

Fish Audio Drama 3 Technical Report

Fish Audio Team

 <https://github.com/fishaudio/fish-speech>

 <https://huggingface.co/fishaudio/drama-3>

 <https://fish.audio>

Abstract

We introduce Fish Audio Drama 3, an open-source speech generation model that turns text into directed performances, built on a 35B-parameter mixture-of-experts backbone that activates 3B parameters per token. It clones voices from multiple references, follows natural-language direction, revises a take when the next turn gives a note, voices multi-character scenes, edits existing speech by infilling, and synthesizes long-form and streaming audio. Multi-turn revision, scene dubbing, editing by infilling, and streaming text input are new in Drama 3, all in one model. Drama 3 pairs the hybrid linear-attention backbone with a new codebook multi-token-prediction audio decoder that predicts the acoustic codebooks of each frame in dedicated layers that attend over the entire sequence. The model is trained with a three-stage pre-training recipe, instruction fine-tuning, and large-batch reinforcement learning whose only reward is a single learned model of human preference. Its inference engine decodes a single stream at 271 frames per second on one H200, against 177 for Fish Audio S2, although Drama 3 has more than seven times as many parameters. The model weights are available on [GitHub](#) and [Hugging Face](#), and inference is available through [SGLang-Omni](#). We highly encourage readers to visit <https://fish.audio> to direct their own performances.

1 Introduction

Large-scale text-to-speech (TTS) models now read text aloud with near-human intelligibility and speaker similarity ([Anastassiou et al., 2024](#); [Zhang et al., 2025](#); [Du et al., 2025](#); [Hu et al., 2026](#)). The applications that drive the field today—audiobooks and audio drama, film and game dubbing, and interactive characters—ask not only for a faithful reading but for a performance. A performance is shaped by direction, the natural-language description of how a line is to be delivered. The delivery of a line is described in words, refined over several takes, placed within a scene among other characters, and corrected locally when a single word is wrong.

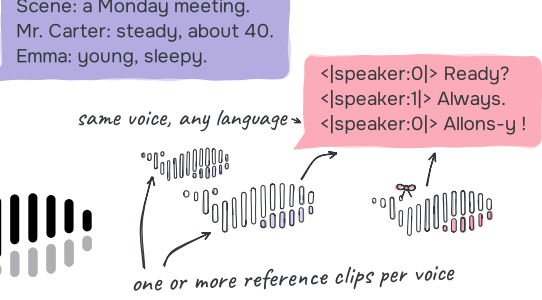
Our previous model, Fish Audio S2 ([Liao et al., 2026](#)), took a first step in this direction with inline natural-language instructions and native multi-speaker generation. Four limitations remain. First, the architecture was not built for long directed sessions, whose references and earlier takes run to thousands of audio tokens. S2’s full-attention backbone attends over all of them, and its per-frame fast autoregressive (AR) module ([Liao et al., 2024](#)) sees the references only through one backbone state. Second, control is single-shot. An instruction shapes one generation, and there is no way to give a note on a take the model has already produced and have it redone. Third, generation is all-or-nothing. Fixing one word inside a long take requires regenerating the take, which changes everything around the word. Fourth, post-training optimizes proxies. Reinforcement learning (RL) for TTS commonly rewards low word error rate and high speaker similarity ([Du et al., 2025](#); [Liao et al., 2026](#)), yet an expressive performance legitimately departs from both. A laugh, a long pause, or a whisper changes exactly the transcript and timbre that such metrics expect.

In this report, we present Fish Audio Drama 3, a speech generation model built around directed performance. Drama 3 keeps the discrete audio tokenizer of S2 and redesigns the model, the task formulation, and the post-training objective around it. Specifically, we introduce three key technical innovations:

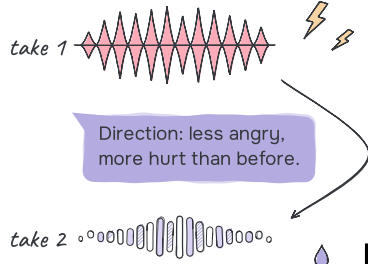
Fine-Grained Direction



Multi-Speaker Scenes

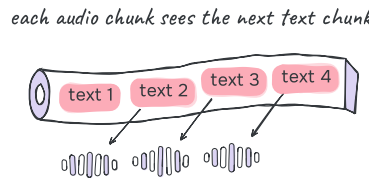


Multi-Turn Revision

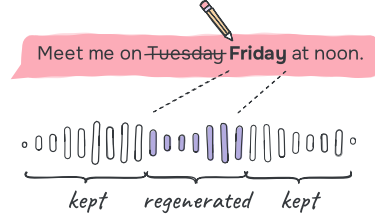


Fish Audio Drama 3

Direct a performance



Edit Takes



Long-Form & Streaming

Figure 1: **Fish Audio Drama 3** turns text into directed performances. It casts voices from references, follows natural-language direction, revises a take when the next turn gives a note, stages multi-character scenes, edits existing takes, and synthesizes long-form audio, streaming speech as the text arrives.

- **Hybrid Backbone with a Codebook-MTP Decoder.** Drama 3 builds on a hybrid linear-attention language model (Katharopoulos et al., 2020; Yang et al., 2025; Qwen Team, 2026) with a sparse mixture-of-experts feed-forward network (Shazeer et al., 2017), and replaces the per-frame Fast AR of S2 with a codebook multi-token-prediction (MTP) decoder (Gloeckle et al., 2024; DeepSeek-AI et al., 2024), in which each residual codebook (Zeghidour et al., 2022; Kumar et al., 2023) is predicted by a dedicated layer that attends causally over the entire sequence, including the fine acoustic detail of the reference audio.
- **Multi-Turn Revision, Scene Dubbing, Editing, and Streaming Text Input in One Model.** Drama 3 learns tasks that S2 did not support, such as revising a take from a note in the next turn, dubbing a whole multi-character scene, editing speech by infilling, and speaking text as it streams in. Each is written as a conversation over the same token stream as cloning and directing, with the loss applied only to the model’s own turns (Section 3, *Conversation format*).
- **A Single Reward Model, No Rule-Based Rewards.** Because expressive delivery breaks the transcription and similarity proxies above, we eliminate every rule-based reward. Reinforcement learning optimizes a single learned reward model (Christiano et al., 2017; Ouyang et al., 2022), a smaller speech language model of our own fine-tuned on human preference judgments, whose calibrated and bounded reward keeps reward hacking (Amodei et al., 2016; Gao et al., 2023a) observable. We train with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) without a Kullback–Leibler (KL) penalty and on very large batches, which in our development was more stable than small batches with many steps.

These innovations directly enable three functional breakthroughs (Figure 1):

- **Directable Performance.** Drama 3 follows inline and scoped tags, style descriptions, and free-form director’s notes. Directing is multi-turn. When a take is not what the director wanted, the next turn gives a note such as “softer and warmer versus previous” and the model redoes the take, with the earlier takes kept in context.
- **Scenes and Dialogue.** The model casts several voices from reference clips and performs multi-speaker dialogue and multi-character scenes in a single pass, with explicit control over line boundaries; a

character’s lines may be in different languages, in the same voice.

- **Editing and Long-Form Synthesis.** The model replaces a word or a sentence inside existing speech while keeping the surrounding audio, and synthesizes long-form audio line by line or in streaming mode.

Drama 3 is served by an inference engine designed for its hybrid backbone and codebook-MTP decoder (Section 4). Its codebook-MTP decoder runs once per frame instead of once per codebook, its experts run in the block-scaled FP8 of the RL rollouts, and the engine removes the kernel launches that CUDA graphs leave in place, optimizes kernels on the shapes of live serving, and separates prefill from decode so that new requests do not stall running streams. On one H200, it decodes a single stream at 271 frames per second against 177 for S2, although it has more than seven times as many parameters.

We evaluate Drama 3 on public benchmarks (Section 5). For directable performance, each revision round on the Step-Audio-Edit benchmark (Yan et al., 2025b) raises its emotion and style accuracy. For voice cloning, it has the lowest error rate of all compared systems in Chinese, Dutch, English, German, and Turkish on the MiniMax Multilingual Testset (Zhang et al., 2025). For editing on the Ming-Freeform-Audio-Edit benchmark (Yan et al., 2025a), it has a lower error rate than the infill API of Cartesia Sonic-3 in English and Chinese, and in Chinese a higher speaker similarity than dedicated editing models on insertion and substitution.

To accelerate research and lower the barrier to expressive speech generation, we publicly release the model weights on [GitHub](#) and [Hugging Face](#), with inference available through [SGLang-Omni](#). To our knowledge, Drama 3 is the largest open-weight model dedicated to speech generation to date, by total parameters. We also highly encourage readers to direct their own performances at our official site <https://fish.audio/>.

2 Architecture

Drama 3 keeps the overall design of S2, in which an autoregressive language model reads interleaved text and audio tokens and predicts the next frame’s semantic code, and a second module predicts its remaining acoustic codes. It changes three components (Figure 2): a hybrid linear-attention backbone replaces full attention, only two of the ten codebooks are fed back into the backbone, and a time-causal codebook-MTP decoder replaces the per-frame Fast AR of S2’s Dual-AR design (Liao et al., 2024). Drama 3 has no audio encoder, so reference and prompt audio enter only as audio tokens.

2.1 Audio Tokenizer

We reuse the S2 audio tokenizer unchanged. Built on the Descript Audio Codec (Kumar et al., 2023), it encodes 44.1 kHz audio into frames of ten codebooks at about 21.5 frames per second. The first codebook is semantic, and the remaining nine are residual vector quantizers (Zeghidour et al., 2022) that add progressively finer acoustic detail. It is fully causal, so encoding and decoding both stream. We write $q_t^{(k)}$ for the code of codebook $k \in \{0, \dots, 9\}$ at frame t . Each frame occupies one backbone position, so a second of speech costs about 21.5 positions regardless of the number of codebooks.

2.2 Backbone

The backbone is initialized from Qwen3.5-35B-A3B (Qwen Team, 2026), of which we use only the language model, a 40-layer hybrid that stacks blocks of three Gated DeltaNet layers (Yang et al., 2025), a gated form of linear attention (Katharopoulos et al., 2020; Yang et al., 2024), and one gated full-attention layer (Qiu et al., 2025), a design that suits long-form speech. The Gated DeltaNet layers carry the history in a constant-size state and only one layer in four keeps a key–value (KV) cache (Pope et al., 2022), while the full-attention layers keep exact retrieval over the whole context, such as the timbre of a reference clip or a line far back in a script. Every layer uses a mixture-of-experts (MoE) feed-forward network (Shazeer et al., 2017) that routes each token to 8 of 256 experts and adds a shared expert (Dai et al., 2024), so only about 3B of the 35B backbone parameters are active per token. We replace Qwen3.5’s native MTP layer (Qwen Team, 2026), which predicts the token after next, with our own codebook-MTP decoder (Section 2.4).

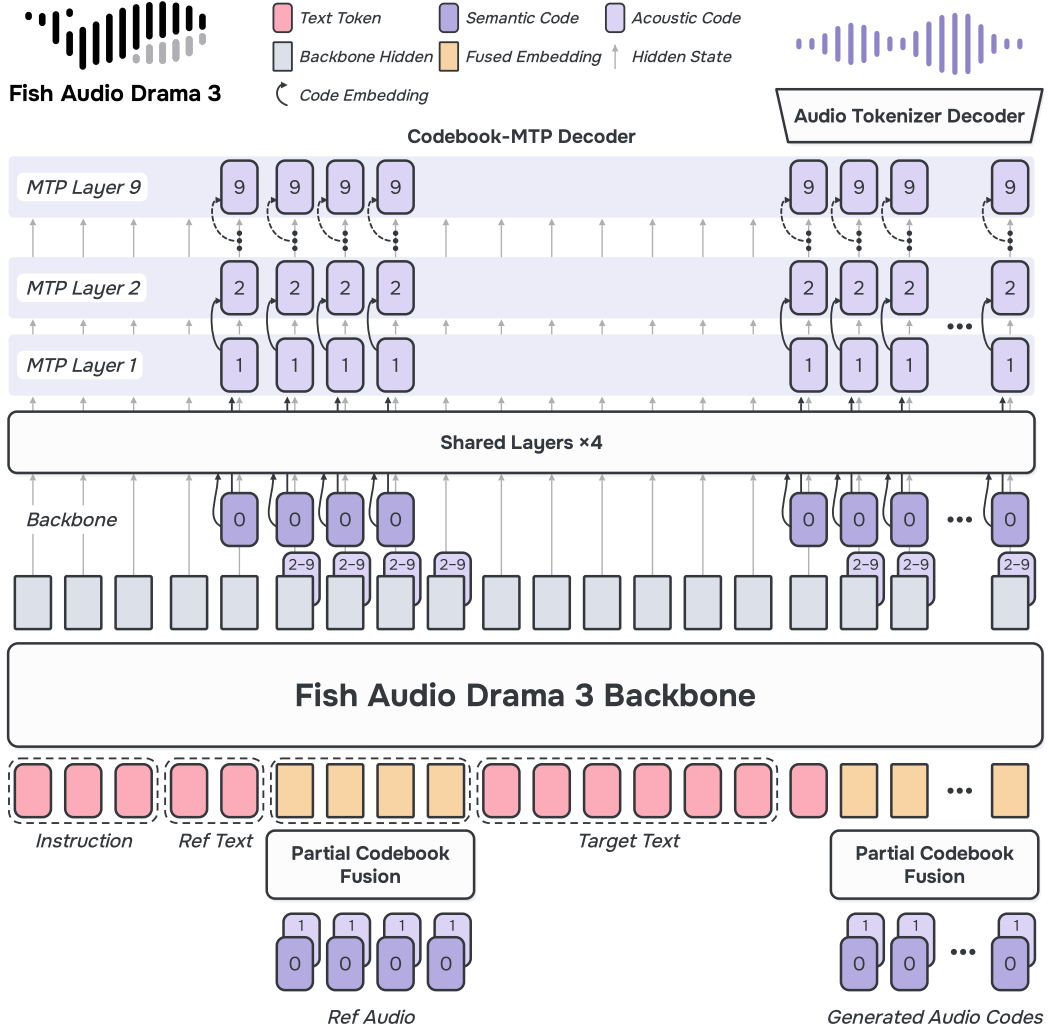


Figure 2: **Fish Audio Drama 3 architecture.** The backbone reads the system prompt, the target text, and then its own audio frames, each entered as a fused embedding of its codes 0 and 1. The codebook-MTP decoder runs over every position on top of the backbone’s hidden state. Its four shared layers take that state, the predicted code 0, and the input frame’s codes 2–9, which bypass the backbone, and MTP Layers 1–9 then predict the remaining codes one at a time. Codes are predicted wherever the next token is an audio frame, every layer attends causally along time, and the audio tokenizer decoder turns the ten codes into a waveform.

2.3 Partial Codebook Fusion

S2 fed all ten codebooks back into its backbone by summing their embeddings, a scheme it calls Multi-Codebook Fusion (Liao et al., 2026). Drama 3 feeds back only the semantic and the first acoustic codes, to limit exposure bias (Bengio et al., 2015; Ranzato et al., 2016) from its own sampled codes while still giving the backbone a coarse view of the acoustics of both the reference audio and its own generated speech. At the position of audio frame t , the backbone input is

$$\mathbf{x}_t = \frac{1}{\sqrt{2}} \left(\mathbf{E}^{\text{LM}}[s(q_t^{(0)})] + \mathbf{E}^{(1)}[q_t^{(1)}] \right), \quad (1)$$

where $\mathbf{E}^{\text{LM}}$ is the backbone’s token embedding, $s(\cdot)$ maps a semantic code to one of the 4,096 semantic tokens added to the Qwen3.5 vocabulary, $\mathbf{E}^{(k)}$ is the slice of one learned codebook-embedding table reserved for codebook k , and $1/\sqrt{2}$ keeps the fused embedding at the scale of a single one. Text positions use $\mathbf{E}^{\text{LM}}$ alone, and codebooks $q_t^{(2)}, \dots, q_t^{(9)}$ bypass the backbone and enter the decoder’s first layer, described in Section 2.4.

In Figure 2, the fed-back pair is the stacked 0/1 chips above Generated Audio Codes; the Partial Codebook Fusion boxes turn each pair into one fused embedding (orange), which is also how reference audio enters the token stream.

2.4 Codebook-MTP Decoder

At the position of frame t , the backbone predicts the next frame’s semantic code $q_{t+1}^{(0)}$, and the codebook-MTP decoder predicts its nine acoustic codes $q_{t+1}^{(1)}, \dots, q_{t+1}^{(9)}$ at the same position (Figure 2, right). The decoder stacks 13 layers of one design, with four shared layers followed by nine per-codebook layers, one for each acoustic codebook.

Layer anatomy. Every decoder layer merges the hidden state from below with an injected code embedding and runs one transformer block (Vaswani et al., 2017):

$$\begin{aligned} \mathbf{u}_t^{(\ell)} &= \text{Merge}^{(\ell)}(\mathbf{e}_t^{(\ell)}, \mathbf{h}_t^{(\ell-1)}) = \mathbf{W}^{(\ell)}[\text{RMSNorm}(\mathbf{e}_t^{(\ell)}); \text{RMSNorm}(\mathbf{h}_t^{(\ell-1)})], \\ \mathbf{h}_t^{(\ell)} &= \text{RMSNorm}(\text{Block}^{(\ell)}(\mathbf{u}_{1:t}^{(\ell)})) =: \text{Layer}^{(\ell)}(\mathbf{e}_t^{(\ell)}, \mathbf{h}_t^{(\ell-1)}), \end{aligned} \quad (2)$$

where t indexes positions in the whole sequence (each audio frame occupies one position, so t is also its frame index), $\ell \in \{1, \dots, 13\}$, $\mathbf{h}_t^{(0)}$ is the backbone’s final hidden state, and $\mathbf{e}_t^{(\ell)}$ is the code embedding injected by layer ℓ ; the decoder runs over every position of the sequence, text included, and at a position that predicts no audio frame $\mathbf{e}_t^{(\ell)}$ is zero (except for the input-frame codes that the first shared layer adds, described in Equation 3) and no loss is taken. Here RMSNorm is root-mean-square normalization (Zhang & Sennrich, 2019), $\mathbf{W}^{(\ell)}$ is a bias-free projection of the concatenation from $2d$ to d , and $\text{Block}^{(\ell)}$ is one transformer layer with gated full attention (Qiu et al., 2025) and a dense SwiGLU feed-forward network (Shazeer, 2020) at the backbone’s width, whose causal self-attention covers positions $1, \dots, t$, and $\text{Layer}^{(\ell)}$ denotes the whole layer. The MTP layers are stacked along the codebook axis instead of shifted along time, so they differ from the MTP modules that predict several future frames at once (Wang et al., 2025b; Xie et al., 2026). Each layer attends causally over the whole sequence, including the reference audio, so fine codes are predicted from full context rather than from one summary vector.

Shared layers. At position t , whose input is frame t , the backbone predicts the semantic code $q_{t+1}^{(0)}$ of the next frame, and the decoder completes that frame at the same position. Each of the four shared layers, $\ell = 1, \dots, 4$, shown as Shared Layers $\times 4$ in Figure 2, takes the hidden state from below and injects the semantic code of the frame being completed. The first one also receives codes 2–9 of the input frame, which the backbone never saw:

$$\mathbf{e}_t^{(\ell)} = \mathbf{E}_s^{(\ell)}[q_{t+1}^{(0)}] + \mathbf{1}[\ell = 1] a_t \sum_{k=2}^9 \mathbf{E}^{(k)}[q_t^{(k)}], \quad (3)$$

where $\mathbf{E}_s^{(\ell)}$ is layer ℓ ’s own table for the semantic code (the hook from code 0 into the Shared Layers box in Figure 2), $\mathbf{1}[\cdot]$ is the indicator function, $a_t = 1$ if the input at position t is an audio frame and 0 if it is text, and $\mathbf{E}^{(k)}$ is the codebook-embedding table of Equation 1. The second term is the plain, unscaled sum of the embeddings of codes 2–9 (the chip stacked behind the backbone’s hidden state at the foot of each audio column in Figure 2). A shared layer has no output head, and its only output is the hidden state of Equation 2, which the layer above takes as its input,

$$\mathbf{h}_t^{(\ell)} = \text{Layer}^{(\ell)}(\mathbf{e}_t^{(\ell)}, \mathbf{h}_t^{(\ell-1)}), \quad \ell = 1, \dots, 4, \quad (4)$$

and $\mathbf{h}_t^{(4)}$ is the input of the first codebook-MTP layer. Together they fuse the backbone’s hidden state, the semantic information of the current frame, and the acoustic detail carried over from the previous frame into a single representation. All nine per-codebook layers then predict their codebooks from this one shared representation, with better temporal and acoustic context.

Per-codebook layers. The nine per-codebook layers (MTP Layer 1–9; $\ell = 4 + k$ for $k = 1, \dots, 9$) have the same structure plus an output head, with no weights shared across codebooks. MTP Layer k takes the hidden state from below and injects the code chosen just before it, codebook $k - 1$ of the frame:

$$\mathbf{e}_t^{(4+k)} = \mathbf{E}_c^{(k)}[q_{t+1}^{(k-1)}], \quad (5)$$

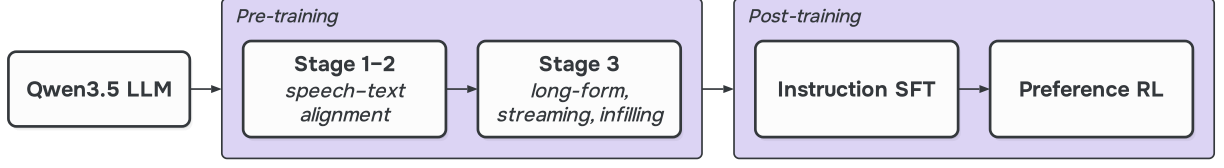


Figure 3: **Drama 3 training pipeline.** A pretrained Qwen3.5 LLM goes through three pre-training stages, then instruction SFT and preference RL.

where $\mathbf{E}_c^{(k)}$ is the layer’s own table (for $k = 1$ the code is code 0, the straight arrow past the shared box in Figure 2). Its hidden state follows Equation 2 as in the shared layers,

$$\mathbf{h}_t^{(4+k)} = \text{Layer}^{(4+k)}(\mathbf{e}_t^{(4+k)}, \mathbf{h}_t^{(3+k)}), \quad k = 1, \dots, 9, \quad (6)$$

starting from the shared layers’ output $\mathbf{h}_t^{(4)}$; unlike a shared layer, it has an output head, which turns that state into codebook k of the frame:

$$q_{t+1}^{(k)} \sim \text{softmax}(\mathbf{W}_{\text{out}}^{(k)} \mathbf{h}_t^{(4+k)}). \quad (7)$$

Each layer thus reads the code just chosen and emits the next, up to code 9, while its attention (Equation 2) still sees the whole sequence up to position t . Reference frames receive the same injections, with their loss switched off, so every layer sees all ten codebooks of the reference audio.

Training and inference. In training, every code is teacher-forced (Williams & Zipser, 1989) from the ground truth, so the decoder is one extra causal pass over the whole sequence, parallel over time. At inference, each frame takes one backbone step and then 13 single-position decoder steps, and the head of each per-codebook layer is sampled before the next layer runs (Section 4). The sampled $q^{(0)}$ and $q^{(1)}$ of frame $t + 1$ form the next backbone input (Equation 1); its codes 2–9 go to the first shared layer as in Equation 3.

Comparison with S2. Codec language models handle the codebook axis in one of three ways: by delaying or flattening codebooks in the main stream (Copet et al., 2023); with a small depth transformer inside each frame (Lee et al., 2022; Défossez et al., 2024; Liao et al., 2026); or with models that run over the whole sequence once per codebook stage, either non-autoregressively (Wang et al., 2023) or autoregressively (Wang et al., 2025a; Cremer & Cremer, 2026). S2’s Fast AR is of the second kind, a separate four-layer transformer that loops over one frame’s codebooks, conditioned on one projected backbone state (Liao et al., 2026). The codebook-MTP decoder is of the third kind but is one decoder rather than a chain of separate models. To our knowledge, it is the first whose per-codebook layers are full-width, attend causally along time over the backbone’s own sequence, and pass the hidden state upward as DeepSeek-V3’s MTP modules do. It differs from S2 in three ways.

- **Context.** It reads the fine codes of the reference audio and of all previous frames directly, whereas the Fast AR sees them only through the backbone state, so acoustic detail such as the reference’s timbre reaches every codebook prediction.
- **Training layout.** It runs on the backbone’s own sequence, reusing its rotary position embeddings (Su et al., 2021) and attention masks, so every codebook of every frame is trained in one extra causal pass, whereas the Fast AR is trained on separate per-frame codebook sequences.
- **Capacity.** It gives every codebook a full-width layer, embedding, head, and KV cache over the whole sequence of its own, whereas the Fast AR shares four layers across all codebooks and keeps only per-frame scratch state.

3 Training

Drama 3 is initialized from a pretrained Qwen3.5 language model (Qwen Team, 2026) with a newly initialized codebook-MTP decoder (Section 2) and trained in three pre-training stages (Section 3.1), one stage of supervised fine-tuning (SFT) on instructions (Section 3.2), and reinforcement learning (RL) from human

preference (Christiano et al., 2017) (Section 3.3) (Figure 3).

Conversation format. Every task is written as a conversation in the ChatML-style chat markup of the base model, in which the system turn holds the task instruction and the voice references for zero-shot cloning (Wang et al., 2023), the user turn holds the text to speak, and the assistant turn holds the audio. Loss covers only the assistant turns, so reference audio is context only. Stages 1 and 2 of pre-training concatenate the references as in S2; Stage 3 either concatenates them or gives each clip a numbered block pairing its speaker-tagged transcript with its audio, at random; and from SFT on, every sequence uses these numbered blocks, the indexed reference layout (Appendices A and B).

3.1 Pre-training

Data. Pre-training data is transcribed speech, all of which passes quality control first, and text documents, which keep the backbone’s language ability. In total, pre-training processes several hundred billion tokens. The language coverage is that of S2, about 80 languages and dialects (Liao et al., 2026).

Stages. Stages 1 and 2 teach the model to turn text into audio codes; their sequences are at most 8K tokens long and of two kinds, text documents and multi-turn TTS conversations (Figure 7). Stage 2 continues on the same sequences with the batch doubled (Smith et al., 2018), which lowers gradient noise, and leaves the learning-rate decay to Stage 3. Stage 3 extends the model to whole recordings, streaming input, and speech editing. It raises the maximum sequence length to 16K tokens and adds long-form synthesis of a whole recording from its transcript, as one segment (Figure 8) or line by line (Figure 9), streaming interleave (Figure 10), and speech infilling (Figure 14). Throughout, samples are packed with document masking (Zhao et al., 2024) into rows of 128K tokens, and the model is optimized with Muon (Jordan et al., 2024; Liu et al., 2025a).

Objective. Let $\mathcal{T}$ be the supervised text positions, including the closing $\langle |im_end| \rangle$ of each audio segment and the line separators, and $\mathcal{S}$ the supervised positions whose target is a semantic code $q_t^{(0)}$; then

$$\mathcal{L}_{\text{text}} = -\frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \log p_{\theta}(x_t | x_{<t}), \quad \mathcal{L}_{\text{sem}} = -\frac{1}{|\mathcal{S}|} \sum_{t \in \mathcal{S}} \log p_{\theta}(q_t^{(0)} | x_{<t}), \quad (8)$$

$$\mathcal{L}_{\text{cb}} = \frac{1}{9} \sum_{k=1}^9 \left(-\frac{1}{|\mathcal{S}|} \sum_{t \in \mathcal{S}} \log p_{\phi}(q_t^{(k)} | \mathbf{h}_{<t}, \mathbf{q}_{<t}, q_t^{(<k)}) \right), \quad \mathcal{L} = \mathcal{L}_{\text{text}} + 0.5 \mathcal{L}_{\text{sem}} + 0.5 \mathcal{L}_{\text{cb}}, \quad (9)$$

where p_{θ} is the backbone’s output distribution, p_{ϕ} the decoder (Section 2), $\mathbf{h}_{<t}$ the backbone hidden states at earlier positions, $\mathbf{q}_{<t}$ all ten codebooks of earlier frames, and $q_t^{(<k)}$ the lower codebooks of the current frame. Each term is a mean over its own bucket, so the text–speech balance does not drift with batch composition; the semantic and codebook terms share one per-frame mask, and $\mathcal{L}_{\text{cb}}$ back-propagates into the backbone. In pre-training and SFT, an auxiliary load-balancing loss (Lepikhin et al., 2020) keeps the MoE router from sending most tokens to a few experts.

3.2 Instruction SFT

A single SFT stage starts from the Stage-3 checkpoint and trains on sequences of up to 32K tokens in the indexed reference layout (Section 3, *Conversation format*). It teaches the model to follow directions and to perform dialogue and scenes. We choose the SFT recipe by the teacher-forced negative log-likelihood (NLL) of the semantic tokens of held-out real speech, each clip cloned from other clips of its voice, rather than by training loss.

Instruction formats. As in S2, directions are natural language in the user text, in the forms of Appendix C: point tags such as [laughs] or [pause], which insert a vocal event or a pause at that position; scoped tags such as $\langle \text{whisper} \rangle \dots \langle / \text{whisper} \rangle$, which set the delivery of the enclosed words only; style prefixes (Tian et al., 2026), which set the delivery of the whole line; and director’s notes on intent, pacing, or emphasis, each phrased in several templates and registers. Directing sessions are multi-turn. When a take is not what the director wanted, the next turn gives a note relative to it, such as [softer and warmer versus previous], and the model produces a new take, with the earlier takes kept in context as audio (Figure 12). Multi-speaker dialogue gives each voice one or more reference clips and switches voices in the user text with speaker

tokens (Figure 11). Whole-scene dubbing describes the characters in text and gives every line up front, joined by supervised separators, so that the delivery of a line can depend on the lines that follow (Figure 13). Plain cloning, long-form, infilling, and text-only chat sequences (Appendix A and Figure 14) stay in the mixture so that the pre-training abilities are kept.

3.3 Reinforcement Learning from Human Preference

RL starts from the SFT checkpoint and aligns Drama 3’s takes with human preference over expressive requests.

A single reward model. S2 combined three rewards: a transcription-error reward, a speech-quality model, and a speaker-similarity reward. The transcription-error reward compares what an automatic speech recognition (ASR) system hears with the script, but laughs and long pauses legitimately change what it hears. The speech-quality model scores the whole utterance at once, so it cannot reward fine-grained control. The speaker-similarity reward scores a whispered or shouted line lower against a neutral reference. Each of these rewards either penalizes the expressiveness we want or cannot see it, so Drama 3 scores every take with a single learned model of human preference and uses no other reward.

Reward model. The reward model (RM) is a smaller speech language model of our own, similar in design to Drama 3 and pre-trained with the same audio tokenizer and sequence format through the end of Stage 2, with a scalar head on the final hidden state of the take’s closing token. It reads exactly the policy’s prompt followed by the take’s audio tokens, so no tokenizer decoder, ASR, or speaker model runs in the reward path; because it sees the references and the script, voice consistency and content fidelity remain part of what it judges. It is trained on human preference judgments over pairs of takes with the Bradley–Terry loss (Bradley & Terry, 1952)

$$\mathcal{L}_{\text{RM}} = -[y \log \sigma(s_a - s_b) + (1 - y) \log \sigma(s_b - s_a)] + 10^{-3}(s_a^2 + s_b^2), \quad (10)$$

where s_a, s_b are the RM’s outputs for takes a and b , σ is the logistic function, and y is the preference label smoothed (Szegedy et al., 2015) as in conservative direct preference optimization (DPO) (Rafailov et al., 2023; Mitchell, 2023). Each raw score is standardized by a frozen anchor, clipped, and turned into an expected win rate among the n scored takes sampled for the same prompt:

$$z_i = \text{clip}\left(\frac{s_i - \mu_0}{\sigma_0}, -3, 3\right), \quad R_i = \frac{1}{n-1} \sum_{j \neq i} \sigma(z_i - z_j), \quad (11)$$

where s_i is the RM score of take i , and μ_0, σ_0 are a mean and a standard deviation fitted once on held-out takes and then kept fixed. With the anchor fixed, a drift of the whole batch stays visible in the mean z , and with the clip, a take gains nothing past three standard deviations. Here $\sigma(z_i - z_j)$ estimates the probability that take i beats take j , and the sum runs over the other $n - 1$ takes of the group, $j \neq i$, so R_i is the rate at which take i beats the rest of its group. The RM does not score a take with no audio, which stays in its group with reward zero and is left out of this comparison.

Sampling. For each prompt, the inference engine (Section 4) draws $G = 8$ takes at temperature 1 by plain ancestral sampling, with no top- p (Holtzman et al., 2020) or top- k (Fan et al., 2018) truncation, so that the takes follow exactly the probabilities the engine reports. Length is limited by a sampling budget rather than a reward. Each take may run to about 2.5 times the expected duration of its own script, and no take may exceed 4,096 frames, about 190 s, which bounds runaway takes.

Policy optimization. We use Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with the mean-centered advantages of Dr. GRPO (Liu et al., 2025b) and no KL penalty or reference model. Groups whose reward standard deviation is below 0.05 are dropped, a filter similar to the dynamic sampling of Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2025) but without re-sampling. With rewards $R_i \in [0, 1]$ (Equation 11),

$$A_i = R_i - \frac{1}{G} \sum_{j=1}^G R_j, \quad \mathcal{L}_{\text{RL}}(\theta) = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\left(\rho_{i,t} A_i, \text{clip}(\rho_{i,t}, 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) A_i\right), \quad (12)$$

where $\rho_{i,t} = \pi_{\theta}(y_{i,t} \mid x_i, y_{i,<t}) / \pi_{\text{eng}}(y_{i,t} \mid x_i, y_{i,<t})$ is the proximal policy optimization (PPO) ra-

tio (Schulman et al., 2017), x_i is the prompt of take i , π_θ is the probability that the backbone being trained gives the token, π_{eng} is the probability of the same token reported by the inference engine that sampled it, $\mathcal{B}$ holds the takes of the kept groups, and y_i is the sequence the backbone samples for take i , made of the semantic code $q_t^{(0)}$ of every frame and the closing $\langle |im_end| \rangle$. Unlike in S2, the codebook-MTP decoder, which produces the nine acoustic codes of each frame, stays frozen during RL and receives no RL loss. Because each rollout batch gets one optimizer step, the trainer’s weights at the update are the weights that sampled the batch, and the trainer routes each token to the experts that the engine chose, as in rollout routing replay (Ma et al., 2025), so ρ departs from 1 only through numerical train–inference mismatch (Qi et al., 2025). The rollout engines run the routed experts in block-scaled 8-bit floating point (FP8) (Micikevicius et al., 2022), re-quantized after every weight sync, while the trainer runs in bfloat16 (BF16), and this mismatch grows with that precision gap. We bound this departure with the asymmetric clip of DAPO, $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$. We keep GRPO’s per-take length normalization, which Dr. GRPO removes.

Large batches, few steps. Each step uses $1,536 \text{ prompts} \times G \text{ takes}$, drawn from diverse expressive requests that carry all of their reference clips, at a constant Muon learning rate without weight decay, and the schedule is short. Without a KL penalty or weight decay, few steps at a small constant learning rate bound how far the policy moves, and large batches average each step over many prompts, so a step follows broad preference rather than the particulars of a few prompts. In addition, this setup avoids the drift toward reward over-optimization (Gao et al., 2023a) that small-batch, many-step schedules showed in our RL development. Recent large language models such as MiMo-V2.6 (Xiaomi MiMo Team, 2026) likewise use very large RL batches.

4 Inference Engine

A streaming request, or stream, plays to its listener while it is generated, so its frames must arrive at least as fast as the codec plays them, about 21.5 per second (Section 2). Serving a request has two phases, a prefill that reads its text and reference audio in one pass, and decoding, one step per frame. One GPU serves many streams at once, in rounds that are each a prefill of new requests or one decode step for all running streams. We report the per-stream decode rate, the frames per second each stream receives, at a given concurrency, the number of streams a GPU serves at once.

Drama 3 differs from S2 in generation structure, model size, and serving load, and we rewrote the critical path of the inference engine to match. Its generation structure shortens the sequential path of each frame and reduces the weights it reads (Section 4.1). We remove the kernel launches and host waits that CUDA graphs (Gray, 2019), which replay a whole step at once, leave in place (Section 4.2). The kernels that stay are tuned to the shapes that serving produces (Section 4.3), and arriving requests no longer stall the streams already playing (Section 4.4).

4.1 Generation Structure

Streaming batches are usually small, so a step does little arithmetic per weight it reads: decoding is bound by memory bandwidth, and the time of a frame is set mainly by the weights it reads. In the Dual-AR design of S2, the four-layer Fast AR makes ten passes after the backbone step, so a frame runs 40 layer passes in sequence and reads the Fast AR’s weights again on every pass (Figure 4a). At batch size 1, a frame of S2 therefore reads about 15.6 GB of BF16 weights, two bytes per parameter, the equivalent of about 7.8B parameters, although S2 has about 4.6B. Drama 3 reads less for two reasons. First, its codebook-MTP decoder (Section 2.4) runs each of its 13 layers once per frame, so the decoder’s weights are read once (Figure 4b). Second, its routed experts are stored in block-scaled FP8, with one scale per 128×128 block, so each expert parameter takes one byte instead of two. This is the format in which the RL rollouts already run (Section 3.3), so serving adds no quantization of its own and runs with the numerics of RL sampling. A frame of Drama 3 thus reads about 6.0 GB of weights (Table 1), less than two fifths of what S2 reads, with more than seven times as many parameters.

At a concurrency of 1, Drama 3 decodes 271 frames per second on one NVIDIA H200 GPU (NVIDIA Corporation, 2026a), against 177 for S2 measured the same way. On one NVIDIA B300 GPU (Aubrey & Stam, 2025), it decodes 295 frames per second; there the experts use the microscaling FP8 format (MXFP8) (Open Compute Project, 2023), converted from the same FP8 weights. The one-byte experts matter at larger batches

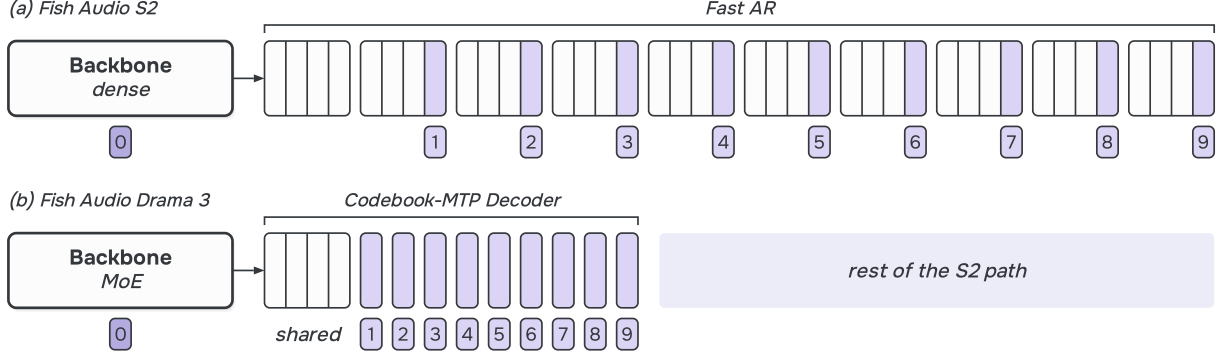


Figure 4: **Per-frame sequential path after the backbone step.** (a) S2’s four-layer Fast AR runs ten passes, one that takes in the backbone’s state and one for each of the nine acoustic codes, and reads its weights on every pass. (b) Drama 3’s codebook-MTP decoder runs its four shared and nine per-codebook layers once. Boxes are layer passes drawn to scale; each chip is the code emitted by the layer above it.

Table 1: **Decoding cost of Drama 3 and S2.** Weights read per frame are computed from the tensor sizes of the checkpoints.

Metric	Drama 3	S2
Layer passes in sequence after the backbone step	13	40
Weights read per frame (GB)	≈6.0	≈15.6
Backbone without routed experts	2.9	7.3
Eight active routed experts	1.0	–
Codebook-MTP decoder or Fast AR	2.1	8.3
Layers whose KV cache grows with the context (backbone and decoder)	23	36

too, where the time of a step still depends mainly on the bytes it moves: on the H200, the FP8 experts raise the per-stream decode rate at a concurrency of 32 by 36% over BF16 experts.

4.2 Kernel Launches Below CUDA Graphs

A step is a sequence of kernels, small programs that the host CPU launches on the GPU, and every kernel costs twice: the host pays to launch it, and the GPU pays a fixed cost to start and drain it. CUDA graphs record a whole step once and replay it without the host issuing kernels one by one, and programmatic dependent launch (PDL) (NVIDIA Corporation, 2026d) lets a kernel start while the kernel it depends on is still finishing; neither removes the fixed cost on the GPU. Before the changes below, we took a census of Drama 3 decoding a single stream on a B300, counting every kernel in a frame with CUDA graphs, PDL, and piecewise prefill graphs (the prefill captured in pieces) enabled. The median frame took 4.0 ms and ran 1,171 kernels, with only 0.09 ms of gaps between them, so the gaps that graphs and PDL target were already closed. Moving all the data the frame needs would take 0.76 ms at the GPU’s memory bandwidth, one fifth of the frame; the other four fifths went mostly to starting and draining kernels: about 830 kernels moved almost no data yet took 2.7 ms. That time shrinks neither with more bandwidth nor under graphs or PDL; the only remedy is fewer kernels.

Most of these 830 kernels come from auxiliary operations that a general-purpose serving framework adds around the computation, such as mask writes, tensor copies, layout changes, and type conversions. We trace every kernel in the graph back to the framework operator that produced it and decide whether its work is needed, then delete the kernel or fold its work into an adjacent compute kernel. Fewer kernels is not the goal in itself: the roughly 320 launches per step in the MoE layers are the expert computation and stay. A kernel that fused the router’s general matrix multiplication (GEMM), softmax, and top-8 selection into one launch took 14.5 μ s, against 7.5 μ s for the kernels it would have replaced, so it was rejected. Under PDL, saved kernel time also does not fully reach the wall clock: changes that saved 0.218 ms of kernel time per step gained 0.14–0.16 ms in serving. Every change is therefore accepted on a same-machine A/B comparison,

the old and the new engine serving the same load on the same GPU, and Table 2 lists the accepted changes.

Table 2: **Kernel launches and waits removed below CUDA graphs and PDL.** Gains are in the per-stream decode rate.

Change	Kernels or time	Gain	GPU, concurrency
Auxiliary decode kernels removed	1,429 $\rightarrow$ 1,278 kernels per step	+1.7%	B300, 32
Gated DeltaNet decode fused; sampler counter folded in	-100 kernels per step	+4.2%	B300, 1
Gated DeltaNet prefill launches merged	23 $\rightarrow$ 13 kernels per layer	+4.4%	B300, 128
First-frame sampling captured into CUDA graphs [†]	30.9 $\rightarrow$ 14.7 ms host time per round	+15.3%	H200, 32
Execution plan built from host-side lengths	9.75 $\rightarrow$ 0.86 ms GPU idle per round	$\approx$ +7%	B300, 128

[†]Measured on an earlier development model.

Auxiliary kernels in decode. Every MoE layer launched kernels to write a mask for the padding rows and to copy the shared expert’s input, every Gated DeltaNet layer copied the input of its gated normalization into a contiguous layout, and the routing weights were converted to another data type. Now the routing kernel writes the mask, the copy is made only when the shared expert computes in place, the normalization reads the strided input directly, and the conversion is gone: 1,429 kernels per decode step become 1,278, and the per-stream rate at a concurrency of 32 rises 1.7%. The Gated DeltaNet decode, previously three kernels and a gated RMSNorm per layer, is now one kernel that updates the convolution window, applies the gated delta rule, and normalizes, with bit-identical output. The sampler advances its random-number counter inside the sampling kernel instead of in a launch of its own. These two changes remove 100 kernels per step at every concurrency, and the per-stream rate at a concurrency of 1 rises 4.2%; the total per step still varies with concurrency, mainly through the attention kernels, which are chosen by batch size.

Auxiliary kernels in prefill. The Gated DeltaNet prefill operator launched 23 kernels per layer, most of them copies, reads and writes of per-request state, and index computations. It now reads the projection tensor by stride, writes the boundary states and initializes the state snapshots in one launch each, and computes its indices once per forward pass rather than in every layer: 13 launches per layer, and the per-stream rate at a concurrency of 128 rises 4.4%. Each prefill round also produces the first frame of every new request, through semantic sampling, the 13 decoder layers, and nine codebook samplings, about 100 kernels that the host launched one by one outside the graph. Capturing this chain into CUDA graphs, together with the Gated DeltaNet prefill chain, cut the host time per prefill round from 30.9 to 14.7 ms and raised the per-stream rate at a concurrency of 32 by 15.3% (H200, earlier development model); the Gated DeltaNet prefill operator now runs entirely inside the prefill graph.

A wait outside the graph. In each prefill round, the attention backend built its execution plan, the schedule of the round’s attention work, from indices copied back from the GPU, a copy that waited for all queued backbone work and left the GPU idle. We now build the plan from lengths that the host already tracks, and the GPU idle time per prefill round at a concurrency of 128 fell from 9.75 to 0.86 ms, a gain of about 7% in the per-stream rate. Removing the wait exposed a data race: the plan’s inputs reach the GPU by an asynchronous copy from a pinned host buffer, host memory the GPU reads directly, and the old wait had kept the next round from overwriting that buffer before the copy finished. Each workspace, the buffers of one round, now has a CUDA event, a marker the GPU records when it has finished reading, that fences the buffer with no waiting in steady state.

Rounds that fell out of the graph. A CUDA graph helps only a round that matches the form it was captured for. Every captured graph was guarded by a condition on the batch size or the number of new requests, and a round that failed the condition launched its kernels one by one. Nine requests arriving at once were enough to cause this fallback. The scheduler now admits only rounds that the prefill graphs can replay, and the largest captured decode batch equals the admission limit, the most streams the scheduler lets in, so every admitted round runs inside a graph.

4.3 Shape-Driven Kernel Optimization

General-purpose kernel libraries are tuned for generic matrix shapes, the sizes of the matrices a kernel multiplies. The shapes in TTS serving are set by the workload: decode batches are small and fall into a few sizes, the matrices are thin, the sampler touches only a small slice of the vocabulary, and the Gated DeltaNet

geometry is fixed by the model. A kernel tuned for shapes that serving never produces gains nothing there, so kernel optimization starts from the serving workload and iterates a loop of five steps (Figure 5).

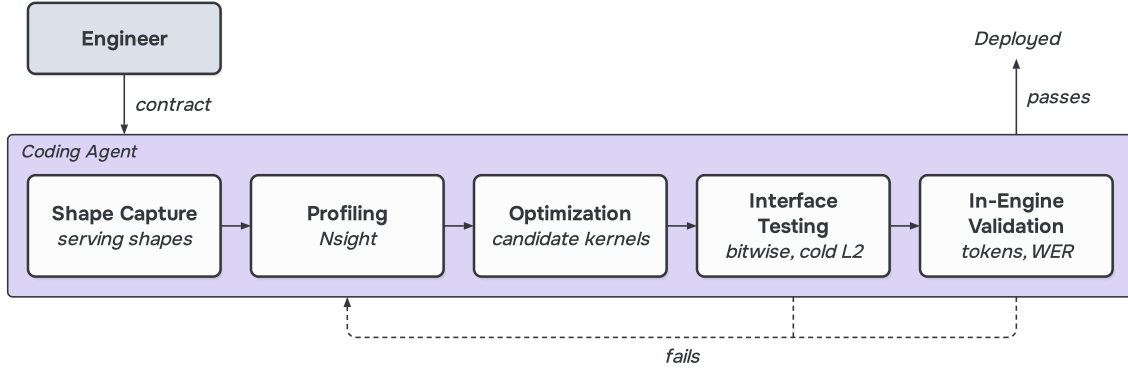


Figure 5: **Shape-driven kernel optimization loop.** An engineer writes the contract for a round; a coding agent captures the serving shapes, profiles, writes candidate kernels, and tests them, first on the operator interface and then inside the engine. A candidate that fails returns to profiling, and one that passes is deployed. WER: word error rate.

Shape capture reads the GEMM dimensions (M, N, K) and batch sizes of every operator from serving logs and graph replays; M , the number of rows, is the batch size in decode. At a concurrency of 32, for example, 91.8% of decode batches have 31 or 32 rows, so a kernel tuned only for $M \leq 16$ gains almost nothing in serving. *Profiling* attributes the time of each frame to its kernels with the profiler NVIDIA Nsight Systems (NVIDIA Corporation, 2026c) and measures with Nsight Compute (NVIDIA Corporation, 2026b) how far each kernel is from the roofline (Williams et al., 2009), the ceiling set by the GPU’s memory bandwidth or arithmetic rate. In a single-operator test at Drama 3’s serving dimensions and batch size 256, for example, 97% of the memory traffic of the Gated DeltaNet decode kernel is the full read and write of the recurrent state, already at 81% of peak bandwidth. Faster arithmetic cannot help such a kernel; only fewer bytes can, so on the B300 the state is stored in BF16. *Optimization* writes candidate kernels from scratch or adapts upstream ones. *Interface testing* checks the operator alone for bitwise agreement with a reference and times it with a cold L2 cache, one that does not already hold the weights, so that the cache does not flatter small batches. To keep the cache cold, the test rotates weight copies of at least twice the capacity of the L2, the GPU’s on-chip cache, inside the graph. *In-engine validation* accepts a candidate inside the inference engine on token-level agreement and on the word error rate (WER, Section 5) at every concurrency level, and the serving A/B comparison decides deployment.

The last two steps are kept apart because their verdicts often disagree; we recorded nine such cases. The Gated DeltaNet prefill kernel, for example, was 3.4 times faster as a kernel from the start but lost to the existing one in serving three times, because of graph capture, state movement, and a cold compile at run time. Once these were resolved, it became the default set and raised throughput at a concurrency of 256 by 11.1%.

Each round of the loop is executed by a coding agent in its own workspace, a separate copy of the code, under a written contract in which an engineer sets the goal, constraints, validation commands, and deployment criteria. A second agent re-derives each conclusion from the primary evidence, and only candidates that pass every check are deployed. The FP8 MoE kernels of Drama 3 were developed and iterated in this loop with the Kernel Design Agents workflow (NVIDIA Research, 2026). The FP8 MoE kernel cuts the MoE layer time at $M = 256$ by 19.2% and raises the per-stream rate in serving at a concurrency of 1 by 3.0%.

4.4 Prefill–Decode Disaggregation

The changes so far make each frame cheaper; this one keeps new requests from interrupting the streams already playing, since decoding that falls behind playback is heard as a stall. When one GPU serves both phases, every arriving request makes the scheduler insert a prefill round, and during that round no running stream emits a frame (Figure 6a). On single-GPU workers, these rounds were the main cause of interrupted

streams at high concurrency. A request whose reference audio is already cached needs only a short prefill, but at high request rates such prefills arrive very often, and the longer prefill of a cold request, whose reference audio is not cached, stalls every stream at once.

Prefill-decode disaggregation (PD) (Zhong et al., 2024; Patel et al., 2024) runs the two phases on different GPUs (Figure 6b). A PD unit consists of a load balancer, a prefill process, and a decode process.

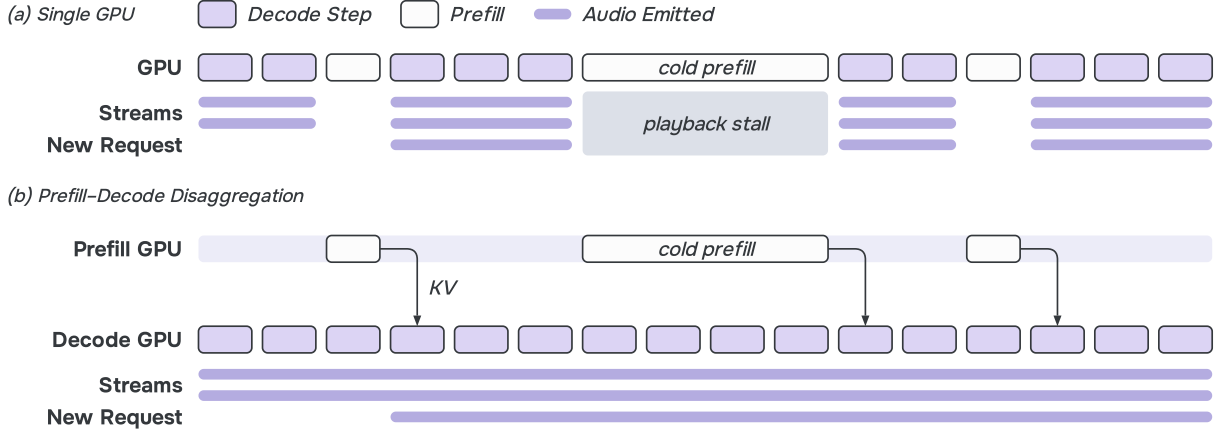


Figure 6: **Schematic timelines of streaming on one GPU (a) and with prefill-decode disaggregation (b).** On one GPU, every prefill pauses all running streams, and the long prefill of a cold request, one whose reference audio is not cached, causes a playback stall. With disaggregation, the prefill GPU runs the same prefills, each finished prefill writes its KV and recurrent state into the decode GPU, and the streams play without gaps.

When a prefill finishes, its KV entries and its Gated DeltaNet states are written within the node straight into the paged KV cache (Kwon et al., 2023) and the state buffers of the decode process, in packed transfers with an end-to-end checksum. The transfer uses NVLink fabric handles on B300 and a PCIe peer-to-peer proxy on NVIDIA RTX PRO 6000 GPUs (NVIDIA Corporation, 2026e). Each role, prefill or decode, captures only the CUDA graphs it needs, so a unit becomes ready in 47 s instead of 148 s.

PD already runs in the production service of Drama 3 on RTX PRO 6000 GPUs. A capacity test on one RTX PRO 6000 node applies a closed-loop streaming load (each finished stream is replaced by a new one) that reproduces the production mix. The bar is no playback stall and a per-stream rate of at least 35 frames per second at the first percentile, that is, for all but the slowest one percent of streams. Under this bar, a unit of one prefill and one decode GPU carries 24 streams per GPU, averaged over both, against about 10 for a single GPU. In a production canary, a trial on live traffic with PD units and single-GPU workers behind the same router, the PD decode GPUs carry about twice as many streams as the single-GPU workers. The first percentile of their per-stream rates is 55 frames per second, against 45–47 for the single-GPU workers. No PD stream falls below the service floor of 42 frames per second, whereas a small share of single-GPU streams does. The character error rate (CER, Section 5) of PD streams is equivalent to that of single-GPU serving, a difference of 0.02 points.

5 Evaluation

We evaluate Drama 3 on public benchmarks. Multilingual cloning is measured on the MiniMax Multilingual Testset (Zhang et al., 2025). Editing by infilling is measured on Ming-Freeform-Audio-Edit (Yan et al., 2025a), and revision from a note in the next turn on the Step-Audio-Edit benchmark (Yan et al., 2025b). All results come from a single Drama 3 model, whereas most baselines are built for one kind of task. We also analyze what the single preference reward changes (Section 5.2) and measure cloning from several references and streaming synthesis.

5.1 Evaluation Setup

Drama 3 is run with the prompt formats of Appendix B and with each benchmark’s own reference audio, reference text, and target text, sampling at temperature 0.7 with top- p 0.7 and top- k 30. Outputs that fail simple signal checks, such as silent output, too little speech, runaway length, or long stretches of noise or silence, are regenerated and rescored. Baseline numbers are quoted from their papers, except where a table marks a baseline as run by us. Unless a section or table states otherwise, error rates follow each benchmark’s official protocol, with the character error rate (CER) for Chinese, Japanese, and Korean and the word error rate (WER) for the other languages. Unless stated otherwise, we transcribe Chinese with Paraformer-zh (Gao et al., 2023b) and the other languages with Whisper-large-v3 (Radford et al., 2023), and speaker similarity (SIM) is the cosine similarity of the speaker embeddings of WavLM-large (Chen et al., 2022) with an ECAPA-TDNN head (Desplanques et al., 2020), the speaker model of Seed-TTS-Eval. Where the conditions of Drama 3 and the baselines differ on a benchmark, we state the difference where the benchmark is introduced.

5.2 Preference Reward vs. Rule-Based Rewards

Reinforcement learning in Drama 3 optimizes a single learned reward of human preference and uses no transcription or speaker-similarity reward (Section 3.3). The rewards it leaves out penalize delivery in the requested style. In Espresso (Nguyen et al., 2023), each of four speakers reads the same sentences in a default style and in several other styles. We pair every styled reading with the same speaker’s default reading of the same sentence, so the two readings of a pair differ only in style. With one default-style recording of each speaker as the SIM reference, every style lowers SIM below the default reading, by 0.019 for clearly enunciated reading and by 0.310 for whispering, so a similarity reward prefers the default reading in 62.0% to 99.7% of the pairs, depending on the style (Table 3). Reading with a laugh raises WER by 0.79 points, so a WER reward penalizes laughing.

The reward model of Drama 3 instead responds to the direction. The voice reference is a default-style clip of the same speaker. We score both readings of each pair with the reward model twice, once with the style written as a direction in the syntax of Appendix C, such as <whisper>...</whisper> or [laughing], and once with the sentence alone. With the direction, the reward model prefers the whispered reading in 62.2% of pairs and the laughing reading in 64.4%, against 10.6% and 27.1% without it, whereas a similarity reward prefers the default reading in 99.7% and 93.9% of these pairs. In every style, the direction raises the share of pairs in which the reward model prefers the styled reading (Table 3).

Table 3: **Styled versus default human readings in Espresso.** Δ SIM and Δ WER are means over pairs, styled minus default, Δ WER in percentage points. SIM is to one default-style recording per speaker. SIM prefers default is the share of pairs in which the default reading has the higher SIM. Reward model prefers styled is the share of pairs in which our reward model gives the styled reading the higher score; with direction, the style is written into the script as a direction, and without direction, the script is the sentence alone.

Style	Pairs	Δ SIM	Δ WER (points)	SIM prefers default (%)	Reward model prefers styled (%)	
					With direction	Without direction
Whisper	1,439	−0.310	+0.24	99.7	62.2	10.6
Laughing	1,423	−0.193	+0.79	93.9	64.4	27.1
Sad	1,432	−0.178	+0.34	96.1	54.3	13.1
Confused	1,372	−0.107	−0.61	89.3	26.2	16.3
Happy	1,436	−0.063	−0.16	77.9	38.5	29.5
Clearly enunciated	1,447	−0.019	−1.06	62.0	8.1	6.6

A preference reward alone also cuts the failures that rule-based rewards are usually added to catch. We serve the SFT checkpoint and Drama 3, the policy after RL, through the same inference engine, sampling settings, and tokenizer decoder, and synthesize Seed-TTS-Eval (Anastassiou et al., 2024; ByteDance Speech, 2024) with each. The first take is the output before the regeneration of Section 5.1, the final take is the output after it, and a first take is flagged when it fails the signal checks. Without any transcription reward, RL cuts the share of flagged first takes from 2.82% to 1.24% on test-zh and from 3.40% to 1.84% on test-en, and the first-take

error rates fall with it, from 1.66 to 1.00 CER and from 3.49 to 2.20 WER (Table 4). After the regeneration replaces the flagged takes, Drama 3 still has the lower error rate, 0.83 against 0.84 CER on test-zh and 1.54 against 1.60 WER on test-en, so leaving out the transcription reward costs nothing in intelligibility on the takes that pass the checks either.

Table 4: **SFT checkpoint and Drama 3 on Seed-TTS-Eval.** First take is the output before the regeneration of Section 5.1, Final the output after it, and Flagged the share of first takes that fail the signal checks. **Bold:** better model.

Model	test-zh CER (%) ↓		test-en WER (%) ↓		Flagged (%) ↓	
	First take	Final	First take	Final	test-zh	test-en
SFT checkpoint	1.66	0.84	3.49	1.60	2.82	3.40
Fish Audio Drama 3	1.00	0.83	2.20	1.54	1.24	1.84

5.3 Zero-Shot Voice Cloning

5.3.1 Cloning from Multiple References

Drama 3 can clone one voice from several reference clips, each in its own [Reference N] block (Appendix B). No public benchmark reports multi-reference cloning, so we extend the LibriSpeech-PC (Meister et al., 2023) test-clean list of F5-TTS (Chen et al., 2025), 1,127 sentences from 39 speakers, with two more clips of each target speaker that never include the target sentence. We score the outputs as OmniVoice (Zhu et al., 2026) scores itself and the systems it compares on this list (Table 5). With three references instead of one, SIM to the benchmark’s reference clip rises from 0.605 to 0.633. With one reference, its UTMOS (Saeki et al., 2022) of 4.31 is above the 4.28 of OmniVoice and CosyVoice 3.

Table 5: **Cloning from one or three references on LibriSpeech-PC test-clean.** The other systems use one reference and are quoted from OmniVoice. Scored as in OmniVoice, with HuBERT-large (Hsu et al., 2021) WER, UTMOS (Saeki et al., 2022), and SIM to the first reference. **Bold:** best per column.

System	WER (%) ↓	SIM ↑	UTMOS ↑
Fish Audio Drama 3, one reference	1.85	0.605	4.31
Fish Audio Drama 3, three references	1.85	0.633	4.31
OmniVoice (Zhu et al., 2026)	1.30	0.729	4.28
Qwen3-TTS (Hu et al., 2026)	1.60	0.704	4.41
CosyVoice 3 (Du et al., 2025)	1.59	0.694	4.28

5.3.2 Multilingual Cloning

The MiniMax Multilingual Testset (Zhang et al., 2025) covers 24 languages. We transcribe the outputs with the models of Section 5.1 and compute the error rates with the public evaluation script of OmniVoice (Zhu et al., 2026); ElevenLabs v4, Inworld TTS-2, and Soniox TTS RT v2 are run by us and scored the same way, while OmniVoice, FireRedTTS3-Base, and Fish Audio S2 are quoted from their papers, which used their own scoring. Table 6 reports the error rates. Its error rate is lower than that of OmniVoice in 15 languages, among them Chinese, English, French, German, Korean, and Thai, and lower than that of FireRedTTS3-Base (Shen et al., 2026) in 17. It has a lower error rate than OmniVoice, FireRedTTS3-Base, and Fish Audio S2 (Liao et al., 2026) in eleven languages: Arabic, Chinese, Dutch, English, Finnish, French, German, Polish, Portuguese, Turkish, and Vietnamese. Against the systems we run ourselves, it is lower than Inworld TTS-2 in 18 languages, than Soniox TTS RT v2 in 21 of the 23 it supports, and than ElevenLabs v4 in Chinese, Dutch, English, German, Korean, Russian, Spanish, Thai, and Turkish; in Chinese, Dutch, English, German, and Turkish it is lower than every other system in the table.

Table 6: **MiniMax Multilingual Testset, error rate (%)**. ElevenLabs v4 (ElevenLabs, 2026), Inworld TTS-2 (Inworld AI, 2026), and Soniox TTS RT v2 (Soniox, 2026) are run by us through their public APIs on the same items. OmniVoice, FireRedTTS3-Base, and Fish Audio S2 are quoted from their papers. **Bold**: lowest per row.

Language	Metric	Drama 3	OmniVoice	FireRedTTS3-Base	Fish Audio S2	ElevenLabs v4	Inworld TTS-2	Soniox TTS RT v2
Arabic	CER	0.76	1.39	1.75	3.50	0.71	1.39	2.30
Cantonese	CER	22.70	17.71	40.32	30.67	19.74	24.14	–
Chinese	CER	0.63	1.01	0.91	0.73	0.66	0.97	1.32
Czech	WER	3.53	2.86	3.17	2.84	2.13	3.95	8.36
Dutch	WER	0.59	1.36	1.15	0.99	1.00	1.53	2.01
English	WER	1.13	1.56	2.12	1.62	1.56	1.56	1.94
Finnish	WER	3.09	3.75	3.10	3.33	2.43	4.74	4.14
French	WER	2.68	3.35	5.28	3.05	2.63	2.63	4.12
German	WER	0.48	0.96	0.69	0.55	0.90	0.83	1.10
Greek	CER	3.39	1.06	1.24	5.74	0.56	0.98	1.33
Hindi	CER	6.35	4.33	7.02	14.64	5.78	7.16	8.76
Indonesian	WER	4.93	1.97	1.42	1.46	1.59	1.67	5.08
Italian	WER	1.52	2.07	2.28	1.27	1.29	1.76	2.31
Japanese	CER	5.02	4.03	3.60	2.76	3.66	4.23	5.36
Korean	CER	2.36	2.65	2.42	1.18	2.63	2.65	3.43
Polish	WER	0.82	0.87	1.22	1.26	0.77	1.39	2.62
Portuguese	WER	1.11	2.51	1.79	1.14	1.11	1.79	1.40
Romanian	WER	4.94	2.42	1.93	10.74	1.60	2.86	4.63
Russian	WER	2.23	2.23	3.28	2.40	2.32	2.65	4.05
Spanish	WER	0.96	1.03	1.21	0.91	1.09	0.90	1.67
Thai	CER	2.27	3.98	1.87	4.23	3.08	4.19	5.47
Turkish	WER	0.52	2.17	0.92	0.87	0.87	1.82	1.91
Ukrainian	WER	2.10	1.77	0.55	2.30	1.37	2.18	3.06
Vietnamese	CER	0.80	1.37	0.86	7.41	0.61	1.25	3.36

5.4 Streaming Synthesis

Each sentence is synthesized on its own in two modes: streamed, where its audio is sent out chunk by chunk while the rest is still being generated, and non-streamed, where the whole take is sent out once it is finished. We do this for all 1,088 Seed-TTS-Eval test-en sentences. Streaming costs no quality: on the same sentences, streamed takes reach a WER of 1.86 and non-streamed takes 1.83, and both reach a SIM of 0.640.

5.5 Editing by Infilling

Drama 3 edits a recording by infilling (Le et al., 2023; Peng et al., 2024). We align the source audio to its transcript with Qwen3-ForcedAligner-0.6B (Shi et al., 2026). The edit window runs from the first to the last word that differs between the source and target texts, with characters as words in Chinese. A deletion leaves no new text, so its window also covers the unchanged word before it, or after it at the start of the clip. Drama 3 regenerates the audio inside the window with the infilling prompt of Figure 14, which holds the full target text and the audio tokens of the source before and after the window. The whole token sequence, with the new tokens inside the window, is then decoded to audio.

Ming-Freeform-Audio-Edit, which we shorten to Ming-Freeform, asks for deletions, insertions, and substitutions in English and Chinese. Each item gives a source recording, its transcript, an edit instruction, and the target text, and Drama 3 edits from the target text. The basic split has instructions of limited variety, and the larger full split, which the benchmark’s evaluation script calls open, has more natural and varied instructions and more substitutions. We score the outputs with the protocol of CosyEdit (Chen et al., 2026b) and CosyEdit2 (Chen et al., 2026a), which reports WER, SIM against the source recording, and MAE_D. MAE_D is the mean absolute difference between the DNSMOS P.835 overall score (Reddy et al., 2021) of the edited clip and that of its source, and it is small when the edit leaves the perceived quality of the recording unchanged. Table 7 compares Drama 3 on the basic split with VoiceCraft-X (Zheng et al., 2025), LEMAS-Edit (Zhao et al., 2026), ECPA (Ren et al., 2026), SSR-Speech (Wang et al., 2024), and CosyEdit2, as reported by CosyEdit2. Among them, SSR-Speech reconstructs the unedited spans inside its decoder, which takes the source waveform as an additional input, and VoiceCraft-X decodes the source tokens together with the generated ones; the Drama 3 rows here are likewise decoded whole, without pasting the source

recording back. In Chinese, Drama 3 has the highest SIM on insertion and substitution, 0.909 and 0.898 against 0.89 for CosyEdit2, the lowest WER on deletion, 7.82 against 8.23, and the smallest MAE_D on deletion and substitution.

Table 7: **Ming-Freeform basic split under the CosyEdit protocol.** WER in %. Baselines are quoted from CosyEdit2; – marks a metric not reported. **Bold**: best per column within each language.

System	Insertion			Deletion			Substitution		
	WER ↓	SIM ↑	MAE _D ↓	WER ↓	SIM ↑	MAE _D ↓	WER ↓	SIM ↑	MAE _D ↓
<i>English</i>									
Fish Audio Drama 3	3.03	0.905	0.208	5.79	0.838	0.174	2.08	0.866	0.181
VoiceCraft-X (Zheng et al., 2025)	5.74	0.85	0.166	9.71	0.80	0.172	7.29	0.81	0.155
LEMAS-Edit (Zhao et al., 2026)	6.04	0.81	0.158	9.29	0.80	0.167	6.09	0.80	0.167
ECPA (Ren et al., 2026)	4.50	0.82	–	6.91	0.77	–	4.13	0.78	–
SSR-Speech (Wang et al., 2024)	1.75	0.94	0.139	5.22	0.91	0.132	1.90	0.89	0.146
CosyEdit2 (Chen et al., 2026a)	1.90	0.93	0.107	5.52	0.90	0.131	1.43	0.89	0.137
<i>Chinese</i>									
Fish Audio Drama 3	2.15	0.909	0.113	7.82	0.827	0.105	1.87	0.898	0.100
VoiceCraft-X (Zheng et al., 2025)	3.77	0.84	0.128	13.62	0.77	0.203	4.21	0.82	0.158
LEMAS-Edit (Zhao et al., 2026)	6.21	0.81	0.130	17.43	0.75	0.159	8.16	0.82	0.157
CosyEdit2 (Chen et al., 2026a)	1.38	0.89	0.113	8.23	0.85	0.166	1.26	0.89	0.110

We also run Cartesia Sonic-3 (Cartesia, 2026) through its infill API, which takes the audio before and after a window and the new text of the window and returns speech for the window. Cartesia gets the edit windows of Drama 3 and a voice cloned from the source. Because its API returns only the audio of the window, the new audio of both systems is pasted into the source recording here, matched in loudness, with a 20 ms equal-power crossfade at each joint (Table 8). Drama 3 has the lower WER on both splits in both languages, and in Chinese its WER is less than half of Cartesia’s, 3.69 against 8.64 on the basic split and 2.35 against 6.53 on the full split.

Table 8: **Ming-Freeform WER (%) against Cartesia Sonic-3.** Each cell averages deletion, insertion, and substitution, with the new audio pasted into the source recording. Cartesia is run by us through its public API. **Bold**: lowest per column.

System	Basic		Full	
	English	Chinese	English	Chinese
Fish Audio Drama 3	3.49	3.69	3.43	2.35
Cartesia Sonic-3, infill (Cartesia, 2026)	4.58	8.64	4.68	6.53

5.6 Multi-Turn Revision

The Step-Audio-Edit benchmark (Yan et al., 2025b) clones a voice and then asks for up to three rounds of revision toward a target emotion or speaking style, or for one round that adds a paralinguistic event, such as a laugh or a breath, at a marked position. Iteration 0 is plain cloning, and each later iteration revises the most recent take. Gemini-2.5-Pro (Comanici et al., 2025) judges every take with the official prompts, and we score its answers with the official scripts. For emotion and style, the judge names the one emotion or style it hears, and the accuracy is the share of takes named with their target. The emotion labels are the five targets and a neutral one. For style, the official script counts only the takes named with one of the seven target styles. Where the official script has no valid takes for a speaker and a style, that pair is left out of the average. For the paralinguistic event, the judge gives a score from 1 to 3, where 1 means the event is missing or misplaced, 2 means it is present with a flaw such as a faint sound or an audible join, and 3 means it is clear, in place, and seamless. Drama 3 revises with a note in the next turn, such as [happier versus previous], and keeps the original reference and the most recent take in context, which corresponds to the Prompt-Fixed setting of the Step-Audio-EditX paper, so we compare with Step-Audio-EditX in that setting. Paralinguistic events are added by inserting a point tag such as [coughs] at the target position and infilling the audio there.

Table 9 shows that every revision round raises the emotion and style accuracy of Drama 3, from 55.5 to 58.2 on emotion and from 76.5 to 81.8 on style over three rounds. Adding the event with a point tag and infilling lifts its paralinguistic score from 1.32 to 2.32.

Table 9: **Step-Audio-Edit benchmark.** Step-Audio-EditX is quoted from the Prompt-Fixed setting of its paper, with the judge scores reported there; Drama 3 is judged by our own run of the official Gemini-2.5-Pro protocol. Emotion and style are accuracies (%). **Bold:** better system.

Metric	System	Iter. 0	Iter. 1	Iter. 2	Iter. 3
Emotion ↑	Fish Audio Drama 3	55.5	57.2	57.5	58.2
	Step-Audio-EditX (Yan et al., 2025b)	53.6	66.8	68.7	69.3
Style ↑	Fish Audio Drama 3	76.5	79.4	80.5	81.8
	Step-Audio-EditX (Yan et al., 2025b)	45.6	62.7	64.2	63.8
Paralinguistic ↑	Fish Audio Drama 3	1.32	2.32	–	–
	Step-Audio-EditX (Yan et al., 2025b)	1.91	2.89	–	–

6 Conclusion

In this report, we present Fish Audio Drama 3, a speech generation model that turns text into directed performances. It clones voices from multiple references, follows natural-language direction, revises a take when the next turn gives a note, performs multi-character scenes, edits existing speech by infilling, and synthesizes long-form and streaming audio. These capabilities rest on three design choices. A hybrid linear-attention mixture-of-experts backbone is paired with a codebook multi-token-prediction decoder (Section 2) whose layers attend over the whole sequence, so acoustic detail is predicted with direct access to the reference audio and all previous frames. Multi-turn revision, scene dubbing, editing by infilling, and streaming text input are supported by one model, each written as a conversation over the same token stream (Section 3, *Conversation format*). Reinforcement learning (Section 3.3) uses a single learned human-preference reward and no rule-based rewards, which would penalize laughs, pauses, and whispers. Serving these capabilities is the work of the inference engine (Section 4), which runs the codebook-MTP decoder once per frame and the experts in the block-scaled FP8 of the RL rollouts, removes the kernel launches that CUDA graphs leave in place, optimizes kernels on the shapes of live serving, and separates prefill from decode so that new requests do not stall running streams. On one H200, Drama 3 decodes a single stream at 271 frames per second against 177 for S2, with more than seven times as many parameters. We evaluate this single model on public benchmarks. We hope Fish Audio Drama 3 serves as an open foundation for expressive, directable speech generation.

7 Author

Core Contributors

The core contributors are listed in alphabetical order by last name:

Dawei Han, Shidong Li, Shijia Liao, Jiahua Liu, Songting Liu, Qingzheng Wang, Yuxuan Wang

Contributors

The contributors are listed in alphabetical order by last name:

Xin Chen, Yifan Cheng, Tianyu Li, Gangin Park, Xuehao Yang, Chang Zeng, Haoyang Zhang, Ruoyi Zhang, Xiangyu Zhang

For correspondence, please contact oss@fish.audio.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety, 2016. URL <https://arxiv.org/abs/1606.06565>.

-
- Philip Anastassiou, Jiawei Chen, Jitong Chen, Yuanzhe Chen, Zhuo Chen, Ziyi Chen, Jian Cong, Lelai Deng, Chuang Ding, Lu Gao, Mingqing Gong, Peisong Huang, Qingqing Huang, Zhiying Huang, Yuanyuan Huo, Dongya Jia, Chumin Li, Feiya Li, Hui Li, Jiaxin Li, Xiaoyang Li, Xingxing Li, Lin Liu, Shouda Liu, Sichao Liu, Xudong Liu, Yuchen Liu, Zhengxi Liu, Lu Lu, Junjie Pan, Xin Wang, Yuping Wang, Yuxuan Wang, Zhen Wei, Jian Wu, Chao Yao, Yifeng Yang, Yuanhao Yi, Junteng Zhang, Qidi Zhang, Shuo Zhang, Wenjie Zhang, Yang Zhang, Zilin Zhao, Dejian Zhong, and Xiaobin Zhuang. Seed-TTS: A family of high-quality versatile speech generation models, 2024. URL <https://arxiv.org/abs/2406.02430>.
- Kyle Aubrey and Nick Stam. Inside NVIDIA Blackwell Ultra: The chip powering the AI factory era. NVIDIA Technical Blog, 2025. URL <https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era/>.
- Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle, 2022. URL <https://arxiv.org/abs/2207.14255>.
- Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL <https://arxiv.org/abs/1506.03099>.
- Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. doi: 10.2307/2334029.
- ByteDance Speech. seed-tts-eval. GitHub repository, 2024. URL <https://github.com/BytedanceSpeech/seed-tts-eval>.
- Cartesia. Cartesia API reference: Infill. API documentation, accessed 2026-10-09, 2026. URL <https://docs.cartesia.ai/api-reference/infill/bytes>.
- Junyang Chen, Yuhang Jia, Hui Wang, Jiaming Zhou, Yongchang Gan, and Yong Qin. CosyEdit2: Speech-editing-oriented reinforcement learning unlocks better zero-shot TTS, 2026a. URL <https://arxiv.org/abs/2605.25930>.
- Junyang Chen, Yuhang Jia, Hui Wang, Jiaming Zhou, and Yong Qin. CosyEdit: Unlocking end-to-end speech editing capability from zero-shot text-to-speech models, 2026b. URL <https://arxiv.org/abs/2601.05329>.
- Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, et al. WavLM: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1505–1518, 2022.
- Yushen Chen, Zhikang Niu, Ziyang Ma, Keqi Deng, Chunhui Wang, Jian Zhao, Kai Yu, and Xie Chen. F5-TTS: A fairytaler that fakes fluent and faithful speech with flow matching. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6255–6271, 2025. doi: 10.18653/v1/2025.acl-long.313.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.
- Gheorghe Comanici et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL <https://arxiv.org/abs/2507.06261>.
- Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation. In *Advances in Neural Information Processing Systems*, 2023. URL <https://arxiv.org/abs/2306.05284>.
- Fritz Cremer and Jonathan Cremer. TontaubeV1: Streaming text-to-speech with hierarchical codec modeling and bounded context, 2026. URL <https://arxiv.org/abs/2609.08703>.

-
- Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL <https://arxiv.org/abs/2401.06066>.
- DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, et al. DeepSeek-V3 technical report, 2024. URL <https://arxiv.org/abs/2412.19437>.
- Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue. *arXiv preprint arXiv:2410.00037*, 2024.
- Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. ECAPA-TDNN: Emphasized channel attention, propagation and aggregation in TDNN based speaker verification. In *Interspeech 2020*, pp. 3830–3834, 2020. doi: 10.21437/Interspeech.2020-2650.
- Zhihao Du, Changfeng Gao, Yuxuan Wang, Fan Yu, Tianyu Zhao, Hao Wang, Xiang Lv, Hui Wang, Chongjia Ni, Xian Shi, Keyu An, Guanrou Yang, Yabin Li, Yanni Chen, Zhifu Gao, Qian Chen, Yue Gu, Mengzhe Chen, Yafeng Chen, Shiliang Zhang, Wen Wang, and Jieping Ye. CosyVoice 3: Towards in-the-wild speech generation via scaling-up and post-training, 2025. URL <https://arxiv.org/abs/2505.17589>.
- ElevenLabs. ElevenLabs models. Model documentation, accessed 2026-10-09, 2026. URL <https://elevenlabs.io/docs/overview/models>.
- Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1082. URL <https://arxiv.org/abs/1805.04833>.
- Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023a. URL <https://proceedings.mlr.press/v202/gao23h.html>.
- Zhifu Gao, Zerui Li, Jiaming Wang, Haoneng Luo, Xian Shi, Mengzhe Chen, Yabin Li, Lingyun Zuo, Zhihao Du, Zhangyu Xiao, and Shiliang Zhang. FunASR: A fundamental end-to-end speech recognition toolkit, 2023b. URL <https://arxiv.org/abs/2305.11013>. Interspeech 2023.
- Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR, 2024. URL <https://proceedings.mlr.press/v235/gloeckle24a.html>.
- Alan Gray. Getting started with CUDA graphs. NVIDIA Technical Blog, 2019. URL <https://developer.nvidia.com/blog/cuda-graphs/>.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL <https://arxiv.org/abs/1904.09751>.
- Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. HuBERT: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.
- Hangrui Hu, Xinfu 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. Qwen3-TTS technical report, 2026. URL <https://arxiv.org/abs/2601.15621>.
- Inworld AI. TTS models: Inworld TTS-2. Model documentation, accessed 2026-10-09, 2026. URL <https://docs.inworld.ai/tts/tts-models>.
- Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. <https://kellerjordan.github.io/posts/muon/>, 2024. Blog post.

-
- Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL <https://proceedings.mlr.press/v119/katharopoulos20a.html>.
- Rithesh Kumar, Prem Seetharaman, Alejandro Luebs, Ishaan Kumar, and Kundan Kumar. High-fidelity audio compression with improved RVQGAN. *Advances in Neural Information Processing Systems*, 36: 27980–27993, 2023.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023. URL <https://arxiv.org/abs/2309.06180>.
- Matthew Le, Apoorv Vyas, Bowen Shi, Brian Karrer, Leda Sari, Rashel Moritz, Mary Williamson, Vimal Manohar, Yossi Adi, Jay Mahadeokar, and Wei-Ning Hsu. Voicebox: Text-guided multilingual universal speech generation at scale. In *Advances in Neural Information Processing Systems*, volume 36, pp. 14005–14034. Curran Associates, Inc., 2023. doi: 10.52202/075280-0618. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/2d8911db9ecedf866015091b28946e15-Paper-Conference.pdf.
- Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. URL <https://arxiv.org/abs/2203.01941>.
- Dmitry Lepikhin, Hyoungho Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding, 2020. URL <https://arxiv.org/abs/2006.16668>.
- Shijia Liao, Yuxuan Wang, Tianyu Li, Yifan Cheng, Ruoyi Zhang, Rongzhi Zhou, and Yijin Xing. Fish-Speech: Leveraging large language models for advanced multilingual text-to-speech synthesis, 2024. URL <https://arxiv.org/abs/2411.01156>.
- Shijia Liao, Yuxuan Wang, Songting Liu, Yifan Cheng, Ruoyi Zhang, Tianyu Li, Shidong Li, Yisheng Zheng, Xingwei Liu, Qingzheng Wang, Zhizhuo Zhou, Jiahua Liu, Xin Chen, and Dawei Han. Fish Audio S2 technical report, 2026. URL <https://arxiv.org/abs/2603.08823>.
- Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for LLM training, 2025a. URL <https://arxiv.org/abs/2502.16982>.
- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025b.
- Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE reinforcement learning by aligning training and inference routers, 2025. URL <https://arxiv.org/abs/2510.11370>.
- Aleksandr Meister, Matvei Novikov, Nikolay Karpov, Evelina Bakhturina, Vitaly Lavrukhin, and Boris Ginsburg. LibriSpeech-PC: Benchmark for evaluation of punctuation and capitalization capabilities of end-to-end ASR models, 2023. URL <https://arxiv.org/abs/2310.02943>.
- Paulius Micikevicius, Dusan Stolic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. FP8 formats for deep learning, 2022. URL <https://arxiv.org/abs/2209.05433>.
- Eric Mitchell. A note on DPO with noisy preferences & relationship to IPO. <https://ericmitchell.ai/cdpo.pdf>, November 2023. Version 1.1.

-
- Tu Anh Nguyen, Wei-Ning Hsu, Antony D’Avirro, Bowen Shi, Itai Gat, Maryam Fazel-Zarani, Tal Re-
mez, Jade Copet, Gabriel Synnaeve, Michael Hassid, Felix Kreuk, Yossi Adi, and Emmanuel Dupoux.
EXPRESSO: A benchmark and analysis of discrete expressive speech resynthesis, 2023. URL <https://arxiv.org/abs/2308.05725>.
- NVIDIA Corporation. NVIDIA H200 GPU. Product page, accessed 2026-09-24, 2026a. URL <https://www.nvidia.com/en-us/data-center/h200/>.
- NVIDIA Corporation. NVIDIA Nsight Compute. Product page, accessed 2026-10-12, 2026b. URL <https://developer.nvidia.com/nsight-compute>.
- NVIDIA Corporation. NVIDIA Nsight Systems. Product page, accessed 2026-10-12, 2026c. URL <https://developer.nvidia.com/nsight-systems>.
- NVIDIA Corporation. Programmatic dependent launch and synchronization. CUDA Programming Guide, Section 4.5, 2026d. URL <https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/programmatic-dependent-launch.html>.
- NVIDIA Corporation. NVIDIA RTX PRO 6000 Blackwell. Product page, accessed 2026-10-13, 2026e. URL <https://www.nvidia.com/en-us/products/workstations/professional-desktop-gpus/rtx-pro-6000/>.
- NVIDIA Research. Kernel design agents (KDA): an agent-centric workflow to write high-performance CUDA kernels. GitHub repository, accessed 2026-10-13, 2026. URL <https://github.com/NVlabs/kda>.
- Open Compute Project. OCP microscaling formats (MX) specification, version 1.0, 2023. URL <https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf>.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. doi: 10.52202/068431-2011. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.
- Pratyush Patel, Esha Chouksey, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132, 2024. doi: 10.1109/ISCA59077.2024.00019. URL <https://arxiv.org/abs/2311.18677>.
- Puyuan Peng, Po-Yao Huang, Shang-Wen Li, Abdelrahman Mohamed, and David Harwath. VoiceCraft: Zero-shot speech editing and text-to-speech in the wild. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12442–12462, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.673. URL <https://aclanthology.org/2024.acl-long.673/>.
- Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL <https://arxiv.org/abs/2211.05102>.
- Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via FP16, 2025. URL <https://arxiv.org/abs/2510.26788>.
- Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free, 2025. URL <https://arxiv.org/abs/2505.06708>.
- Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL <https://qwen.ai/blog?id=qwen3.5>.

-
- Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pp. 28492–28518. PMLR, 2023.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.
- Marc’Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*, 2016. URL <https://arxiv.org/abs/1511.06732>.
- Chandan K A Reddy, Vishak Gopal, and Ross Cutler. DNSMOS P.835: A non-intrusive perceptual objective speech quality metric to evaluate noise suppressors, 2021. URL <https://arxiv.org/abs/2110.01763>.
- Yong Ren, Jiangyan Yi, Jianhua Tao, Tao Wang, Le Xu, and Zhengqi Wen. Edit content, preserve acoustics: Imperceptible text-based speech editing via self-consistency rewards, 2026. URL <https://arxiv.org/abs/2602.00560>. Interspeech 2026.
- Takaaki Saeki, Detai Xin, Wataru Nakata, Tomoki Koriyama, Shinnosuke Takamichi, and Hiroshi Saruwatari. UTMOS: UTokyo-SaruLab system for VoiceMOS challenge 2022, 2022. URL <https://arxiv.org/abs/2204.02152>. Interspeech 2022.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Noam Shazeer. GLU variants improve transformer, 2020. URL <https://arxiv.org/abs/2002.05202>.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarsz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL <https://arxiv.org/abs/1701.06538>.
- Feiyu Shen, Kun Xie, Yichen Wu, Ziqi Dai, Yichen Han, Junjie Li, Xuelong Geng, Fenglong Xie, Lei Xie, Xu Tang, and Yao Hu. FireRedTTS3: Unified speech generation and editing with semantically enriched speech representations, 2026. URL <https://arxiv.org/abs/2608.17492>.
- Xian Shi, Xiong Wang, Zhifang Guo, Yongqi Wang, Pei Zhang, Xinyu Zhang, Zishan Guo, Hongkun Hao, Yu Xi, Baosong Yang, Jin Xu, Jingren Zhou, and Junyang Lin. Qwen3-ASR technical report, 2026. URL <https://arxiv.org/abs/2601.21337>.
- Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don’t decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018. URL <https://arxiv.org/abs/1711.00489>.
- Soniox. Models: Soniox TTS RT v2. Model documentation, accessed 2026-10-09, 2026. URL <https://soniox.com/docs/tts/models>.
- Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding, 2021. URL <https://arxiv.org/abs/2104.09864>.
- Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision, 2015. URL <https://arxiv.org/abs/1512.00567>.
- Jinchuan Tian, Haoran Wang, Bo-Hao Su, Chien-yu Huang, Qingzheng Wang, Jiatong Shi, William Chen, Xun Gong, Siddhant Arora, Chin-Jou Li, Masao Someki, Takashi Maekaku, Keita Goto, Yusuke Shinohara, Jin Sakuma, Chao-Han Huck Yang, and Shinji Watanabe. Bagpiper: Solving open-ended audio tasks via rich captions, 2026. URL <https://arxiv.org/abs/2602.05220>.

-
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://arxiv.org/abs/1706.03762>.
- Chengyi Wang, Sanyuan Chen, Yu Wu, Ziqiang Zhang, Long Zhou, Shujie Liu, Zhuo Chen, Yanqing Liu, Huaming Wang, Jinyu Li, Lei He, Sheng Zhao, and Furu Wei. Neural codec language models are zero-shot text to speech synthesizers, 2023. URL <https://arxiv.org/abs/2301.02111>.
- Helin Wang, Meng Yu, Jiarui Hai, Chen Chen, Yuchen Hu, Rilun Chen, Najim Dehak, and Dong Yu. SSR-Speech: Towards stable, safe and robust zero-shot text-based speech editing and synthesis, 2024. URL <https://arxiv.org/abs/2409.07556>. ICASSP 2025.
- Juncheng Wang, Chao Xu, Cheng Yu, Zhe Hu, Haoyu Xie, Guoqi Yu, Lei Shang, and Shujun Wang. Language model based text-to-audio generation: Anti-causally aligned collaborative residual transformers, 2025a. URL <https://arxiv.org/abs/2510.04577>. EMNLP 2025.
- Yuhao Wang, Ziyang Cheng, Heyang Liu, Ronghua Wu, Qunshan Gu, Yanfeng Wang, and Yu Wang. VocalNet-M2: Advancing low-latency spoken language modeling via integrated multi-codebook tokenization and multi-token prediction, 2025b. URL <https://arxiv.org/abs/2511.10232>.
- Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270.
- Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785.
- Xiaomi MiMo Team. MiMo-V2.6-Pro-RL. <https://huggingface.co/XiaomiMiMo/MiMo-V2.6-Pro-RL>, 2026.
- Hanke Xie, Xiaming Ren, Dake Guo, Ruonan You, Wenhao Li, Jingbin Hu, Guobin Ma, Huakang Chen, Kejie Xu, Rui Huang, Weiguo Tan, Xianrong Wang, and Lei Xie. FlashTTS: Fast streaming TTS with MTP acceleration and X-pred mean flow distillation, 2026. URL <https://arxiv.org/abs/2606.09141>. Interspeech 2026.
- Canxiang Yan, Chunxiang Jin, Dawei Huang, Haibing Yu, Han Peng, Hui Zhan, Jie Gao, Jing Peng, Jingdong Chen, Jun Zhou, Kaimeng Ren, Ming Yang, Mingxue Yang, Qiang Xu, Qin Zhao, Ruijie Xiong, Shaoxiong Lin, Xuezhi Wang, Yi Yuan, Yifei Wu, Yongjie Lyu, Zhengyu He, Zhihao Qiu, Zhiqiang Fang, and Ziyuan Huang. Ming-UniAudio: Speech LLM for joint understanding, generation and editing with unified representation, 2025a. URL <https://arxiv.org/abs/2511.05516>.
- Chao Yan, Boyong Wu, Peng Yang, Pengfei Tan, Guoqiang Hu, Li Xie, Yuxin Zhang, Xiangyu (Tony) Zhang, Fei Tian, Xuerui Yang, Xiangyu Zhang, Daxin Jiang, Shuchang Zhou, and Gang Yu. Step-Audio-EditX technical report, 2025b. URL <https://arxiv.org/abs/2511.03601>.
- Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, pp. 115491–115522. Curran Associates, Inc., 2024. doi: 10.52202/079017-3668. URL <https://arxiv.org/abs/2406.06484>.
- Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, pp. 29687–29707, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/4904fad153f6434a7bcf04465d4be2cc-Paper-Conference.pdf.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale, 2025. URL <https://arxiv.org/abs/2503.14476>.

-
- Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30: 495–507, 2022. doi: 10.1109/TASLP.2021.3129994. URL <https://arxiv.org/abs/2107.03312>.
- Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL <https://arxiv.org/abs/1910.07467>.
- Bowen Zhang, Congchao Guo, Geng Yang, Hang Yu, Haozhe Zhang, Heidi Lei, Jialong Mai, Junjie Yan, Kaiyue Yang, Mingqi Yang, Peikai Huang, Ruiyang Jin, Sitan Jiang, Weihua Cheng, Yawei Li, Yichen Xiao, Yiyang Zhou, Yongmao Zhang, Yuan Lu, and Yucen He. MiniMax-Speech: Intrinsic zero-shot text-to-speech with a learnable speaker encoder, 2025. URL <https://arxiv.org/abs/2505.07916>.
- Yu Zhao, Yuanbin Qu, Konrad Staniszewski, Szymon Tworkowski, Wei Liu, Piotr Miłoś, Yuxiang Wu, and Pasquale Minervini. Analysing the impact of sequence composition on language model pre-training. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2024. URL <https://arxiv.org/abs/2402.13991>.
- Zhiyuan Zhao, Lijian Lin, Ye Zhu, Kai Xie, Yunfei Liu, and Yu Li. LEMAS: A 150K-hour large-scale extensible multilingual audio suite with generative speech models, 2026. URL <https://arxiv.org/abs/2601.04233>.
- Zhisheng Zheng, Puyuan Peng, Anuj Diwan, Cong Phuoc Huynh, Xiaohang Sun, Zhu Liu, Vimal Bhat, and David Harwath. VoiceCraft-X: Unifying multilingual, voice-cloning speech synthesis and speech editing, 2025. URL <https://arxiv.org/abs/2511.12347>. EMNLP 2025.
- Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024. URL <https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin>.
- Han Zhu, Lingxuan Ye, Wei Kang, Zengwei Yao, Liyong Guo, Fangjun Kuang, Zhifeng Han, Weiji Zhuang, Long Lin, and Daniel Povey. OmniVoice: Towards omnilingual zero-shot text-to-speech with diffusion language models, 2026. URL <https://arxiv.org/abs/2604.00688>.

A Pre-training Sequence Formats

Figures 7–10 show the TTS conversation of Stages 1 and 2 and the long-form, line-separated, and streaming sequences of Stage 3; text documents are plain text. `{{...}}` stands for a span of audio frames, and long lines are broken at spaces for display. `<|voice|>` is the modality token that opens an audio turn. The system instructions are the ones in use; all other text is illustrative. In every sequence, loss falls only on the audio frames of the assistant turns, on the separators of the line-separated form, and on the closing `<|im_end|>`. Speech infilling uses the layout of Figure 14; its loss covers either the whole middle or only its first and last few frames and the closing token, so the seams are always supervised.

```
<|im_start|>system
convert the provided text to speech reference to the following:

Text:
<|speaker:0|> I told you already, the train leaves at seven.

Speech:
{{reference clip}}<|im_end|>
<|im_start|>user
<|speaker:0|> [sighs] Fine. If we miss it, it's on you. [audio tokens: 60] [language: en]<|im_end|>
<|im_start|>assistant
<|voice|>{{utterance 1}}<|im_end|>
<|im_start|>user
<|speaker:0|> Seven sharp. I'll be at the door.<|im_end|>
<|im_start|>assistant
<|voice|>{{utterance 2}}<|im_end|>
```

Figure 7: **Stages 1–2: a multi-turn TTS conversation.**

The sequence of Figure 7 teaches plain synthesis and cloning. Consecutive utterances of one recording follow each other as user and assistant turns, with any reference clips in the system turn (Section 3, *Conversation format*), and the user text may carry point tags and language and duration hints.

```
<|im_start|>system
convert the provided text to speech reference to the following:

Text:
{{transcript of a clip from later in the recording}}

Speech:
{{that clip}}<|im_end|>
<|im_start|>user
<|speaker:0|> {{transcript of the recording up to the clip}}<|im_end|>
<|im_start|>assistant
<|voice|>{{the recording up to the clip}}<|im_end|>
```

Figure 8: **Stage 3: long-form synthesis.** The reference clip is cut from later in the same recording.

The sequence of Figure 8 voices a long recording from its transcript in one turn. Its reference clips are cut from later in the same recording and the target stops where the first clip begins, so no audio is both context and target. With several clips, the system turn either concatenates them as in S2 or gives each its own numbered block, at random. Some of these sequences have no reference; the system turn then holds only the instruction and the loss covers the whole recording.

The sequence of Figure 9 voices one text line by line. The separator `<|audio_separator|>` joins the lines in the user turn, and the model emits one audio segment per line with the separator between segments; its reference clips are chosen as in Figure 8, and some of these sequences carry none. The separators are supervised outputs, so the model learns where segments end and how many there are. The separators act as anchors between the text and the audio. If the model skips or repeats a sentence inside one segment, it still starts the next segment from the next line, so the mistake does not spread; and if the number of segments differs from the number of lines, the run-on is detected.

```

<|im_start|>system
convert the provided text to speech reference to the following:

Text:
{{transcript of a clip from later in the recording}}

Speech:
{{that clip}}

convert the provided text to speech; the lines are separated by <|audio_separator|>
– synthesize each line in order and output <|audio_separator|> between adjacent audio
segments<|im_end|>
<|im_start|>user
It was late.
<|audio_separator|>
The hall was empty.
<|audio_separator|>
Nobody answered.<|im_end|>
<|im_start|>assistant
<|voice|>{{line 1}}
<|audio_separator|>
{{line 2}}
<|audio_separator|>
{{line 3}}<|im_end|>

```

Figure 9: **Stage 3: line-separated synthesis.** The reference clip precedes the line instruction in the system turn.

```

<|im_start|>system
streaming speech continuation with one-chunk lookahead split by <|endofturn|><|im_end|>
<|im_start|>user
It was late<|endofturn|>and the hall was empty.<|im_end|>
<|im_start|>assistant
<|voice|>{{"It was late"}}<|im_end|>
<|im_start|>user
and the hall was empty.<|endofturn|>Nobody answered.<|im_end|>
<|im_start|>assistant
<|voice|>{{"and the hall was empty."}}<|im_end|>
<|im_start|>user
Nobody answered.<|endofturn|><|im_end|>
<|im_start|>assistant
<|voice|>{{"Nobody answered."}}<|im_end|>

```

Figure 10: **Stage 3: streaming interleave.**

The sequence of Figure 10 is for text that arrives in pieces. Each user turn holds the chunk to voice and, after `<|endofturn|>`, the chunk that follows, so the model knows what comes next while voicing the current chunk; an empty next chunk marks the last round. Chunks are consecutive spans of the transcript.

B Prompt Examples

Figures 11–14 render the main prompt formats in full. As in Appendix A, `{{...}}` stands for a span of audio frames. In the backbone stream, each frame occupies one position, entered as the fused embedding of its codes 0 and 1; codes 2–9 enter the decoder as side input (Section 2.3). The system instructions of the cloning, directing, and infilling prompts are shown exactly as used; all other text is illustrative, and long lines are broken at spaces for display. Loss applies only to assistant turns (Section 3, *Conversation format*), in a spoken turn to all ten codebooks of every supervised frame and to the closing `<|im_end|>`; the infilling format additionally weights its loss toward the seams (Appendix A). At inference, the prompt ends with the last `<|voice|>`, and the model generates frames until it emits `<|im_end|>`.

Figure 11 casts two voices from three reference clips. Speaker tokens are global to the request (Section 3, *Conversation format*). The identity in `<|speaker:S|>` is independent of the block number, so References 1 and

```

<|im_start|>system
convert the provided text to speech reference to the following:

[Reference 1]
Text: <|speaker:0|> I told you already, the train leaves at seven.
Speech: {{voice A, clip 1}}

[Reference 2]
Text: <|speaker:0|> Fine. But if we miss it, that's on you.
Speech: {{voice A, clip 2}}

[Reference 3]
Text: <|speaker:1|> Seven? Nobody said anything about seven.
Speech: {{voice B}}<|im_end|>
<|im_start|>user
<|speaker:0|> [whispering] Don't wake the baby.
<|audio_separator|>
<|speaker:1|> [laughing] Okay, okay, I'm going.<|im_end|>
<|im_start|>assistant
<|voice|>{{turn 1}}<|audio_separator|>{{turn 2}}<|im_end|>

```

Figure 11: Cloning a two-speaker dialogue from three reference clips, two of one voice.

2, two clips of one voice, carry the same speaker token, and Reference 3 introduces the second voice.

```

<|im_start|>system
convert the provided text to speech reference to the following:

[Reference 1]
Text: <|speaker:0|> Honestly, I thought we had a real shot this year.
Speech: {{reference clip}}<|im_end|>
<|im_start|>user
<|speaker:0|> [with heavy, deflated sadness] The results just came in. We didn't make it.<|im_end|>
<|im_start|>assistant
<|voice|>{{take 1}}<|im_end|>
<|im_start|>user
<|speaker:0|> [softer and warmer versus previous] The results just came in. We didn't make it.<|im_end|>
<|im_start|>assistant
<|voice|>{{take 2}}<|im_end|>
<|im_start|>user
<|speaker:0|> Direction: more amused than before; drop the hurt.
Line: The results just came in. We didn't make it.<|im_end|>
<|im_start|>assistant
<|voice|>{{take 3}}<|im_end|>

```

Figure 12: Multi-turn directing, in which each later turn gives a note on the previous take and the model produces a new one.

In Figure 12, turn 1 gives an absolute direction as a style prefix, turn 2 a relative note as a bracketed prefix, and turn 3 a relative note in the director’s-note register. At inference, the earlier takes are the audio that the model generated in previous turns, so each note is interpreted against what the listener actually heard.

In Figure 13, whose content is illustrative, the whole scene is one turn and every line is given up front. Characters are cast from their text descriptions rather than from reference audio, each line starts with its speaker’s name, and the separators are supervised outputs, so the model learns to emit exactly one audio segment per line, also when one character speaks consecutive lines (lines 1 and 2). The line-separated long-form format uses the same layout, with the lines of one text joined by separators in the user turn and one audio segment per line with supervised separators in the output. Only the scene and character descriptions are absent.

Figure 14 edits a recording by replacing one spoken word (“Wednesday” becomes “Thursday”). The prefix audio is the original recording before the edited word and the suffix audio is the recording after it, while the user turn carries the full new transcript, including the replacement word. Following the suffix-prefix-middle ordering of fill-in-the-middle training (Bavarian et al., 2022), the suffix comes before the prefix, so the prefix

```

<|im_start|>system
Scene: a small office on a Monday morning; the weekly meeting is about to start.
Characters:
- Mr. Carter: steady male voice, about 40, formal.
- Emma: young female voice, sleepy, half talking to herself.
Voice every line of the script in order, with <|audio_separator|> between consecutive lines.<|im_end|>
<|im_start|>user
Mr. Carter: [clears throat][serious] Good morning, everyone. Let's get started.
<|audio_separator|>
Mr. Carter: [reading from a sheet] First item: the launch date.
<|audio_separator|>
Emma: [softly][to herself] Here we go again...<|im_end|>
<|im_start|>assistant
<|voice|>{{line 1}}<|audio_separator|>{{line 2}}<|audio_separator|>{{line 3}}<|im_end|>

```

Figure 13: **Multi-character scene dubbing from text descriptions.**

```

<|im_start|>system
fill in the missing part of the speech; the suffix follows <|fim_suffix|> and the prefix follows
<|fim_prefix|><|im_end|>
<|im_start|>user
<|speaker:0|> The demo is Thursday, not Friday. [audio tokens: 14]
<|fim_suffix|>{{audio of ", not Friday."}}<|fim_prefix|>{{audio of "The demo is"}}<|im_end|>
<|im_start|>assistant
<|voice|>{{new audio of "Thursday"}}<|im_end|>

```

Figure 14: **Speech editing by infilling.**

sits directly next to the generated audio and the model continues from it; the optional duration hint gives the length of the new span in frames.

C Control Syntax Reference

Table 10 lists the controls that Drama 3 reads in its prompts. Structural tokens, written `<|name|>`, are single vocabulary entries; every other control is ordinary text. Section 3 names the training stage at which each format is introduced, and Appendix B shows complete prompts.

Conventions. A control is a short span in square or angle brackets, typically a few words, without nested brackets or line breaks; text in round parentheses is spoken. The direction controls (point tags, scoped tags, style prefixes, director’s notes, and relative notes) are free natural language, so any wording can be used and Table 10 only gives examples. A point tag acts at its position, a scoped tag on the enclosed span, and a style prefix, director’s note, or relative note on the whole line. Language and duration hints condition the output and are never spoken; the language hint is optional and names the main language of a request, and each line is spoken in the language of its own text.

Table 10: Drama 3 control syntax.

Control	Syntax	Examples
<i>Structure</i>		
Speaker token	<code>< speaker:N ></code> , $N = 0, \dots, 15$	<code>< speaker:0 ></code> Ready? <code>< speaker:1 ></code> Always.
Reference block	[Reference N] Text: <code>< speaker:S ></code> transcript Speech: <code>{{audio}}</code>	Figure 11
Line separator	<code>< audio_separator ></code>	The first line. <code>< audio_separator ></code> The second line.
Infill markers	<code>< fim_suffix >{{after}}</code> <code>< fim_prefix >{{before}}</code>	Figure 14
Streaming look-ahead	<code>chunk< endofturn >next chunk</code>	It was late <code>< endofturn ></code> and the hall was empty.
<i>Direction</i>		
Point tag	[tag] at a word position	Well [sighs] fine, you win. [laughs]
Scoped tag	<code><tag> ... </tag></code>	[pause] [gasps] [clears throat] [hesitates] Promise me <code><whisper></code> you won't tell her <code></whisper></code> . <code><slow>...</slow></code> , <code><emphasis>...</emphasis></code>
Style prefix	[style] line	[with bright eager excitement] We won!
Director's note	Direction: <i>note</i> Line: <i>text</i>	[icy calm, as if you already accepted it] Fine. Direction: icy calm, as if you already accepted it. Put the rise on "goodbye". Line: So this is goodbye.
Relative note	[change versus previous] line (second and later turns)	[softer and warmer versus previous] ... Direction: more amused than before; drop the hurt.
<i>Hints</i>		
Language hint	line [language: xx]	Welcome back to the show. [language: en]
Duration hint	line [audio tokens: N], N in frames	Welcome back to the show. [audio tokens: 65] (about 3s)