CosyVoice 3, explained from zero

If you've ever used a text-to-speech (TTS) system and thought:

- “It said the right words… but it didn't sound like the person.”
- “The pronunciation was off.”
- “It sounded robotic-no emotion, no rhythm.”

…you've run into what speech researchers call *in-the-wild* speech generation: the messy, real-world version of the problem.

CosyVoice 3 is a modern TTS system built specifically for that reality-multiple languages, multiple accents, varied text formats, and voice prompts recorded in imperfect conditions.

This post explains the *fundamentals* you need to understand CosyVoice 3 (even with zero prior speech ML knowledge), and then walks through its architecture and training pipeline.

What “text-to-speech” really means

At the most basic level:

- **Input:** text (what you want spoken)
- **Output:** an audio waveform (a long list of numbers that, when played, becomes sound)

The tricky part is that the waveform is huge and very detailed:

- It encodes *what* is said (words),
- *who* says it (speaker identity),
- and *how* it's said (tone, emotion, rhythm, emphasis).

CosyVoice 3 aims to generate all of that reliably in real-world settings, and it reports improvements on **content consistency**, **speaker similarity**, and **prosody naturalness** compared to prior versions.

The big idea behind CosyVoice 3: “speech as tokens” + a fast renderer

A useful mental model is:

1. **Turn speech into discrete tokens** (like turning sound into “audio letters”)
2. Use a **language-model-like system** to generate those speech tokens from text + a voice prompt
3. Use a **fast continuous generator** to render those tokens into high-quality audio

This “two-stage hybrid” approach is described as a mainstream industrial choice because it balances quality, flexibility, and streaming compatibility.

Here's the pipeline in one diagram:

```
flowchart LR
  A[Text] --> B[Token LM<br/>predict speech tokens]
  P[Reference audio prompt<br/>(target voice)] --> B
  B --> C[Speech tokens<br/>(discrete IDs)]
  C --> D[CFM renderer<br/>(DiT backbone)]
  D --> E[Acoustic features]
  E --> F[Vocoder]
  F --> G[Waveform audio]
```

CosyVoice 3 names the renderer **CFM** (Conditional Flow Matching). (arXiv)

The 5 building blocks you need to understand CosyVoice 3

1) Discrete-token TTS (VALL-E-style): treating TTS like language modeling

Most people know **language models** (LMs) generate text one token at a time:

> “The next token after *hello* might be *world*.”

Token-based TTS does something similar, but the model generates **speech tokens** instead of word tokens.

A clean reference is **VALL-E**, which explicitly frames TTS as **conditional language modeling over discrete audio codec codes**:

- encode speech into discrete codes,

- train an LM to generate codes conditioned on text + an **acoustic prompt** (a short voice sample),
- then decode the codes back into speech.

VALL-E also highlights a practical “wow factor” of this approach: **zero-shot voice cloning** from a short enrolled recording (e.g., a few seconds).

How this maps to CosyVoice 3: CosyVoice 3 is in the same family: an LLM-like model generates discrete speech tokens, conditioned on text and (optionally) a reference prompt. (arXiv)

---

2) FSQ: the “simple quantizer” that turns continuous sound features into discrete tokens

To generate speech tokens, you first need a **speech tokenizer**: a model that converts waveform → a sequence of discrete IDs.

That requires **quantization**, which means:

> take a continuous vector (real numbers) and map it to a finite set of discrete values.

Classic approach: vector quantization (VQ)

Many systems use a learned “codebook” (a table of vectors). The encoder output picks the nearest codebook entry.

FSQ approach: Finite Scalar Quantization

FSQ simplifies this idea:

- project the representation down to a small number of dimensions,
- quantize each dimension to a small fixed set of values,
- the combination yields a large “implicit codebook” (product of per-dimension choices). (arXiv)

FSQ explicitly positions itself as “VQ-VAE made simple,” emphasizing fewer tricks and avoiding codebook-collapse style problems seen in some VQ setups. (arXiv)

How CosyVoice 3 uses FSQ

CosyVoice 3 inserts an **FSQ module** into the voice encoder of **MinMo** (their base speech understanding model), and forms tokens by projecting into a low-rank space, quantizing, then computing an index. (arXiv)

A very concrete spec that matters:

- **Token rate = 25 Hz**, i.e., **25 speech tokens per second**. (arXiv)

That means the LM is reasoning about speech at ~40 ms chunks-coarser than raw audio, but detailed enough to capture rhythm and style.

---

3) Conditional Flow Matching (CFM): a fast "renderer" from tokens to audio

Even if you have speech tokens, you still need high-fidelity sound. Tokens are like a *plan*; you need a *render*.

CosyVoice 3 uses a **Conditional Flow Matching (CFM)** model as its renderer. (arXiv)

Flow matching in plain language

Flow matching is a way to train a model to transform **noise → data** by learning a *direction field* (a "how to move" function) along a path. It is described as a **simulation-free** method for training continuous normalizing flows by regressing vector fields of conditional probability paths. (arXiv)

A key motivation: it can enable **fast sampling** using ODE solvers, and some paths (e.g., optimal transport) can be more efficient than diffusion paths. (arXiv)

Why speech people care

Speech generation systems often want:

- **high quality**
- **low latency**
- **non-autoregressive rendering** (generate frames in parallel / with few steps)

A speech-specific example is **Matcha-TTS**, which uses **optimal-transport conditional flow matching** and emphasizes **high output quality in fewer synthesis steps** than score-matching diffusion. (arXiv)

How this maps to CosyVoice 3: CosyVoice 3 uses CFM as the renderer after the token LM (and notes that downstream CFM + vocoder are computationally substantial in conventional RL setups). (arXiv)

---

4) DiT: using Transformers inside diffusion/flow-style models

Most people associate Transformers with text, but they're also used as backbones inside diffusion-like generators.

DiT (Diffusion Transformers) replaces the usual U-Net backbone with a Transformer operating on patches and reports strong scalability with compute ("Gflops"). (arXiv)

CosyVoice 3 adopts **DiT as the backbone** for its CFM renderer, scaling the CFM model up and simplifying other modules as a result (e.g., removing a complicated text encoder and length regularization, using interpolation for frame-rate mismatch). (arXiv)

---

5) Gumbel-Softmax: making "sampling tokens" differentiable

Here's a subtle but important point:

- The LM outputs probabilities over tokens.
- To generate audio, you typically **sample** a token.
- But sampling is not differentiable, so gradients can't flow through it.

Gumbel-Softmax is a trick that replaces a hard discrete sample with a differentiable approximation that can be smoothly annealed toward a true categorical sample. (arXiv)

CosyVoice 3 uses Gumbel-Softmax in DiffRO to sample predicted tokens and then optimize them with backprop (instead of running a traditional RL loop). (arXiv)

CosyVoice 3's core architecture

CosyVoice 3's "stack" has three named pillars:

A) A supervised multi-task speech tokenizer (MinMo + FSQ)

- Base model: **MinMo**, described as a multimodal LLM trained on **>1.4M hours of speech** with strong performance on speech tasks. (arXiv)
- Tokenizer training: supervised multi-task learning on **~530K hours**, including ASR, language ID, emotion recognition, audio event detection, and speaker analysis. (arXiv)
- Output: **25 Hz** speech tokens. (arXiv)

Why this matters: the tokenizer isn't trained just to reconstruct audio; it's trained to capture *meaning + paralinguistics* (emotion, pronunciation style), so the tokens are more useful for generating natural prosody. (arXiv)

B) A text-to-speech token LM (scaled up)

CosyVoice 3 scales:

- **Training data** from ~10K hours to **~1M hours**. (arXiv)
- **LM parameters** from **0.5B → 1.5B**. (arXiv)

It also expands language coverage to **9 languages** and **18+ Chinese dialects/accents**. (arXiv)

C) A CFM renderer (DiT backbone), plus a vocoder

CosyVoice 3 scales the renderer too:

- CFM model **100M → 300M parameters** and adopts **DiT** as the backbone. (arXiv)

Training methodology: how CosyVoice 3 is built

The paper's Figure 2 is the best “map,” because it shows:

- tokenizer training, and
- the multi-stage pipeline: pretraining → post-training → continual pretraining → speaker fine-tuning. (arXiv)

Below is that pipeline in plain language.

Step 1: Build a multilingual dataset from the internet (the “data pipeline”)

A token-based TTS model is only as good as the alignment between:

- audio (what was said) and
- text (what the transcript says)

CosyVoice 3 describes a six-step pipeline for “in-the-wild” audio (audiobooks, videos, podcasts): (arXiv)

1. **Speech detection & segmentation** Use diarization + voice activity detection + audio event detection to cut speaker-level segments. (arXiv)
2. **Noise reduction** Use MossFormer2 and remove abnormal truncations; trim silences. (arXiv)
3. **ASR transcription (and sanity checking)** Run multiple ASR systems and keep only transcriptions where the **average pairwise WER is < 15%** across systems. (arXiv)
4. **Punctuation adjustment (match text punctuation to real pauses)** Use forced alignment durations to add/remove punctuation with thresholds (e.g., ~300ms add comma; ~50ms remove pause punctuation). (arXiv)
5. **Volume standardization** Normalize volume via a simple normalization rule. (arXiv)
6. **Filter abnormal audio/text length ratios** Compute speech-token length vs text-token length; discard the smallest **1%** and largest **5%** to remove mismatched pairs. (arXiv)

If you're new to speech ML, this might sound like “boring plumbing,” but it's actually central to why these systems work.

Step 2: Train the speech tokenizer (supervised multi-task)

CosyVoice 3 trains the tokenizer by inserting FSQ into MinMo and supervising it on multiple tasks (ASR, emotion, language ID, etc.). (arXiv)

This is how they aim to produce tokens that carry:

- content (words),
- identity (speaker),
- and paralinguistics (emotion / style). (arXiv)

Step 3: Pretrain the token LM + CFM renderer at large scale

CosyVoice 3 reports scaling:

- data to **1M hours**, and
- LM to **1.5B parameters**, and
- CFM renderer to **300M parameters** with DiT. (arXiv)

Scaling is not just “bigger is better” marketing here-CosyVoice 3 explicitly studies data and model scaling as core levers. (arXiv)

Step 4: Post-training with DiffRO (Differentiable Reward Optimization)

The problem DiffRO tries to solve

Reinforcement learning (RL) can improve TTS, but there's a practical issue:

To judge a sample, you often need to run the full pipeline: **tokens → CFM → vocoder → waveform**.

CosyVoice 3 points out this downstream processing is computationally substantial, and the resulting waveforms can become hard to separate for reward modeling because they sound very similar. (arXiv)

The DiffRO move: optimize tokens directly

DiffRO:

1. trains an ASR-like **Token2Text** model,
2. uses the Token2Text posterior probability as a **reward**, and
3. uses **Gumbel-Softmax** to sample LM-predicted tokens in a differentiable way, allowing **backprop** rather than a classic RL loop. (arXiv)

It also adds a **KL divergence** term (computed on token-level logits) to keep the post-trained model from drifting too far from a reference model. (arXiv)

Finally, DiffRO can use **multi-task rewards** (emotion, MOS prediction, audio event detection, etc.) to help instruction-following control. (arXiv)

Step 5: Add “production painkillers”: pronunciation, text normalization, and instructions

CosyVoice 3 spends real effort on the stuff that breaks TTS in products.

Pronunciation inpainting

LLM-based TTS often consumes raw text tokens (BPE), which limits pronunciation controllability.

CosyVoice 3 adds the ability to model mixed sequences of words + phonemes by creating auxiliary data:

- replace some Chinese characters with **pinyin**
- replace some English words with **phonemes** using CMUdict (arXiv)

Text normalization without a brittle frontend

Traditional TTS systems use handcrafted rules to convert:

- “$12.50” → “twelve dollars and fifty cents”
- dates, symbols, etc.

CosyVoice 3 constructs auxiliary data using:

- rule-based text normalization + synth via CosyVoice 2,
- LLM-based normalization (Qwen-Max) + synth,
- and inverse text normalization to create raw text paired with real audio. (arXiv)

Instruction-following speech

CosyVoice 3 expands instruction-following data from **1,500 hours → 5,000 hours**, growing style types to **100+**, and supports:

- natural-language prompts prepended to text (with a special `<endofprompt>` token),
- tags like `[laughter]`, `[breath]`, and emphasis markers. (arXiv)

Step 6: Transfer capabilities into speaker fine-tuning

CosyVoice 3 also discusses how to preserve multilingual and instruction-following abilities when fine-tuning on specific speakers.

Two highlighted ideas:

- train an auxiliary dataset to help turn a monolingual speaker into a multilingual speaker via explicit instructions,
- mix speaker data with instruction-following data and randomly mask prompts to reduce catastrophic forgetting. (arXiv)

Why CosyVoice 3 matters (even if you never train a model)

CosyVoice 3 is a strong example of a modern "production-shaped" speech model:

- It treats speech generation as **token generation + rendering** (like LMs + image decoders). (arXiv)
- It improves the weakest link of token-based TTS-**the tokenizer**-by using supervised multi-task training so tokens carry richer prosodic cues. (arXiv)
- It proposes a pragmatic post-training method (**DiffRO**) that avoids expensive waveform-level RL. (arXiv)
- It shows the "boring parts" (data cleaning, punctuation matching, filtering) are essential at million-hour scale. (arXiv)

Practical note: code & demos

There is a public repository that presents "Fun-CosyVoice 3.0" with demos and deployment tooling, and lists capabilities like multilingual coverage, pronunciation inpainting, text normalization, and low-latency streaming. (GitHub)

(If you're evaluating the system for product work, treat the paper as the conceptual foundation and the repo as an evolving implementation snapshot.)

Glossary (so you don't need a speech ML dictionary)

- **TTS (Text-to-Speech):** generating audio from text.
- **Token:** an integer ID from a finite vocabulary.
- **Speech tokenizer:** a model that converts audio into a sequence of tokens.
- **LM (Language Model):** a model that predicts the next token given context.

- **Prosody:** rhythm, pitch, emphasis-how speech is delivered.
- **Vocoder:** converts acoustic features into waveform audio.
- **Flow matching / CFM:** a generative method that learns how to transform noise into data, often enabling fast sampling. (arXiv)
- **DiT:** diffusion/flow-style generator with a Transformer backbone. (arXiv)
- **Gumbel-Softmax:** differentiable approximation to sampling discrete tokens. (arXiv)

If you want a “study path” to reread CosyVoice 3 Sections 2-3

1. Read CosyVoice 3's Section 2.1 (tokenizer): identify MinMo, FSQ insertion, and the 25 Hz token rate. (arXiv)
2. Read Section 2.2 (DiffRO): focus on Token2Text reward + Gumbel-Softmax + KL term. (arXiv)
3. Read Section 3 (data pipeline): understand why ASR cross-validation and punctuation adjustment exist. (arXiv)
4. Only then revisit the scaling paragraph (1M hours, 1.5B LM, 300M DiT renderer). (arXiv)

That order makes the paper feel like a coherent system design-not a bag of tricks.
