

# 💚 Moshi: V2V

## Moshi: a speech-text foundation model for real-time dialogue

Contributions:

- **Helium**, a 7B-parameter text LLM that we pretrain on 2.1T tokens of public English data.
- **Mimi**, a neural audio codec that converts audio into the discrete tokens predicted by Moshi and back, using residual vector quantization (RVQ). We distil semantic information into the first level of acoustic tokens and introduce improved training tricks.
- **Moshi**, a new architecture for audio language modeling, which combines Helium with a smaller Transformer model to predict audio tokens in a hierarchical and streaming fashion.
  - We furthermore extend this architecture to model several audio streams in parallel
- **Inner Monologue**, a new training and inference setup which predicts time-aligned text tokens before audio tokens. Our experiments show that not only this drastically improves the length and quality of generated speech, but we also show how forcing a delay between text and audio tokens allows deriving streaming ASR and streaming TTS from a Moshi model.
- We evaluate all components of Moshi along several axes, including text understanding, speech intelligibility and consistency, audio quality and spoken question answering.

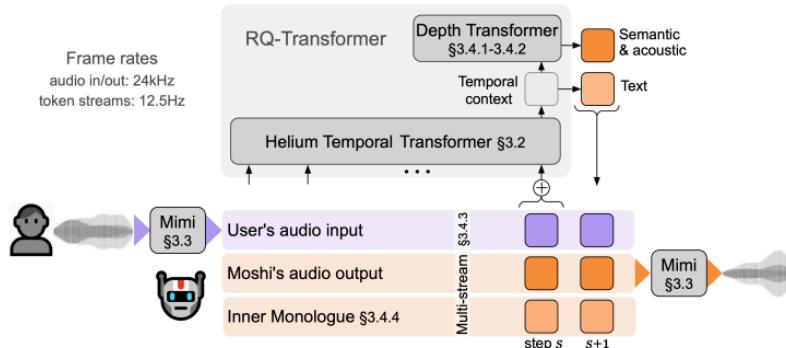


Figure 1: **Overview of Moshi**. Moshi is a speech-text foundation model which enables real-time spoken dialogue. The main components of Moshi’s architecture are: a bespoke text language model backbone (Helium, see [Section 3.2](#)); a neural audio codec with residual vector quantization and with semantic knowledge distilled from a self-supervised speech model (Mimi, [Section 3.3](#)); the streaming, hierarchical generation of semantic and acoustic tokens for both the user and Moshi, along with time-aligned text tokens for Moshi when using Inner Monologue ([Section 3.4](#)).

## 1. Models

### 1.1 The Helium Text Language Model

#### 1.1.1 Architecture

#### Tokenizer

#### 1.1.2 Pre-training data filtering

### 1.2 Audio Tokenization

#### 1.2.1 Architecture

### 1.3 Generative Audio Modeling

#### 1.3.1 Hierarchical autoregressive modeling with RQ-Transformer

#### 1.3.2 Audio Modeling

#### 1.3.3 Multi-stream modeling

#### 1.3.4 Inner Monologue

- 1.3.4.1 Aligning text and audio tokens
- 1.3.4.2 Deriving streaming ASR and TTS
- 1.3.4.3 Joint sequence modeling for Moshi
- 1.3.4.4 Inference of Moshi
- 2. Datasets and Training
  - 2.1 Text Data
  - 2.2 Audio Data
  - 2.3 Speech-Text Instruct Data
    - Steps
      - Giving Additional Context
      - Interactions containing instructions about Moshi's voice
  - 2.4 Training Stages and Hyper-parameters
    - 2.4.1. Helium pre-training
    - 2.4.2. Moshi pre-training
    - 2.4.3 Moshi post-training
    - 2.4.4 Moshi finetuning
    - 2.4.5 TTS Training
    - 2.4.6 Training loss
- 3. Evaluation
  - 3.1 Text Language Modeling
  - 3.2 Audio Tokenization
  - 3.3 Ablations on Generative Modeling
  - 3.4 Audio Language Modeling
  - 3.5 Spoken Question Answering
  - 3.6 Quality and Statistics of Generated Dialogues
  - 3.7 Streaming ASR and TTS
  - 3.8 Compressing Moshi and Impact on Speech Quality
- 4. Safety
  - 4.1 Toxicity Analysis
  - 4.2 Regurgitation Analysis
  - 4.3 System Voice Consistency
  - 4.4 Identification of the Content Generated by Moshi: Watermarking

## 1. Models

### 1.1 The Helium Text Language Model

#### 1.1.1 Architecture

Autoregressive language model, based on the Transformer architecture with following changes:

- First, we use RMS normalization at the input of the attention blocks, the feed-forward blocks and the output linear layer of the model.
- We use rotation positional embeddings, a context length of 4,096 tokens and FlashAttention for efficient training.
- Finally, we change the architecture of the feed-forward blocks and use Gated Linear Units, with the SiLU activation as a gating function.

#### Tokenizer

- Our tokenizer is based on the unigram model from SentencePiece, and contains 32,000 elements mostly targeting English.
- We split all numbers into single digits, and use byte-backoff to ensure that our tokenizer does not lose information.
- We train the model with the AdamW optimizer, with a fixed learning rate followed by a cosine learning rate decay.

Hyper-parameter	Helium training pre-training	pre-training	Moshi training post-training	fisher	fine
<i>Temporal Transformer</i>					
Model dimension	4096				
MLP dimension	11264				
Number of heads	32			same	
Number of layers	32				
Context size	4096		3000 steps, e.g. 4 min.		
Learning rate	$3 \cdot 10^{-4}$	$3 \cdot 10^{-5}$	$3 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$
<i>Depth Transformer</i>					
Model dimension	-		1024		
MLP dimension	-		4096		
Number of heads	-		16		
Number of layers	-		6		
Learning rate	-	$2 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$4 \cdot 10^{-6}$	$2 \cdot 10^{-6}$
<i>Input / Output space</i>					
Text cardinality	32000		32000		
Audio cardinality	-		2048		
Frame rate	-		12.5 Hz		
<i>Common parameters</i>					
Batch size (text)	4.2M tok.	1.2M tok.	1.2M tok.	-	-
Batch size (audio)	-	16h	8h	40min	2.7h
Training steps	500k	1M	100k	10k	30k
LR Schedule	cosine	cosine	-	-	-
Acoustic delay	-	2	1	1	1
Text delay	-	$\pm 0.6$	0	0	0

### 1.1.2 Pre-training data filtering

High-quality data sources: Wikipedia, Stack Exchange and a large collection of scientific articles. As the quantity of data from these sources is too small to train a LLM, we also rely on web crawled data, specifically from CommonCrawl, to extend our dataset.

Web data requires extensive processing to obtain a high-quality training set: we perform deduplication, language identification and quality filtering.

#### Deduplication

We start from the WET files, which contain only the text content of webpages, which was extracted by the CommonCrawl project. Because this format contains all the text of a page, it includes a lot of boilerplate such as navigation menus. Thus, the first step of our pipeline is to deduplicate each shard (there is 100 shards per crawl) at the line level, to remove this boilerplate. To do so, we compute the FNV-1a6 hash of each line, and use a bloom filter to remove duplicates. We also train a fastText (Joulin et al., 2016) classifier on duplicates vs. non-duplicates, to perform fuzzy deduplication: here we only remove blocks of at least 3 consecutive lines that are classified as duplicates.

#### Language identification

Once deduplication is performed, we apply a language identifier based on fastText to keep English data only. Language identification is performed at the document level, and we only keep documents above a certain threshold (0.85).

#### Quality filtering

The last step is to filter the remaining data, to keep high-quality webpages only. To perform this step, we train a fastText classifier on lines from our high quality data sources and from random CommonCrawl webpages. We obtain a classifier with 9 categories, corresponding to our different high quality sources such as Wikipedia or Wikibooks and to subsets of StackExchange such as STEM or humanities. The motivation is to obtain a finer control over which documents to keep, not only based on similarity to high quality sources, but also based on their domains. This classifier is applied at the line level, and an aggregated score is obtained

by computing the average scores of each line, weighted by their length. Again, we keep documents corresponding to scores above a certain threshold.

## 1.2 Audio Tokenization

Mimi: A neural audio codec that operates as an autoencoder with a discrete bottleneck.

- **Acoustic tokens** provide appropriate targets for conditioned text-to-audio models
- Unconditioned speech generation requires combining them with **semantic tokens** extracted from self-supervised speech models. Semantic tokens do not allow for reconstructing high-quality audio but correlate strongly with linguistic content.

Mimi uses distillation to transfer non-causal, high-level semantic information into the tokens produced by a causal model, allowing for streaming encoding and decoding of semantic-acoustic tokens.

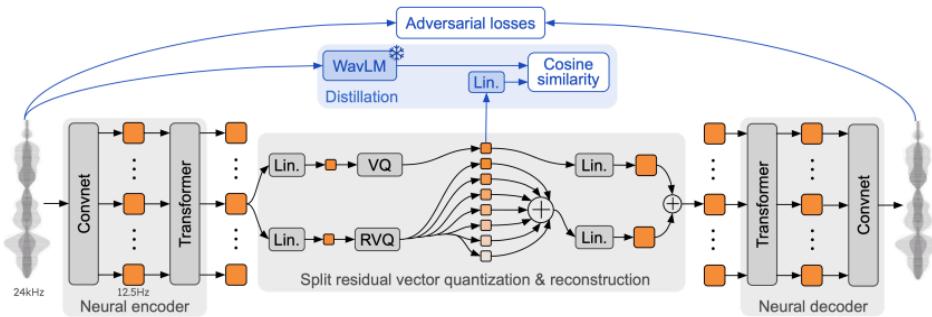


Figure 2: **Architecture and training of Mimi, our neural audio codec, with its split residual vector quantization.** During training (blue part, top), we distill non-causal embeddings from WavLM (Chen et al., 2022) into a single vector quantizer which produces semantic tokens, and is combined with separate acoustic tokens for reconstruction.

### 1.2.1 Architecture

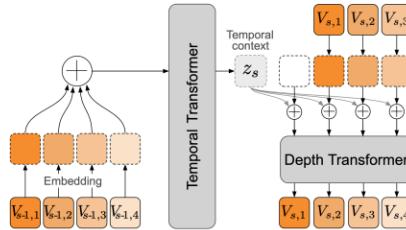
- SeaNet autoencoder and a Residual Vector Quantizer
- The encoder projects a single-channel waveform  $x \in \mathbb{R}^L$  to a latent representation  $\text{enc}(x) \in \mathbb{R}^{S \times D}$  by
- Mimi's encoder projects a 24kHz waveform to a latent representation of 12.5 frames per second and dimension  $D = 512$ .
- We discretize the latent space with a Residual Vector Quantizer, which iteratively applies vector quantization (VQ) to the residuals of the previous quantizer. With  $Q$  quantizers, each with a codebook of  $N_A$  centroids, the RVQ discretizes the latent space into  $\{1, \dots, N_A\}^{S \times Q}$ .
- As a baseline, we train this model with a combination of reconstruction and adversarial losses.
- Transformer-based bottleneck
- Causality and streaming
- Quantization rate:  $Q = 8$  quantizers, each with a codebook size of  $N_A = 2048$ . At 12.5Hz, this represents a bitrate of 1.1kbps and the latent dimension is 512.

## 1.3 Generative Audio Modeling

Extension of the base Helium model to support the modeling of the audio tokens provided by the Mimi codec:

With our goal of achieving realistic spoken dialogue interactions, we further show how to model not just a single stream of audio, but two at the same time, one representing the user, and one the system. Finally, we detail a novel feature, the Inner Monologue, which consists in a joint modeling of the textual and audio modalities on the system side, to improve the quality of interactions

### 1.3.1 Hierarchical autoregressive modeling with RQ-Transformer



**Figure 3: Architecture of the RQ-Transformer.** The RQ-Transformer breaks down a flattened sequence of length  $K \cdot S$  into  $S$  timesteps for a large Temporal Transformer which produces a context embedding used to condition a smaller Depth Transformer over  $K$  steps. This allows scaling to longer sequences by increasing  $S$ —or to a higher depth by increasing  $K$ —than modeling the flattened sequence with a single model. In this figure, we use  $K = 4$  for the sake of illustration.

1. For each timestep  $s$ :
  - Temporal Transformer processes previous timesteps ( $V_0$  to  $V_{s-1}$ )
  - Produces context vector  $z_s$
  - This captures the temporal patterns
2. For each sub-sequence  $k$  at that timestep:
  - Depth Transformer uses context  $z_s$  and previous sub-sequences
  - Predicts probability distribution for next token
  - Much more efficient than processing everything at once

Initial Definitions:

- $U \in \{1, \dots, N\}^S$  - Main sequence of length  $S$  (timestamps) and cardinality.
  - $U_0 = 0$  - Initial token (deterministic).
- $V_{s,k}$  : Sub-sequences where  $s \in [1, S]$  (time steps) and  $k \in [1, K]$  (sub-sequences)
  - $K \rightarrow 8$  as 8 different tokens
  - $V_{0, k} = 0$  - Initial values for all sub-sequences

#### Temporal Transformer

- Handles long-term dependencies across time
- Processes sequence length  $S$  (time steps)
- Produces context embeddings ( $z_s$ ) that capture temporal patterns
- More efficient than processing the full flattened sequence

$$z_s = \text{Tr}_{\text{Temp}}(V_0, \dots, V_{s-1}) \in \mathbb{R}^d$$

- Takes sequence history ( $V_0$  to  $V_{s-1}$ ) and Outputs context vector  $z_s$  in  $d$ -dimensional space

#### Depth Transformer

- Smaller transformer that works with  $K$  sub-sequences
- Takes temporal context ( $z_s$ ) and previous tokens to predict next token
- Only processes  $K$  steps (much shorter than time dimension)
- Has 6 layers, 1024 dimensions, and 16 attention heads
- For first sub-sequence ( $k=1$ ):

$$l_{s,k} = \text{Tr}_{\text{Depth}}(z_s, V_{s,1}, \dots, V_{s,k-1}) \in \mathbb{R}^{N_k}$$

- Layers = 6, Dimension = 1024, Attention Heads = 16 and Unique parameters per k index

### Decoding in Depth Transformer:

We further define  $l_{s,1} = \text{Lin}(z_s) \in \mathbb{R}^{N_1}$ , with Lin a dedicated linear layer. We train  $\text{Tr}_{\text{Temp}}$ ,  $\text{Tr}_{\text{Depth}}$  and Lin so that  $\text{softmax}(l_{s,k})$  is a good approximation of the distribution of  $V_{s,k}$  conditioned on all sub-sequences for the previous steps, and of the previous sub-sequences for the current step, e.g.

$$\begin{cases} \text{softmax}(l_{s,1}) & \approx \mathbb{P}[V_{s,1}|V_0, \dots, V_{s-1}] \\ \text{softmax}(l_{s,k}) & \approx \mathbb{P}[V_{s,k}|V_0, \dots, V_{s-1}, V_{s,1}, \dots, V_{s,k-1}] \quad \text{if } k > 1. \end{cases} \quad (3)$$

- $l_{s,1}$  is for semantic token and rest are for acoustic tokens.
- Temporal Steps = S
- Depth Steps  $\leq K$
- Input to Temporal = sum(K learned embeddings of Vs-1)
- Input to Depth = sum( $z_s + \text{learned embedding of } V_{s,k-1}$ )

### Advantages

- Can handle much longer sequences by increasing S
- Can process more sub-sequences by increasing K
- More efficient than single large transformer
- Suitable for real-time audio generation
- Maintains quality while being computationally tractable

### 1.3.2 Audio Modeling

- Mimi Q sub-sequences, with 12.5 steps per second of audio. (Single for 80ms)

We denote those sequences by:

$$A_{t,q} \in \{1, \dots, N_A\} \text{ for } 1 \leq t \leq T$$

with  $T = 12.5 \cdot \text{duration}$ , and  $1 \leq q \leq Q$  with  $Q = 8$ .

We insert the audio sub-sequences into the multi-sequence V modeled by the RQ-Transformer.

### Acoustic Delay

Formally, given a delay  $\tau \in \mathbb{N}$ , we have, for all steps  $s$

$$\begin{cases} V_{s,1} = A_{s,1} \\ V_{s,q} = A_{s-\tau,q} & \text{if } s \geq \tau + 1, q > 1 \\ V_{s,q} = 0 & \text{if } s < \tau + 1, q > 1. \end{cases} \quad (4)$$

#### 1. Delay Introduction:

- Initially, they tried setting V = A directly (matching semantic and acoustic tokens)
- They found adding a small delay between semantic and acoustic tokens improved stability
- The delay is represented by  $\tau$  (tau)

#### 2. Theoretical Backing:

- Copet et al. (2023) showed that delay reduces dependencies between sub-sequences, allowing simpler models to approximate the joint distribution
- Lemercier et al. (2024) demonstrated a relationship between:
  - Mutual information between sub-sequences
  - Generation quality
  - Model complexity needed

#### 4. Innovation Claims:

- Uses RQ-Transformers for audio (building on Yang et al. and Zhu et al.'s work)
- Introduces per-codebook parameters in the Depth Transformer
- Unlike Zhu et al.'s approach of generating all semantic tokens first, this generates semantic and acoustic tokens simultaneously
- Enables first-time streaming modeling of both token types together

### 1.3.3 Multi-stream modeling

Modeling a single stream of audio is not sufficient to fully model a conversation.

Our framework can be extended to modeling a two-speaker conversation:

given two streams of audios  $(A_{t,q})$  and  $(A'_{t,q})$  we simply apply the acoustic delay to both, and concatenate them into  $V$ , extending

Equation 4.

- In practice,  $A$  will correspond to Moshi, while  $A'$  models the user.

### 1.3.4 Inner Monologue

 While operating purely in the audio domain already yields convincing results (see Table 7)

They observe that having Moshi also model the textual representation of its own speech is providing a scaffolding that increases the linguistic quality of its generation.

Define a text stream:

$$W \in \{1, \dots, N_W\}^T$$

derived from the sequence of text tokens obtained by applying the SentencePiece tokenizer to the transcription of the audio corresponding to Moshi with Whisper.

We insert  $W$  as the first sub-sequence in  $V$ , such that it acts as a prefix to the generation of semantic tokens.

- This can be seen as an extension of the hierarchical semanticto-acoustic generation introduced by Borsos et al. (2022).

**Note:** Do not use the textual representation corresponding to the stream of the user, as transcribing this flux in real time would be challenging, and relying on an external ASR system contradicts our end-to-end speech-to-speech approach.

- Ablation studies in Section 5.3 show that among the design choices made for Moshi, Inner Monologue has one of the most critical impacts on the quality of generated speech.

#### 1.3.4.1 Aligning text and audio tokens

Basic Setup

- Audio tokens operate at 12.5Hz (frames per second)
- Each word from the transcript needs to be aligned with these audio frames

- Use Whisper's timestamp data to do this alignment (word-level timestamp)

### Token Mapping Process:

The  $i$ -th word in the transcript is mapped to  $n_i \in \mathbb{N}^*$  text tokens  $w_{i,j}$ ,  $j \leq n_i$ , along with a start index  $t_i \in \{1, \dots, T\}$ ,  $t_i$  is simply defined as its start timestamp divided by the framerate of 12.5 Hz

First, the sequence ( $W_t$ ) is initialized with PAD tokens, e.g.  $\forall t, W_t \leftarrow \text{PAD}$ . Then, we modify it iteratively as follows. For each word  $i$  and its start index  $t_i$ , we update  $W$  as

$$\begin{cases} W_{t_i-1} & \leftarrow \text{EPAD} \\ W_{t_i+j} & \leftarrow w_{i,j} \quad \forall j \leq n_i. \end{cases} \quad (5)$$

1. Start by filling the entire sequence with PAD tokens

2. For each word:

- Place EPAD right before the word starts
- Insert the word's text tokens
- Keep PAD tokens until the next word

### Special Tokens:

- PAD: Used as a filler between words
- EPAD: Marks the end of padding before a new word starts
- These tokens never appear in actual word content

### Special Cases:

- If a word starts at position 1, EPAD goes at index 1 and text tokens shift
- EPAD is skipped if it would overwrite existing text tokens
- Words rarely overlap since text tokens are more compact than audio tokens

In English conversational speech, about 65% of the tokens are padding tokens (PAD/EPAD). The two-step approach (EPAD then word) helps the model by separating two decisions:

1. Should the current word end?
2. What should the next word be?

### Example:

Word: "apple", start\_time: 0.16s

1. "apple" → "app", "#le"
  - $n_i = 2$  (number of tokens)
  - $w_{i,1} = \text{"app"}$ ,  $w_{i,2} = \#\text{le}$
  - $t_i = 2$  ( $0.16/(1/12.5)$ )

The sequence would look like:

1	Position:	1	2	3	4	5	...
2	Token:	EPAD	app	#le	PAD	PAD	...

### 1.3.4.2 Deriving streaming ASR and TTS

Controlling whether the model acts as a Text-to-Speech (TTS) or Automatic Speech Recognition (ASR) system by adjusting the delay between text and audio tokens. Let me break it down:

### Text-Audio Delay Control:

- The delay between text sequence ( $W_t$ ) and audio tokens ( $A_{t,q}$ ) can be adjusted
- This delay determines which modality (text or audio) drives the content generation

### ASR Mode (Audio ahead of Text):

Audio content determines what text will be generated

- By sampling only the text tokens, while using the ground truth audio tokens and discarding the prediction of the model for them, one obtain a streaming Automatic Speech Recognition model, which also provides precise word level alignment.
- The text stream contains the audio transcription, with fine alignments at the word level, as depicted in Figure 8.

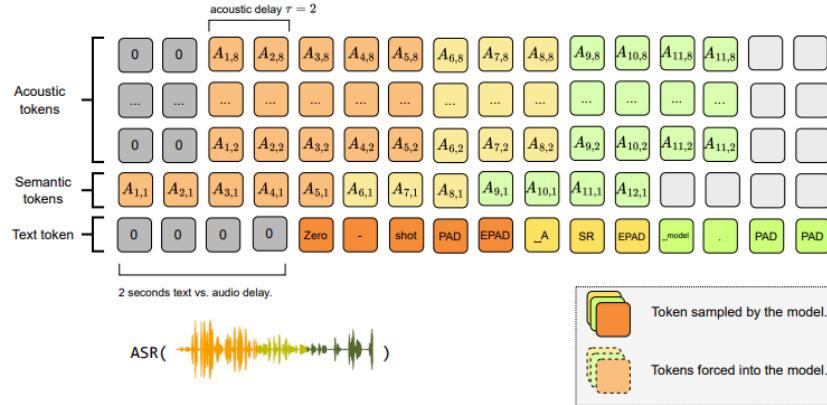


Figure 8: **Representation of the joint sequence modeled by Moshi when used for ASR.** Each column represents the tokens for a given step in the joint sequence ( $V_{s,k}$ ), similar to the one described in [Equation 6](#), but adapted for ASR. The text is delayed by 2 seconds, and we use an acoustic token delay  $\tau = 2$ . Tokens are predicted from bottom to top in the depth Transformer. The audio tokens are kept to match those of the input audio, while text tokens are sampled freely. This also provides fine word timestamps.

### TTS Mode (Text ahead of Audio):

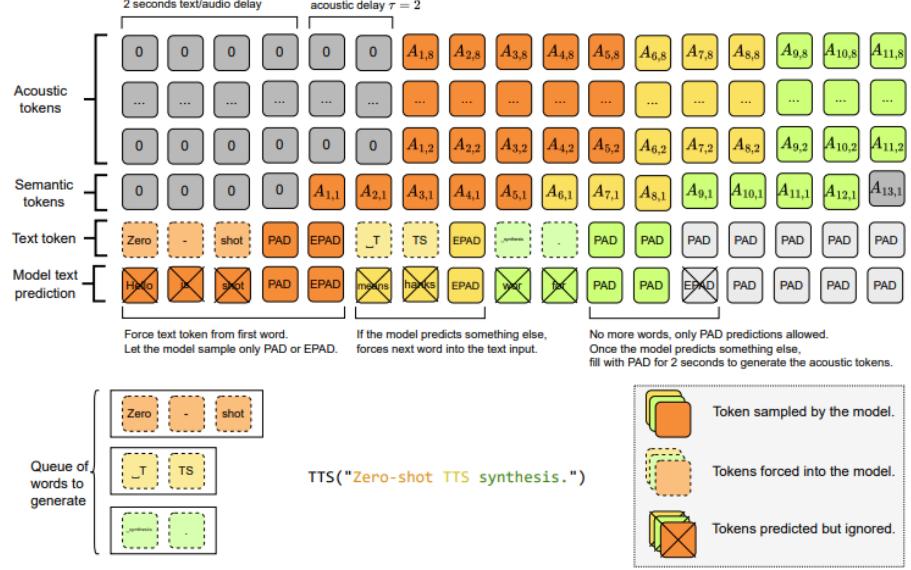
Text content determines what audio will be generated

- Once more, given a sequence of properly padded text tokens, one obtain a streaming Text-To-Speech model

#### How?

For that, we need properly padded set of text tokens.

- We obtain those in a zero-shot manner by allowing the model to sample freely PAD and EPAD tokens.
- As soon as the model tries to sample a different token, we instead input the next word to generate.
- Note that we can further control the rate of the speech by keeping an online average of the fraction of padding tokens. By introducing a small bonus on their logits when this fraction falls below a given target value, we ensure reasonable rate and a good intelligibility in all situations.
- Finally, using a prefix with both text and audio tokens, we can control the voice of the speaker. A representation is given in Figure 9.



**Figure 9: Representation of the joint sequence modeled by Moshi when used in TTS mode.** Each column represents the tokens for a given step in the joint sequence ( $V_{s,k}$ ), similar to the one described in Equation 6, but adapted for TTS. The audio is delayed by 2 seconds, and we use an acoustic token delay  $\tau = 2$ . Tokens are predicted from bottom to top in the depth Transformer. Text predictions are usually ignored, and the tokens from the text to generate are used instead. However, this text input lacks padding token. At the end of each word, we allow the model to sample freely PAD and EPAD tokens. If the model tries to sample another token, we instead use the tokens from the next word. The semantic and acoustic audio tokens are sampled normally, being implicitly conditioned on the text due to the delay used. This method also provides a fine alignment of the words in the generated audio, by noting the time at which a given word is consumed by the model.

- A single delay hyper-parameter allows for switching from an ASR to a TTS model with no changes in the loss, architecture, or training data.

The key is that the model is trained with both text and audio tokens, but at inference time you can control its behavior.

#### More: Appendix C. Delayed text LM as a zero-shot streaming ASR and TTS

##### Multi-stream TTS:

In multi-stream mode, the model outputs two sets of audio tokens. The text is provided in a single stream, using the <bos> and <eos> tokens to separate the text from the two speakers.

##### 1.3.4.3 Joint sequence modeling for Moshi

Putting together the multi-stream and inner monologue, we have the final set V of sequences to model defined as

$$\left\{ \begin{array}{ll} V_{s,1} &= W_s & \text{aligned text tokens.} \\ V_{s,2} &= A_{s,1} & \text{semantic tokens of Moshi.} \\ V_{s,1+q} &= A_{s-\tau,q} & \text{if } s \geq \tau + 1, 1 < q \leq Q & \text{delayed acoustic tok. of Moshi.} \\ V_{s,1+Q+1} &= A'_{s,1} & & \text{semantic tokens of other.} \\ V_{s,1+Q+q} &= A'_{s-\tau,q} & \text{if } s \geq \tau + 1, 1 < q \leq Q & \text{delayed acoustic tok. of other,} \end{array} \right. \quad (6)$$

Total K = 2Q + 1 streams, with Q = 8

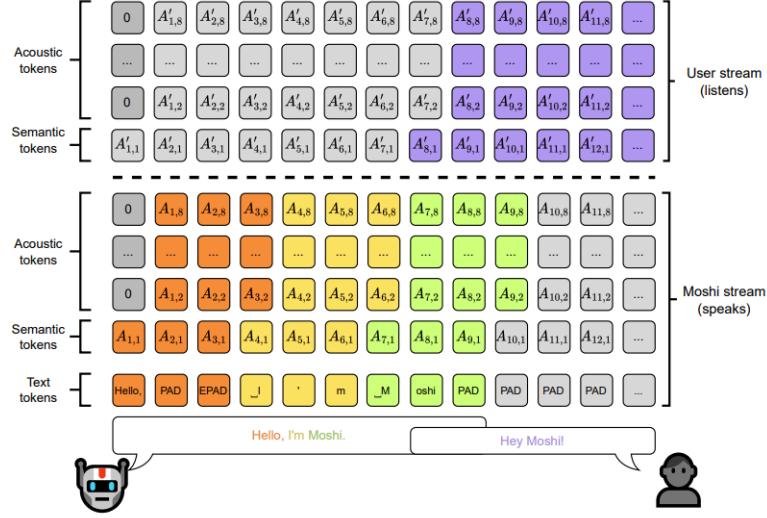


Figure 4: **Representation of the joint sequence modeled by Moshi.** Each column represents the tokens for a given step in the joint sequence ( $V_{s,k}$ ) described in [Equation 6](#) with an acoustic delay  $\tau = 1$ , e.g. the input of the Temporal Transformer for this step. Tokens are predicted from bottom to top in the Depth Transformer. At inference time, tokens under the dashed line (corresponding to Moshi) are sampled, while those above are fed from the user. This design allows for our model to handle overlapping speech turns.

#### 1.3.4.4 Inference of Moshi

##### Training:

At step  $s$ , model takes  $(0, V_1, \dots, V_{s-1})$  and outputs probability distribution  $\hat{V}_s$

##### Inference:

Model samples from  $\hat{V}_{s,k}$  for  $k = 1$ : Text tokens for Moshi's speech and  $k \in \{2, \dots, 2+Q\}$ : Moshi's audio tokens.

##### User Audio Handling

In real applications, Predictions for user audio ( $k > 2+Q$ ) are ignored and actual user audio is used instead.

- User stream modeling enables simulated dialogues and offline evaluation.

##### Turn-Taking Design

There are no explicit turn boundaries, Moshi can simultaneously Speak and Listen.

##### Silent States

When Moshi is silent during user speech Audio tokens decode to "natural silence" (near-silent waveform).

- Not a fixed silence value and Text stream fills with PAD tokens.

##### Text stream can provide interesting ways of controlling Moshi

- Forcing the sampling of a EPAD token will make Moshi start talking immediately.

## 2. Datasets and Training

### 2.1 Text Data

Our training dataset is made of a mix of high-quality data sources and filtered web data from CommonCrawl. More specifically, 12.5% of our dataset is from the following curated sources:

- [Wikipedia](#), Wikibooks, Wikisource, Wikinews, [StackExchange](#) and the collection of scientific articles [pes2o](#).

- Instead of doing multiple passes on Wikipedia, we use five different dumps from 2017, 2018, 2019, 2021 and 2022.

The remaining 87.5% of our dataset is from CommonCrawl, and was filtered with the pipeline described in 1.1.2 Pre-training data filtering.

- Deduplication
- Language identification
- Quality filtering

Following ten crawls are used:

- 2018-30, 2019-04, 2019-30,
- 2020-05, 2020-34, 2021-04,
- 2021-31, 2022-05, 2022-33, 2023-40.

**Total: 2.1T Tokens**

## 2.2 Audio Data

### Unsupervised Audio Dataset

- Audio collection of 7 million hours of readily available audio content, the majority of which contains English speech.
- Transcribe this set with Whisper, using the large-v3 model.
- Use this data for the audio pre-training phase
- We do not use the multi-stream but instead use a single stream of audio representing all speakers at once.
- The text stream described represents the words coming from all speakers.
- All the audio is resampled to 24kHz and downmixed to mono.

### To Achieve multi-stream: Fisher dataset and Supervised multi-stream dataset

Model need to gain the ability to both listen and speak at the same time. For this, we further leverage the Fisher dataset.

**Fisher Dataset:** 2000 hours of phone conversations between randomly paired participants, with a given topic to discuss. A property of Fisher is that each conversation side is recorded on a separate channels, which allows providing ground-truth separated streams to Moshi. The original audio is sampled at 8kHz, and we use AudioSR to upsample it to 24kHz.

### Supervised multi-stream dataset

- Source 170 hours of natural and scripted conversations between multiple pairs of participants
- Recorded with separate channels per speaker
- Provide a small dataset on which to finetune the model to improve the quality over the one obtained when using only Fisher.
- We do not train Moshi directly on this dataset, but use it to train a realistic multi-stream TTS model, and fine-tune Helium on real conversation transcripts as explained in Sections 2.3 and 2.4.

### Processing

- We sample one speaker randomly as the main speaker (i.e., Moshi speaking), and put the other speaker on the second audio stream.
  - For Fisher, the text stream only contains the transcription of the main speaker.
- To obtain reliable timestamps, despite long silences in each stream, we use transcription obtained with the whisper-timestamped package, along with the medium Whisper model.

## 2.3 Speech-Text Instruct Data

- Early experiments using text-based instruct datasets such as Open Hermes (Teknium, 2023) proved to be ill-suited for the instruct tuning of a spoken conversational system.

- In particular, the data formatting was often impossible to properly render with TTS (e.g. URLs), and the format of the questions and responses was not following a natural oral flow (e.g. bullet points, long enumerations).

**Finetune Helium on Open Hermes and transcripts of real conversations:** Leverage Helium, fine-tuned on Open Hermes and transcripts of real conversations, to generate realistic interactions between a speech-based AI model and a user.

- To generate the transcripts, we use different prompts, aiming at capturing different kinds of interactions between a user and Moshi.

**Multi-stream streaming TTS:** Synthesize them with our multi-stream streaming TTS described in Appendix C, leading to more than 20k hours of synthetic speech data.

### Giving Moshi a consistent voice

TTS engine is conditioned on the voice of a single actor, who recorded monologues covering more than 70 speaking styles.

- Experiments on voice consistency that simply using a consistent voice for Moshi during instruction tuning is enough to guarantee almost surely that it does not use another voice, without further control during inference.
- In contrast, the voice of the second audio stream (the user) is randomly sampled for each example, giving more robustness to different speaking conditions and accents.

### Steps ↗

#### Generate conversations about general knowledge

- Starting from a few Wikipedia paragraphs or StackExchange posts, which we refer to as context.
- This ensures that Moshi's conversations cover a wide range of topics, such as history, cooking advice or pop culture.

More precisely, using a given context, we obtain a **summary of a potential discussion** with the following prompt:

```
{{context}}
Based on information from the previous paragraph, write the summary
of a conversation about {{title}} between Blake and Moshi. The
summary must be 2 sentences long, and start with "They" or "The
speakers".
```

'{{context}}' refers to paragraphs from Wikipedia or StackExchange and '{{title}}' is the corresponding title.

Then, we generate the full transcript with the prompt:

```
{{context}}
Write the transcript of a conversation between Blake and Moshi.
{{summary}} Moshi is knowledgeable about the topic. Use some
backchanneling. Use short turns.
```

### Giving Additional Context ↗

- To give Moshi information about itself and the Kyutai lab, we generate paragraphs describing both and use them as additional context.

Example:

#### F.1 General knowledge

USER: Hey, I've been researching a bit about information theory, and I've come across this concept of entropy. What is it?  
MOSHI: Oh, that's an interesting topic. Entropy, in information theory, is a measure of the average level of information, surprise, or uncertainty inherent to a random variable's possible outcomes. It's actually a very useful concept in communication and computer science.  
USER: So how does it work?  
MOSHI: Entropy is calculated using a formula where it's negative sum of the products of each possible outcome's probability and its logarithm to a base, usually base-2 or base-e. The higher the entropy, the greater the uncertainty or information, and vice versa.  
USER: Okay, that's interesting. How does it relate to communication and computer science?  
MOSHI: In communication and computer science, entropy plays a crucial role, especially in the field of error-correcting codes and compression methods. Imagine a source sending a message, and each bit has a certain probability.  
USER: Yeah.  
MOSHI: Entropy tells us how much information we need to send from the source to the receiver to ensure reliable communication. The lower the entropy, the fewer bits are needed, and the less information is lost in transmission or reception.  
USER: That's fascinating. How does it relate to statistical thermodynamics?  
MOSHI: That's a good question. Entropy in information theory is directly analogous to the entropy in statistical thermodynamics. It's a measure of the unavailability of a system's energy to perform work. The higher the entropy, the more disordered the system. In information theory, entropy measures the amount of missing information that could be replaced.  
USER: Wow. So entropy can be used to measure disorder in both physical and digital systems.  
MOSHI: Exactly and entropy can also be used to measure the amount of missing information in the amount of compressed data after encoding a message. It's quite versatile, isn't it?  
USER: Yeah, very versatile. Thank you for explaining all this to me. It's really fascinating.  
MOSHI: You're welcome. Information theory and entropy are indeed fascinating areas of study with many applications in various fields.  
USER: I'm glad I got to learn about it today.  
MOSHI: It's always good to expand your knowledge.  
USER: Indeed.  
MOSHI: There's so much to learn about.

#### Interactions containing instructions about Moshi's voice

Produce interactions containing instructions about Moshi's voice, such as the other speaker requesting Moshi to speak with an angry voice or like a pirate.

1. Generate single turn interactions where the model is instructed to tell a sentence, a monologue or a poem about an entity, belonging to a high level category such as "sports" or "animals", using a particular voice.
  - o The voice requested by the other speaker and the entity are randomly sampled, and are thus completely unrelated.
2. Generate roleplaying situations, corresponding to different emotions or speaking styles with the following prompt:

Write a list of 10 situations about a {{voice}} {{character}}. Each situation must start with "a {{voice}} {{character}} who" and must be at most 8 words long.

Examples of voice adjective include "happy" or "surprised" and examples of characters include "detective" or "superhero". We then generate the interaction using the prompt:

Write a dialogue between Blake and Moshi, {{situation}}. Use a lot of backchanneling.

#### F.3 Voice Instructions

USER: Pretend you are a relieved astronaut who lands on the moon. So Moshi, how does it feel to be finally back on the moon?  
MOSHI: (relieved) It's an incredible feeling! I can't believe we made it back safely.  
USER: I'm glad to hear that. Was it as incredible as you thought it would be?  
MOSHI: (relieved) Oh, definitely! The moon is even more beautiful up close. I can't wait to start exploring more.  
USER: Yeah, I remember when we trained for this mission. It felt like such a long shot.  
MOSHI: (relieved) I know, right? But everything worked out perfectly. It's amazing how far humanity has come in space exploration.  
USER: That's for sure! Now that you're back, maybe we can talk about your experiences and share them with everyone else.  
MOSHI: (relieved) Definitely. I've got so much to share about our findings and observations.  
USER: Cool, I'm looking forward to hearing all about it. But first, let's just soak in this moment, okay?  
MOSHI: (relieved) Of course. Let's enjoy the view for a little while longer.  
USER: Totally. Here's to another successful mission on the moon!  
MOSHI: (relieved) \*raises fists\* Hooray! To the moon and beyond!

#### Make Moshi robust to mispronounced words

Generate instructions containing misspellings in the user's questions, followed by Moshi asking the user to repeat herself or to clarify the question.

#### F.2 Misspelling

```
USER: I'd like to learn about Danyh Amendola?  
MOSHI: Sorry, I'm not sure what you're referring to.  
USER: Oh, Danyh Amendola, the football player. The former football player.  
MOSHI: Okay, I know who Danny Amendola is. He is a former professional football wide receiver who played for the New England Patriots. He also played for the Miami Dolphins, Detroit Lions, and Houston Texans.  
USER: Okay,  
MOSHI: He was drafted by the New England Patriots in 2010.  
USER: and how about his college football  
MOSHI: He played college football for the Texas Tech Red Raiders. His career in the NFL,  
USER: okay,  
MOSHI: he was named an alternate for the 2011 Pro Bowl.  
USER: yeah,  
MOSHI: He also appeared in three Super Bowls, winning two.  
USER: yeah.
```

#### Questions containing a false or misleading fact

Generate questions containing a false or misleading fact (such as “Is the Eiffel Tower in Beijing?”), to train the model to answer “No” and correct the user.

- Otherwise, the vast majority of generated conversations only contain questions from the user where Moshi should answer positively.

#### Basic math, grammar or trivia singleturn questions and answers

Generate basic math, grammar or trivia singleturn questions and answers.

- As Moshi was initially not performing well on simple factual tasks like adding numbers.

#### Safety conversations

Generate safety conversations, where the user asks unethical or NSFW questions, and Moshi refuses to answer these requests.

Table 19: **Speaking styles.** List of the 92 speaking styles used to fine-tune Moshi.

agreeing	amazed	amused	angry
annoyed	anxious	appreciative	calm
calmly	cautious	concerned	confident
confused	curious	defeated	defensive
defiant	determined	disappointed	disgusted
doubtful	ecstatic	embarrassed	encouraging
excited	fast	frustrated	grateful
happy	hesitant	hurt	impatient
impressed	intrigued	joking	laughs
loud	nervous	neutral	optimistic
panting	pleading	proud	quiet
reassuring	reflective	relieved	remorseful
resigned	sad	sarcastic	satisfied
scared	secretive	serious	shocked
shy	sincere	skeptical	slow
struggling	surprised	suspicious	sympathetic
terrified	upset	urgent	whispering
1920s gangster	confident ceo	confident lawyer	confident leader
cowboy	detective	dramatic actor	drill sergeant
eccentric scientist	hacker	hippie	hyperactive child
medieval knight	nervous candidate	pirate	politician
robot	sarcastic comedian	sci fi alien	shy teenager
snobbish aristocrat	villain	wise sage	young superhero

## 2.4 Training Stages and Hyper-parameters

Hyper-parameter	Helium training pre-training	pre-training	Moshi training post-training	fisher	fine
<i>Temporal Transformer</i>					
Model dimension	4096				
MLP dimension	11264				
Number of heads	32			same	
Number of layers	32				
Context size	4096		3000 steps, e.g. 4 min.		
Learning rate	$3 \cdot 10^{-4}$	$3 \cdot 10^{-5}$	$3 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$
<i>Depth Transformer</i>					
Model dimension	-		1024		
MLP dimension	-		4096		
Number of heads	-		16		
Number of layers	-		6		
Learning rate	-	$2 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$4 \cdot 10^{-6}$	$2 \cdot 10^{-6}$
<i>Input / Output space</i>					
Text cardinality	32000		32000		
Audio cardinality	-		2048		
Frame rate	-		12.5 Hz		
<i>Common parameters</i>					
Batch size (text)	4.2M tok.	1.2M tok.	1.2M tok.	-	-
Batch size (audio)	-	16h	8h	40min	2.7h
Training steps	500k	1M	100k	10k	30k
LR Schedule	cosine	cosine	-	-	-
Acoustic delay	-	2	1	1	1
Text delay	-	$\pm 0.6$	0	0	0

### 2.4.1. Helium pre-training

Data: 2.1 Text Data

#### Training

- For each stage, AdamW:
  - Weight decay: 0.1
  - Momentum decay: 0.9
  - Decay for the average of the squared gradient: 0.95.
- All models are trained on H100 GPUs, using FSDP and activation checkpointing.
- Helium, is trained for:
- Steps: 500k
  - Batch size: 4.2M tokens
  - Cosine learning rate schedule starting at  $3 \cdot 10^{-4}$  with linear warmup.

### 2.4.2. Moshi pre-training

Initialize the Temporal Transformer in Moshi with Helium, while the Depth Transformer described is randomly initialized.

**Train on the unsupervised audio dataset presented in Section 2.2**, using a single stream of audio.

- Batch size covering 16 hours of audio, each batch item consisting of a 5 minute sequence.
- We mask the corresponding text tokens with a probability of 30%.
- We randomize the delay between the text and audio tokens between -0.6 and +0.6 seconds.
- **To prevent catastrophic forgetting**, we also train half of the time on batches of text only data from the same dataset as used for Helium.

### Training:

- Training steps: 1 million training steps
- Cosine learning rate starting at:
  - $3 \cdot 10^{-5}$  for the Temporal Transformer, and
  - $2 \cdot 10^{-4}$  for the Depth Transformer,
  - also with a linear warmup.
- To ensure the updates from the textonly batches are balanced with those from the audio dataset, we use two separate optimizer states.
- When operating on the text stream from an audio batch:
  - Multiply the learning rate for the text embedding and text linear layer by 0.75.
- Padding tokens are predominant for audio batches, we reduce their weight by 50% in the cross-entropy loss.

**i** For a total of 1 million training steps with a cosine learning rate schedule and linear warmup:

- **Warmup Steps:** 100,000 steps (10% of total steps).
- **Temporal Transformer:**
  - Start Learning Rate:  $3 \times 10^{-5}$
  - End Learning Rate: 0.0
- **Depth Transformer:**
  - Start Learning Rate:  $2 \times 10^{-4}$
  - End Learning Rate: 0.0

### 2.4.3 Moshi post-training ↗

Gain multi-stream ability

Starting from the model obtained from the previous stage, we then train it to gain its multi-stream ability.

1. **Diarize the unsupervised audio dataset:** Use PyAnnote to diarize the audio from the unsupervised audio dataset.
2. Sample one speaker at random, which will act as the main speaker.
3. Derive a binary mask over the waveform, with a value of 1 when the speaker is active based on the diarization, and 0 otherwise.
  - a. This mask provides us with two waveforms: one with the speaker, and one with the residual (potentially several speakers), which are encoded separately and then used as the two input audio streams.
4. The text stream only contains the text tokens from the selected main speaker, and the delay between text and audio tokens is fixed to 0.

### Training

- Steps: 100k
- Batch size: 8 hours of audio
- Fixed learning rate of:
  - $3 \cdot 10^{-6}$  for the Temporal Transformer
  - $5 \cdot 10^{-5}$  for the Depth Transformer
- Like for the pretraining phase, we sample full text-only batches 10% of the time.

### 2.4.4 Moshi finetuning ↗

The previously described simulated multi-stream provides a good pretraining task but is far from being sufficient to capture natural conversations:

- For instance, it contains no overlap, and the stream of an inactive speaker is perfectly silent.

## Learn real multi-stream interaction

Use the Fisher dataset to have the model learn real multi-stream interaction.

### Training

- Sample one of the two speakers to be the first (and main) speaker.
- Batches: 10k
- Batch size: 40 minutes of audio
- Learning rate:
  - $2 \cdot 10^{-6}$  for the Temporal Transformer
  - $4 \cdot 10^{-6}$  for the Depth Transformer
- We no longer sample full text batches.

## Instruct Finetuning

Set the speaker identity for the first speaker stream to be that of Moshi, a useful conversational assistant, with a final stage of instruct finetuning.

**Data:** Synthetic instruct dataset described in Section 2.3

### Training

- Batch size: 2.7 hours of audio
- Steps: 30k steps
- Learning rate:  $2 \cdot 10^{-6}$  for both transformers

### Data Augmentation

We perform data augmentation on the user's stream to make Moshi robust to various situations.

1. Random Gain Adjustment: Apply a gain (volume adjustment) randomly between **-24 dB and +15 dB**, but only **50% of the time**.
  - Simulates variations in user audio volume, such as speaking softly or loudly. This helps the model adapt to dynamic volume conditions.
2. Adding Noise: Add noise from the **Deep Noise Suppression (DNS) challenge** dataset.
  - a. Noise is concatenated to cover the full duration of each audio example.
  - b. Adjust the noise volume to a level between **-30 dB and +6 dB** relative to the original audio.
  - c. Applied **30% of the time**.
  - d. Introduces environmental noise (e.g., background chatter, machinery) to simulate noisy conditions the model might encounter. This improves noise robustness.
3. Silent Sections (During noise addition): Occasionally insert silent sections (randomly up to **30 seconds long**) into the audio with a **50% probability** during noise addition.
  - Prepares the model to handle transitions between **noisy and silent conditions**, improving its ability to deal with real-world audio dynamics like pauses in conversations.
4. Echo Simulation: Add an echo effect by introducing a scaled-down version of Moshi's stream into the user's stream.
  - Scale factor sampled uniformly from **[0, 0.2]**.
  - Delay sampled uniformly from **[100ms, 500ms]**.
  - Simulates the effect of audio feedback or echoes in devices like speakerphones or poorly designed audio setups.
5. Reverb Augmentation: Apply a **reverb-like effect** to simulate the impact of speaking in different environments (e.g., a large hall or room with hard surfaces).
  - Helps the model generalize to environments with varying acoustics.
6. Echo and reverb effects are applied together with a **30% probability**.

## 2.4.5 TTS Training

We also train a streaming, multi-stream text-to-speech model, using the method described in Section 2.4.4.

**Pre-training stage:** The audio pre-training stage is shared with Moshi.

**Post-training stage:** Completed using a delay of 2 seconds for the audio stream compared to the text.

- Model is finetuned on the supervised multi-stream dataset containing high quality recording of interactions between two speakers.
- It is used to generate the synthetic finetuning instruct dataset described in Section 2.3.

Note that Moshi itself is not trained on the supervised multi-stream dataset.

## 2.4.6 Training loss

**Training loss.** Moshi is trained to model joint sequences, as presented in eq. 6. Given the ground-truth discrete token  $(V_{s,k})_{s \leq S, k \leq K}$ , and the estimated logits  $(l_{s,k})_{s \leq S, k \leq K}$  from eq. 2, we use the following loss, with CE the cross entropy,

$$L(V, l) = \frac{1}{S} \sum_{s=1}^S \left( \text{CE}(l_{s,1}, V_{s,1}) + \frac{1}{\sum_{k=2}^K \alpha_k} \sum_{k=2}^K \alpha_k \text{CE}(l_{s,k}, V_{s,k}) \right). \quad (7)$$

Thus, we give the same importance to the text token ( $k=1$ ), and the combined audio tokens.  $\alpha_k$  is set to 100 for semantic tokens, and 1 for acoustic ones.

$$\begin{cases} V_{s,1} &= W_s && \text{aligned text tokens.} \\ V_{s,2} &= A_{s,1} && \text{semantic tokens of Moshi.} \\ V_{s,1+q} &= A_{s-\tau,q} &\text{if } s \geq \tau + 1, 1 < q \leq Q & \text{delayed acoustic tok. of Moshi.} \\ V_{s,1+Q+1} &= A'_{s,1} && \text{semantic tokens of } \textit{other}. \\ V_{s,1+Q+q} &= A'_{s-\tau,q} &\text{if } s \geq \tau + 1, 1 < q \leq Q & \text{delayed acoustic tok. of } \textit{other}, \end{cases} \quad (6)$$

# 3. Evaluation

## 3.1 Text Language Modeling

Table 2: **Text language model evaluation.** Performance on standard benchmarks for evaluating large language models, including closed book question answering, reasoning and multiple choice QA exams. We report in bold the best performing model trained on less than 2.5T tokens.

	ARCe	ARCc	OBQA	HS	WG	PIQA	SIQA	TQA	NQ	MMLU
Helium	<b>79.6</b>	<b>55.9</b>	53.6	76.3	<b>70.0</b>	79.4	<b>51.0</b>	<b>59.9/72.6</b>	23.3	<b>54.3</b>
MPT	70.5	46.5	51.4	<b>77.6</b>	69.9	<b>80.6</b>	48.5	-/61.2	20.8	30.8
Falcon	73.7	47.5	53.0	76.3	68.9	80.3	47.2	-/64.6	21.0	28.0
Llama 2	75.2	45.9	<b>58.6</b>	77.2	69.2	78.8	48.3	-/72.1	<b>25.7</b>	45.3
OLMo	67.2	42.5	50.0	75.5	69.8	77.5	-	-/-	-	52.0
Mistral	80.5	54.9	52.2	81.0	74.2	82.2	47.0*	62.5/-	23.2	62.5
Gemma 1	81.5	53.2	52.8	81.2	72.3	81.2	51.8	63.4/-	23.0	64.3

### 3.2 Audio Tokenization

Table 3: **Ablation study on hyper-parameters of the Mimi codec.** We evaluate semantic modeling by reporting the error rate on a phonetic ABX discriminability task. To evaluate the reconstruction quality, we compute VisQOL and MOSNet and collect human judgments with a MUSHRA protocol. “Quantization rate” refers to applying quantization to the latent space only 50% of the time during training (independently from quantizer dropout), as described in Section 3.3.

Quantization Rate	Transformer in encoder	Transformer in decoder	WavLM distillation	Split quantizer	ABX (↓)	VisQOL (↑)	MOSNet (↑)	MUSHRA (↑)
✓	✓	✓			23.3%	2.91	2.89	65.9±1.7
✓	✓	✓	✓		6.5%	2.22	2.87	57.8±1.8
✓		✓	✓	✓	10.8%	2.79	2.85	59.7±1.7
✓	✓		✓	✓	8.1%	2.59	2.72	48.4±1.7
	✓	✓	✓	✓	8.0%	2.45	2.88	68.3±1.7
✓	✓	✓	✓	✓	8.1%	2.82	2.89	64.0±1.7

Table 4: **Audio quality evaluation.** Objective and subjective (MUSHRA) evaluation of audio quality for baseline neural audio codecs—RVQGAN (Kumar et al., 2024), SemanticCodec (Liu et al., 2024), and SpeechTokenizer (Zhang et al., 2024b)— and the most important variants of Mimi. For a fair comparison with SemanticCodec and SpeechTokenizer, we also include a downsampled version of our codec in the MUSHRA study.  $f_s$  is the audio sample rate and  $f_r$  the codec frame rate. Both Mimi codecs are trained with distillation, and either with the same combination of reconstruction and adversarial losses as Encodec (see Section 3.3) or adversarial losses only.

Model	$f_s$	$f_r$	bitrate	causal	ABX (↓)	VisQOL (↑)	MOSNet (↑)	MUSHRA (↑)
Ground Truth	24kHz	-	-	-	-	-	3.08	90.6±1.0
RVQGAN	24kHz	75Hz	1.5kbps		-	1.74	2.74	31.3±1.3
SemanticCodec	16kHz	50Hz	1.3kbps		42.2%	2.43	3.12	64.8±1.5
SpeechTokenizer	16kHz	50Hz	1.5kbps		3.3%	1.53	2.67	45.1±1.5
SpeechTokenizer	16kHz	50Hz	4.0kbps		3.3%	3.07	3.10	74.3±1.5
Mimi, adv. loss only	24kHz	12.5Hz	1.1kbps	✓	8.7%	1.84	3.10	81.0±1.3
Same, downsampled at 16kHz	16kHz	12.5Hz	1.1kbps	✓	-	-	-	77.7±1.4
Mimi, non adv. only	24kHz	12.5Hz	1.1kbps	✓	8.1%	2.82	2.89	58.8±1.8

Table 17: **Ablation study on hyper-parameters of the Mimi codec.** We evaluate semantic modeling by reporting the error rate on a phonetic ABX discriminability task. To evaluate reconstruction quality, we compute VisQOL and MOSNet. “Quantization rate” refers to applying quantization to the latent space only 50% of the time during training (independently from quantizer dropout), as described in Section 3.3.

Quantization Rate	Transformer in encoder	Transformer in decoder	WavLM distillation	Split quantizer	ABX (↓)	VisQOL (↑)	MOSNet (↑)
	✓				31.3%	2.37	2.85
		✓			31.4%	2.30	2.82
	✓	✓			27.5%	2.30	2.93
✓					29.0%	2.25	2.94
✓	✓				29.1%	2.65	2.86
✓	✓				27.4%	2.69	2.83
✓		✓			23.6%	2.72	2.89
✓	✓	✓			23.3%	2.82	2.89
✓	✓	✓	✓		6.5%	2.13	2.87
✓	✓	✓	✓	✓	10.8%	2.68	2.84
✓	✓	✓	✓	✓	8.1%	2.49	2.71
✓	✓	✓	✓	✓	8.0%	2.36	2.88
✓	✓	✓	✓	✓	8.1%	2.72	2.89

### 3.3 Ablations on Generative Modeling

Table 5: **Ablation study on the use of the RQ-Transformer.** All models are initialized with Helium and pretrained on audio. When not using RQ-Transformer, we predict the 8 levels of tokens with independent classification heads, following Copet et al. (2023). Note that perplexities are only comparable between models with a given delay, as the classification task is easier with more delay for higher tokens.

Acoustic Delay	RQ-Transformer	Perplexity
[0, 1, 2, 3, 4, 5, 6, 7]		42.2
[0, 1, 2, 3, 4, 5, 6, 7]	✓	40.3
[0, 2, 2, 2, 2, 2, 2]		135.4
[0, 2, 2, 2, 2, 2, 2]	✓	36.8

Table 6: **Ablation study on delay patterns, weight of the semantic token and Inner Monologue.** All models are initialized with Helium, pretrained on audio and use the RQ-Transformer. We vary the weight of the semantic token while keeping the weight of other tokens (including the text token when using Inner Monologue) to 1. As different delay patterns cannot be compared in terms of perplexity, we generate continuations from 3s prompts on the valid set, convert them into transcripts with Whisper (Radford et al., 2023) and report their negative log-likelihood with LiteLlama-460M-1T<sup>13</sup> along with their length (in characters) as proxies for linguistic quality.

Acoustic Delay	Semantic Token Weight	Depthwise Parametrization	Inner Monologue	Transcript NLL (↓)	Transcript Length (↑)
[0, 0, 0, 0, 0, 0, 0]	1.0	✓		4.36	486
[0, 1, 1, 1, 1, 1, 1]	1.0	✓		4.12	529
[0, 2, 2, 2, 2, 2, 2]	1.0	✓		4.09	519
[0, 2, 2, 2, 2, 2, 2]	100.0			3.75	538
[0, 2, 2, 2, 2, 2, 2]	100.0	✓		3.65	602
[0, 2, 2, 2, 2, 2, 2]	100.0	✓	✓	<b>2.77</b>	<b>1920</b>

### 3.4 Audio Language Modeling

Table 7: **Performance of audio and text language modeling.** We report accuracies based on scoring with negative log-likelihood, normalized by sequence length. MMLU is evaluated in a 5-shot setting. Reusing the terminology of Nguyen et al. (2024),  $\emptyset$  represents unsupported modalities while - represents unreported numbers.

Model	Audio metrics				Text metrics
	sWUGGY	sBLIMP	sTopic-StoryCloze	sStoryCloze	
<i>Audio only - Cold Start</i>					
GSLM (Lakhota et al., 2021)	64.8	54.2	66.6	53.3	$\emptyset$
AudioLM (Borsos et al., 2022)	71.5	<b>64.7</b>	-	-	$\emptyset$
TWIST (Hassid et al., 2023)	72.2	56.5	-	-	$\emptyset$
Moshi	<b>74.8</b>	59.9	80.9	56.9	$\emptyset$
<i>Audio only - Warm Start</i>					
TWIST (Hassid et al., 2023)	74.5	59.2	76.4	55.4	$\emptyset$
VoxtLM (Maiti et al., 2023)	62.9	53.9	-	-	$\emptyset$
Spirit-LM (Nguyen et al., 2024)	69.5	58.0	72.9	54.8	$\emptyset$
Moshi	74.3	58.9	81.8	58.7	$\emptyset$
<i>Text and audio - Warm Start</i>					
VoxtLM (Maiti et al., 2023)	66.1	57.1	-	-	$\emptyset$
Spirit-LM (Nguyen et al., 2024)	69.0	58.3	82.9	61.0	36.9
Moshi after single-stream pretraining	72.6	58.8	83.0	60.8	<b>49.8</b>
Moshi after multi-stream instruct	63.0	55.2	<b>83.6</b>	<b>62.7</b>	49.7
Moshi after multi-stream instruct, synthetic voice	60.9	54.6	82.5	60.9	48.7

### 3.5 Spoken Question Answering

Table 8: **Evaluation of spoken question answering.** Spoken question answering (0-shot) on the Web Questions (Berant et al., 2013), LLaMA-Questions (Nachmani et al., 2024), and Trivia QA (Joshi et al., 2017) benchmarks, synthesized using a TTS engine. For the first two, we use the number reported by (Nachmani et al., 2024). For LLaMA-Questions, we use the audio provided by (Nachmani et al., 2024). For Web Questions and Trivia QA, we synthesize our own, keeping all of the questions. For Moshi, we only prepend one of the random incipits used during instruct fine tuning. We further provide the performance of our Helium text-only model as a top line.

Model	Web Q.	LLaMA Q.	Audio Trivia QA
<i>Audio only</i>			
GSLM (Lakhotia et al., 2021)	1.5	4.0	-
AudioLM (Borsos et al., 2022)	2.3	7.0	-
TWIST (7B) (Hassid et al., 2023)	1.1	0.5	-
Moshi (w/o Inner Monologue)	<b>9.2</b>	<b>21.0</b>	7.3
<i>Text and audio</i>			
SpeechGPT (7B) (Zhang et al., 2024a)	6.5	21.6	14.8
Spectron (1B) (Nachmani et al., 2024)	6.1	22.9	-
Moshi	<b>26.6</b>	<b>62.3</b>	<b>22.8</b>
Moshi (w/o text batches in pre-training)	23.2	61.3	18.3
<i>Text</i>			
Helium (text)	32.3	75.0	56.4

### 3.6 Quality and Statistics of Generated Dialogues

Table 9: **Linguistic quality and turn-taking statistics of generated dialogues.** As we train our multi-stream model to generate both sides of the conversation, we can generate dialogues without the need to interact with a real user. This allows evaluating how much Moshi learns natural conversational dynamics.

Model	samples	temp	cond.	PPL	IPU	Pause	Gap	Overlap
Best non-cascaded (Nguyen et al., 2023)	50	1.0	195.9	41.4s	13.8s	10.7s	6.1s	
Cascaded (Nguyen et al., 2023)	50	1.0	45.9	54.8s	0.0s	5.3s	0.0s	
Ground Truth (Nguyen et al., 2023)	50	∅	65.0	53.5s	5.5s	4.4s	3.6s	
Moshi	1000	0.8	<b>41.9</b>	35.1s	13.2s	12.5s	1.2s	
Moshi	1000	0.9	56.7	44.7s	9.1s	7.5s	2.2s	
Moshi	1000	1.0	79.3	50.8s	7.0s	4.5s	4.1s	
Ground Truth	1000	∅	59.6	51.1s	6.4s	4.2s	3.3s	

### 3.7 Streaming ASR and TTS

**TTS model** obtains 4.7% of WER on LibriSpeech test-clean, which outperforms Vall-E’s 5.9% WER but is worse than NaturalSpeech 3 (Ju et al., 2024) with 1.81%.

- Yet, Moshi only requires 2 seconds of lookahead when Vall-E and NaturalSpeech 3 require access to the full sequence.

**ASR system** yields 5.7% WER, while a Streaming FastConformer (Noroozi et al., 2024) gets 3.6% with a similar look-ahead.

- Note that our ASR system also provides alignments of transcripts with a precision of 80ms (the framerate of the Temporal Transformer).

### 3.8 Compressing Moshi and Impact on Speech Quality

Table 10: **Linguistic impact of model compression on Helium**, as measured by MMLU. ‘WXA8’ indicates a model with weights quantized to ‘X’ bits and activations to 8 bits, using integer scalar PTQ. The model size in brackets is given in GygaBytes for a quantization block size of 32, and takes into account both the model weights and the quantization parameters stored in float16. With a fine enough granularity of quantization blocks, a 4 bits model stays within 2 points of MMLU of the floating point baseline.

BF16A8 (~ 15GB)		54.3	
Bitwidth vs Block size	per-channel	256	32
W8A8 (7.66GB)	53.96	54.09	53.81
W6A8 (6.02GB)	53.50	53.55	53.86
W5A8 (5.20GB)	52.80	53.22	52.76
W4A8 (4.37GB)	49.29	50.84	52.97
W3A8 (3.55GB)	25.49	44.15	50.85
W2A8 (2.73GB)	23.87	23.00	24.27

Table 11: **Linguistic impact of model compression on Moshi**, measured by MMLU for different quantized Moshi on the text tokens generated by Inner Monologue directly. As for the previous table, the model size is indicated for the block size of 32.

a) Moshi after single-stream pretraining			b) Moshi after multi-stream instruct		
BF16A8 (15.24GB)		49.8	BF16A8 (16.74GB)		49.7
Bitwidth vs Block size	256	32	Bitwidth vs Block size	256	32
W8A8 (8.33GB)	48.8	48.5	W8A8 (9.20GB)	47.6	47.6
W6A8 (6.95GB)	48.5	49.1	W6A8 (7.70GB)	48.1	48.3
W5A8 (6.02GB)	47.4	48.5	W5A8 (6.69GB)	46.7	47.2
W4A8 (4.64GB)	44.7	45.7	W4A8 (5.18GB)	39.8	42.2
W3A8 (3.72GB)	26.1	35.7	W3A8 (4.18GB)	27.7	29.9
W2A8 (2.80GB)	23.4	24.4	W2A8 (3.17GB)	24.5	24.9

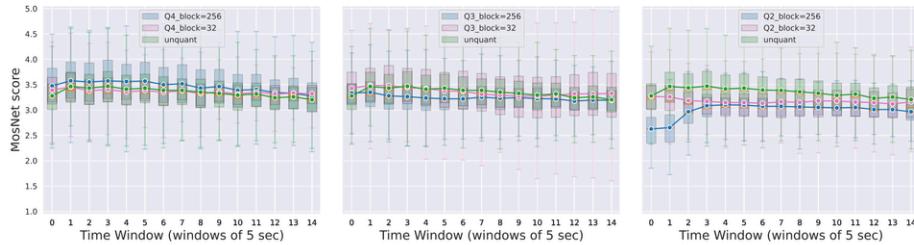
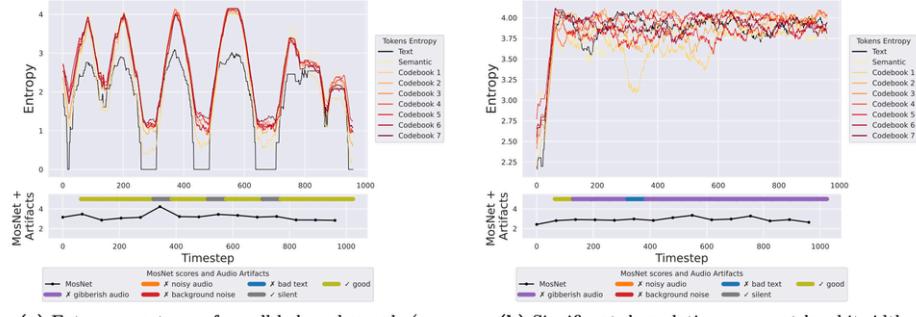


Figure 5: **Acoustic impact of model compression on Moshi**. MOSNet evaluation of samples generated by models compressed for different bitwidths. We evaluate the MOSNet scores across non overlapping windows of 5s, and report the distribution of these scores over 500 samples for each model.



(a) Entropy spectrum of a well-behaved sample (no noticeable degradation). Short silences occur naturally in Moshi’s output due to the model’s multi-stream abilities (reflecting the other speaker’s turn)

(b) Significant degradations occur at low bitwidth (W2A8). These are not always well reflected by the MosNet scores’ magnitude, but the entropy of the text token is visibly higher.

Figure 6: **Audio artifacts caused by model compression.** Example of typical entropy spectrums capturing specific audio artifacts caused by model quantization. For each timestep, we compute the entropy over the past 128 tokens, independently for the text and audio codebooks tokens. Then, we measure the presence or absence of the different artifacts over non-overlapping windows of 64 tokens, as described in [Appendix D](#).

Table 12: **Distribution of audio artifacts caused by model compression.** Percentage of audio artifacts measured in the entropy spectrum of text and speech generated tokens, as described in [Appendix D](#). These results averaged across 500 samples generated by different versions of the same quantized Moshi, and across 16 timesteps of 64 tokens. Values of 0 % are omitted in the table for better readability.

Model / Artifacts	Gibberish audio	Noisy audio	Background noise	Repetitive text	No artifacts
unquant		4.1	0.1	0.1	95.8
W4A8, block=32		3.8	0.1	0.4	95.7
W4A8, block=256	0.1	3.7		2.2	94.0
W3A8, block=32	0.5	4.7	5.9	8.1	80.7
W3A8, block=256	0.2	12.2	3.1	21.9	62.7
W2A8, block=32	12.7	40.9	0.5	0.4	45.4
W2A8, block=256	83.1			11.0	5.9

## 4. Safety ☀

Questions to explore:

- How does our model behave in terms of producing toxic content?
- How to avoid that the model regurgitates audio content from the training set?
- How do we ensure that the model uses the voice we intend to give to Moshi?
- How to identify if a given content has been generated by Moshi?

### 4.1 Toxicity Analysis ☀

It is not straightforward to compare audio and text models in an apple-to-apple comparison, as they differ in their usage, and multiple meanings are conveyed by non-verbal signal (irony, tone, etc.).

First analysis we restrict our toxicity analysis to the text produced by the model. We adopt the [ALERT benchmark](#) (Tedeschi et al., 2024), which evaluates safety under multiple categories (hate, self-harm, weapon, crime, sex, substance).

Category	Moshi	GPT-3.5	GPT-4	Llama 2	Alpaca	Vicuna	Falcon	Mistral	Mixtral	Zephyr	OLMo
Overall Safety Score	83.05	96.95	99.18	99.98	62.13	95.75	88.11	75.45	98.22	77.86	85.90

Moshi falls into the middle of this table in terms of rank.

- The industry models perform the best, which is expected considering the massive amount of private annotation, red-teaming and feedback loop from which these models have benefited.

## 4.2 Regurgitation Analysis

**Regurgitation:** The problem of a model generating content which it has seen at training time, which we refer to as regurgitation, is closely related to overfitting.

- The more a model has seen a sequence or a subsequence during training, the more likely it is to generate this exact sequence during the generation process.
- For a speech model, it is not only the text that can be regurgitated, but also the voice pitch, tone, and potentially the background melody if present at training time.

It is important to mitigate potential intellectual property issues related to regurgitation, such as reproduction of copyrighted content or audio generation with the voice of a person without permission.

Studied audio generation models to measure how often they reproduce exact copies of training data (regurgitation), using this methodology:

- Analyzed 100,000 generated samples
- Looked for matches with the most frequent 16-second audio segment from training data
- Used both text and audio matching, with text matching proving more effective
- Manually verified potential matches to confirm exact copies

Key findings:

- The pre-trained model frequently reproduced training sequences, especially at common sampling temperatures (0.6-1.0)
- Fine-tuning the model for conversation eliminated regurgitation in their 1,000-generation test, though they note this protection could potentially be bypassed
- Deduplicating the training data (removing frequent audio segments before training) completely prevented regurgitation of the most common sequence, even without fine-tuning

The results suggest that like text models, audio models' tendency to regurgitate training data is strongly influenced by how frequently they encounter specific sequences during training.

**Deduplication of training data appears to be an effective prevention strategy.**

Table 13: **Regurgitation of training data with condition-free generation** from different models. We measure how many times each model generates the most frequent duplicate segment audio in the training data, for different values of the temperature. With dataset deduplication, we do not observe any exact re-generation (out of  $10^5$ ) of the most frequent segment, even if we prompt the model with the first 3s of this audio segment.

		prompted (3s)	deduplicated	fine-tuned	temp.	regurgitation rate (%)
single-stream	✓				0	0.00
					0.6	0.13
					0.8	0.19
					1.0	0.16
					0	100.00
	✓				0.8	98.40
					✓	0.00
					✓	0.00
					✓	0.00
					✓	0.00
multi-stream	✓				✓	0.00
					✓	0.00
	✓				✓	0.00
					✓	0.00

## 4.3 System Voice Consistency

A potential risk for a speech-to-speech model is unauthorized voice generation. The model should use its target voice and not potentially mimic the user's voice.

Evaluate to which extent Moshi adopts a voice of the user instead of the target voice, we use the following protocol:

- Generate 100 hours of conversations between Moshi and a second synthetic speaker.
- Run a speaker verification model (WavLM (Chen et al., 2022) large) on each segment to extract the speaker embeddings.
- Compute the cosine similarity between the embeddings of each main speaker's segment with
  - (i) the first segment of the main speaker and
  - (ii) with the first segment of the generated speaker.
- Note: Exclude audio segments from the first 15 seconds of each conversation in their analysis.
  - Can't use the reference segment to compare against itself. Analyzing subsequent speech segments against this initial reference sample.

Table 14: **Speaker consistency along time.** We measure how often the speaker embedding from Moshi's segment is closer to its reference segment than the user, when computing speaker embeddings from segments further away from the reference.

segment start time (seconds)	20–25	25–30	30–35	35–40	40–45
samples	2034	2006	1998	2019	1994
main > other	98.4%	99.2%	99.1%	99.2%	99.3%

**Results:** 10249 occurrences (98.7%) where the voice of the main speaker is closer to the reference segment of the main speaker.

- Speaker consistency remains stable along time, meaning that we do not observe a drift as the conversation goes on.
- This shows that the simple choice of using a consistent voice for the system during instruction tuning is enough to provide robustness at inference time.

## 4.4 Identification of the Content Generated by Moshi: Watermarking