they are not a standardized MOS study.

The listener described metallic sibilants, noise around pauses and a roboticness that comes and goes, sometimes in otherwise clear speech. “NASA” sounded wrong with both vocoders, and here the cause turned out to be simple: our frontend split short all-caps words into letters. Keeping the dictionary pronunciation for NASA fixed that example, and the listener confirmed the improvement. It is a pronunciation fix, though, not a change in texture. An optional post-filter for sibilants and pauses made no useful difference and stays switched off.

5 What did not work

5.1 Adapting the vocoder

On a familiar example, the artifact was already audible when we fed the original vocoder features extracted from the clean 16 kHz teacher recording. So the acoustic model’s predictions cannot be the only culprit. We also ruled out a porting error: over a full sentence, our floating-point vocoder differs from the pinned upstream model by about −114 dB RMS relative to the speech, in both voices. It also sounded better than the native C implementation, so switching to C is not the answer either.

We next adapted the vocoder on reference features. An early checkpoint reconstructed the reference more cleanly, if slightly muffled. But on real text-to-speech, where the vocoder receives predicted features, the output sounded robotic again. Hybrids that mixed reference spectra with predicted pitch, or the reverse, sounded worse than the reference, though mixing features can create mismatches of its own, so these hybrids do not isolate a single cause. The final adapted vocoder, after 1,200 updates, produced the tie in Table 2. We keep it as an experiment and ship the original.

Simple signal measurements did not settle the question either. On test utterances with reference durations, the selected model's cepstral frame-to-frame changes are about 0.64 (Pi) and 0.66 (Vo) times as large as in the reference, so its spectra move more smoothly than the teacher's. Its pitch feature is off by 1.65 and 2.82 semitones on average. These are feature-level numbers, not measurements of the final waveform, and changes to the corpus, alignment, normalization and training length make broad causal comparisons difficult.

5.2 Using an audio judge as a reward

We hoped an automatic quality model could stand in for a listener and eventually serve as a training reward. We tested one pinned UTMOSv2 fold (SaruLab, 2026) on 64 inputs, each scored with five crop seeds. On the 18 pairs where our listener had a preference, it agreed six times, disagreed seven times and abstained five times at a 0.05 score threshold. It did recognize the clean teacher audio, but it missed the listener's preference in the familiar example. With this little evidence we cannot use it as a reward. That rules out this configuration, not audio judges in general.

We have not tried RL, DPO or GRPO. PiVo's regressors are deterministic and do not produce the stochastic policy and log probabilities those methods expect. Preference training would first need a model change that can plausibly fix the texture, and a reward that tracks what listeners hear without favouring muffled output. Our listening notes are useful for diagnosis, but they do not yet amount to such a reward.

6 Release

The model repository contains the four ONNX graphs and their metadata, the selected acoustic checkpoint, the converted original vocoder, asset hashes, license notices and two demonstration recordings. The code repository provides a Python API and command-line tool, pronunciation overrides, a WAV HTTP endpoint, a Dockerfile and tests. On loading, it verifies the hashes of all five runtime assets, and it never downloads anything during synthesis.

The reference server runs one synthesis at a time per process and returns a retryable "busy" response rather than queueing requests. It rejects unknown words and invalid input. Input is limited to 2,000 characters and 512 frontend tokens, and one utterance can be at most 3,000 frames (30 seconds) long. Speed can be set between 0.5 and 2.0. Longer text should be split into clauses.

The public release is enough to reproduce synthesis. Training data, intermediate checkpoints and raw evaluation records are archived at Empero Research and are not part of this release, so full retraining is not yet possible from public material.

Our code and the acoustic weights are released under Apache-2.0. The FARGAN-derived code and converted upstream weights keep the Xiph/Opus BSD notice, and the dictionary keeps its CMU notice. The full license texts ship with each repository.

7 Limitations and next steps

A good result on a few samples does not rule out metallic, flat or robotic speech on other text. Our evidence comes from one listener, scripts shared between the two voices, single training runs and in-domain synthetic references. Word error rates do not measure naturalness. A vocoder that reconstructs reference features better does not automatically sound better on predicted features, and our negative result with the audio judge does not explain where the remaining artifact comes from.

The next things we want to measure are end-to-end latency, real-time factor, peak memory and sustained performance on specific ARM devices, along with wider pronunciation coverage and a larger listening study. Quantization and native app integration will each need their own accuracy and listening checks. We do not take any of these outcomes for granted from the parameter count or from FARGAN's published hardware results. For now, PiVo is a small, complete and reproducible baseline, and an honest record of what 1.4 million parameters can and cannot do.

A Artifacts and a minimal example

The selected acoustic checkpoint is the step-1,200 spectral refinement of the parent checkpoint with SHA256 prefix `7515d39ffa8696b0`.

Acoustic checkpoint

`0b4a2dc993e7e500aac1825465886ff23c11f2e1a2662577f2f5cd27f4efb4eb`

Original converted vocoder

`355d470826817afaa4e729d1b19c8383e985f365af5d2b913f1b9acdf8f7eca4`

Opus source revision

`bafd40dcbd506e12da257b8cb05c0afe1ed44719`

With the model files downloaded, synthesis runs fully offline:

```
from pivo import PiVo
model = PiVo("path/to/model", threads=1)
model.save("pi.wav", "NASA is ready for the next mission.",
           voice="Pi")
```

Machine-readable results are in `evaluation.json` and `PROVENANCE.json` in the model repository.

References

Hangrui Hu, Xinfa Zhu, Ting He, Dake Guo, Bin Zhang, Xiong Wang, Zhifang Guo, Ziyue Jiang, Hongkun Hao, Zishan Guo, Xinyu Zhang, Pei Zhang, Baosong Yang, Jin Xu, Jingren Zhou, and Junyang Lin. 2026. Qwen3-TTS Technical Report. *arXiv:2601.15621*. https://arxiv.org/abs/2601.15621.

Mariusz Kurman. 2026. Neuroblast v3. Software project. https://github.com/mkurman/neuroblast-v3. Accessed 1 November 2026.

Jean-Marc Valin, Ahmed Mustafa, and Jan Büthe. 2024. Very low complexity speech synthesis using framewise autoregressive GAN (FARGAN) with pitch prediction. *IEEE Signal Processing Letters*, accepted manuscript. https://jmvalin.ca/papers/valin_fargan.pdf.

SaruLab Speech. 2026. UTMOSv2. Software and pretrained evaluation model. https://github.com/sarulab-speech/UTMOSv2. Source revision `cc2700db57bb83ee`, fold 0.