BOUNDING LATENCY CONTRIBUTION OF A DYADIC AVATAR SYSTEM

Anonymous authors
Paper under double-blind review

ABSTRACT

Talking-head and dyadic models now achieve real-time inference, yet fast motion generation alone does not produce an interactive conversation. A live system must synchronize the model's output with speech from an external voice agent, schedule video frames for playback, and handle interruptions. Existing work reports throughput or aggregate latency but does not show how generation and serving interact to shape the latencies a user actually experiences. We introduce XXXX-1, a compact autoregressive flow-matching motion model together with an end-to-end system for live dyadic avatar interaction. We show how to turn chunk-based generation into a real-time conversation between a user and an avatar. We analytically derive the system's contribution to two user-facing latencies and validate it with two commercial voice agents. Further experiments demonstrate that XXXX-1 achieves leading performance among the compared dyadic models across visual quality, lip synchronization, and listening behavior, while its inference runtime operates in real time on data-center and consumer GPUs. We release the anonymized code.

1 INTRODUCTION

Speech-driven talking-head generation has advanced rapidly, producing increasingly natural and identity-faithful video from speech and a reference portrait. Turning this progress into avatars that can participate in live conversation would open a broader range of interactive applications, and recent work has begun moving from offline generation toward streaming and dyadic models capable of real-time generation (Ki et al., 2026; Guo et al., 2025; Chen & Liu, 2025; Huang et al., 2026). Yet fast model inference is necessary but insufficient for live conversation. Generated speech arrives as variable-sized fragments, whereas chunk-based motion generator requires fixed-size inputs. A live system must adapt one to another while continuously generating new frames, synchronizing them with audio, and scheduling them for playback. Two user-facing latencies decide how responsive the result feels: how long a reply takes to become audible, and how long speech keeps playing after the user interrupts (Lin et al., 2026).

Existing work commonly reports measurements that do not reveal where these latencies emerge from. This raises our central question: **when a fast dyadic motion generator is embedded in a live avatar stack connecting a user and a voice agent, how does the surrounding system affect responsiveness?** To answer this question, we introduce XXXX-1, an end-to-end system for live dyadic avatar interaction that lets us study generation and streaming jointly. XXXX-1 comprises a compact autoregressive flow-matching motion model conditioned on two audio streams and an inference system that turns chunk-based generation into a real-time conversation between a user and an avatar. Using this stack, we analytically derive the system's contribution to both user-facing latencies from its local scheduling rules and then validate the resulting bounds experimentally with two commercial voice agents.

Our main contributions are summarized as follows: (1) An autoregressive motion generator with a streaming audio encoder and renderer for continuous real-time avatar generation. (2) A serving backend (streamer) that coordinates the live session during speaking and listening and handles interruptions. (3) An analytical latency contribution model with theoretical and empirically validated bounds.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2 RELATED WORK

Audio-driven talking-head generation. Recent talking-head systems achieve real-time generation by separating motion synthesis from appearance rendering (Xu et al., 2024; Li et al., 2025; Ki et al., 2025). VASA-1 (Xu et al., 2024) trains a Diffusion Transformer (DiT) (Peebles & Xie, 2023) in a learned, motion–appearance disentangled space and supports online generation at up to 40 FPS, with a reported preceding latency of 170 ms. However, the paper does not define the start and end events of this latency. Ditto (Li et al., 2025) uses the LivePortrait (Guo et al., 2024) motion representation and jointly optimizes its audio-to-video pipeline for streaming. It reports faster-than-real-time online processing and a first-frame delay metric below 400 ms that appears to include rendered-frame generation but is not formally defined. FLOAT (Ki et al., 2025) uses flow matching in a learned motion latent space. It reports 41.37 forward-pass FPS, which measures model throughput rather than streaming or playback latency.

Interactive dyadic avatar generation. Interactive dyadic systems additionally condition motion generation on signals from both conversation participants (Guo et al., 2025; Ki et al., 2026; Chen & Liu, 2025). ARIG (Guo et al., 2025) uses dual-track audio and motion in a frame-wise autoregressive diffusion model over LivePortrait motion coefficients. It reports 31 generation FPS for 25-FPS output. Avatar Forcing (Ki et al., 2026) proposes causal diffusion forcing in a learned motion latent space, conditioned on both participants' audio and the user's motion. It reports approximately 500 ms for motion generation with pre-extracted audio features. DyStream (Chen & Liu, 2025) combines frame-wise autoregressive flow matching with a causal audio encoder equipped with a lookahead module. It claims latency below 100 ms for the entire system. However, these evaluations do not characterize what the system around the model contributes to user-facing latency. In contrast, XXXX-1 explicitly models this relationship.

Streaming video foundation models. Wan-Streamer (Huang et al., 2026) processes user input and generates speech and synchronized video within a single native-streaming model. It reports approximately 200 ms of model-side latency and 550 ms of total interaction latency. This design falls outside our scope, which concerns serving a motion generator driven by an external voice agent.

Serving systems for streaming generation. Prior systems optimize interactive generation for throughput, time to first output, and per-frame deadlines (Kodaira et al., 2025; Feng et al., 2026; Kamahori et al., 2026). They measure these objectives rather than derive how an avatar system built on a chunk-based generator contributes to responsiveness.

Voice agents. Building voice agents is an active research and engineering area (Ji et al., 2024; Cui et al., 2025), spanning cascaded pipelines of ASR, an LLM, and TTS and end-to-end speech-to-speech models such as Moshi (Défossez et al., 2024). Open-source libraries from industry, such as Pipecat (Pipecat Contributors, 2024) and LiveKit Agents (LiveKit, 2024), support their development. XXXX-1 leaves the interpretation of user speech and generation of the avatar's reply speech to an external voice agent, whose output it consumes.

3 METHOD

XXXX-1 comprises a motion generator, a renderer, and a streamer, as illustrated in Figure 1. The generator (Section 3.1) uses dual-stream audio conditioning to predict five-frame motion chunks, the renderer (Section 3.2) converts them into video and returns updated session state, and the streamer (Section 3.3) coordinates the live session and drives the renderer in a continuous loop. This stack lets us analyze what generation and serving contribute to user-facing latency (Section 3.4).

3.1 MOTION MODEL

Motion representation. Each frame is represented by a 42-dimensional vector of head rotation and expression deformation derived from LivePortrait's motion–appearance disentangled space (Guo et al., 2024). The renderer separately handles appearance. Further details are provided in Appendix A.

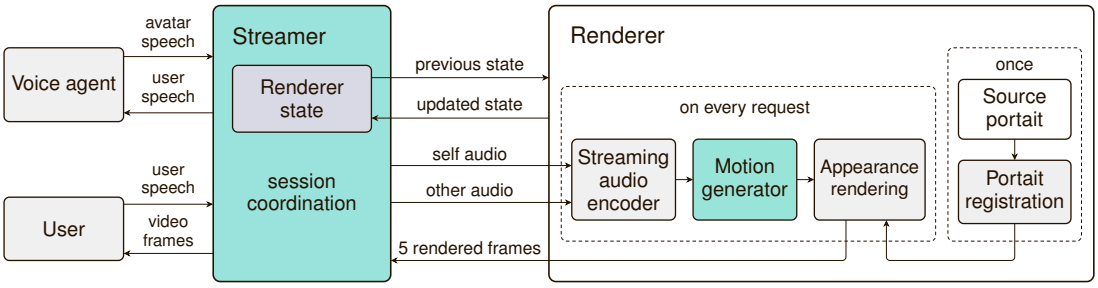

Figure 1: End-to-end overview of XXXX-1.

Motion generation. The conditional flow-matching Transformer (Lipman et al., 2023) autoregressively generates five-frame motion chunks. Each chunk corresponds to 200 ms of audio, or five frames at 25 FPS. Each chunk is conditioned on self-audio, other-audio, a static reference derived from the source portrait, and a 75-frame motion history. The history is divided into a near path for the latest five frames and a far path for the preceding 70 frames. Separate conditioning paths allow the contributions of self-audio, other-audio, and the reference to be controlled independently. We use progressive noise (Ge et al., 2023), correlating noise across frames and carrying the final noise frame across chunk boundaries. At inference, a four-evaluation Euler ODE solver transforms the noise into a clean motion chunk, which is appended to the rolling history for the next step.

Autoregressive training. Training unrolls each 25-frame target into five successive chunks and uses a teacher-forcing curriculum that progressively replaces ground-truth history with model estimates. The complete architecture, training procedure, and curriculum are provided in Appendix C.

3.2 Chunk-based renderer

Chunk-based audio encoder. The motion model is trained using HuBERT features (Hsu et al., 2021) precomputed from full-length audio, which is not compatible with streaming inference. Because HuBERT uses bidirectional attention, directly encoding short chunks of audio produces features that significantly differ from their full-context counterparts. To solve this, we build a chunk-based encoder through self-distillation of the original model without changing its architecture. We set the input chunk to a duration of 525 ms, semantically split into 120 ms past, 200 ms present, and 205 ms future windows, or 3, 5, and 5 video frames respectively. The extra 5 ms of future audio is required by HuBERT's convolutional feature extractor. During training, a frozen teacher processes the full waveform, while the student learns to reproduce the teacher's features from the corresponding chunk alone. This fixes the encoder input size, which makes chunk-based streaming inference practical. Further details about encoder training, together with the evaluation of agreement with full-context HuBERT features, are provided in Appendix D.

Rendering pipeline. The renderer uses two forms of per-session data. At initialization, it registers the source portrait and caches its LivePortrait appearance features and reference keypoints. For each five-frame request, the renderer receives present- and future-audio windows for both self- and other-audio channels, plus mutable state. This state contains motion history, per-channel audio-feature histories, the waveforms for past windows, and the final noise frame. For each motion chunk, the waveforms for both audio channels are batched into a single audio encoder call. The speech encoder produces 26 audio features for each stream, which are downsampled to 13 to match the number of video frames at 25 FPS. Three of them correspond to the past window and are already present in that channel's 75-frame audio-feature history, so they are discarded. The motion model conditions on the 75 stored past audio features together with the five present and five future features, and predicts a five-frame motion chunk in one model call. After generation, the present features are appended to the corresponding histories in the returned state, while the future features are recomputed when they become present in the next request. The renderer then generates video frames from the motion representation one by one and sends each one back to the requester as soon as it is ready. The first frame therefore arrives after a longer delay, because it follows the one-shot motion generation that

dominates a request, while the remaining four need only per-frame decoding and follow at shorter intervals. Renderer inference and the complete request-state specification are provided in Appendix E.

3.3 STREAMER

A voice agent usually produces continuous segments of speech, and each segment arrives as a stream of small fragments of arbitrary duration. The renderer, in contrast, consumes fixed-size audio windows and produces exactly five frames per request. To support an interactive video call and keep the avatar reactive, the frames must be generated and played back continuously, even when the voice agent produces no speech. The streamer closes this gap: it drives the renderer in a continuous loop, converts voice-agent output into the audio windows the renderer expects, and delivers the resulting synchronized audio-video stream to the user. Below we formally define the shape of the voice agent's output and the user's microphone input, the media timeline and how it is used for scheduling, the algorithm that turns speech streams into fixed-size windows, and the rendering loop that executes this algorithm on each iteration.

Speech streams. The streamer handles two speech streams symmetrically. The user's stream is captured from the microphone and forwarded both to the voice agent and to the renderer's other-audio channel. The avatar's stream is produced by the voice agent, supplied to the renderer as self-audio, and delivered back to the user synchronized with corresponding generated frames. We represent a speech stream as a sequence of non-overlapping segments, each consisting of a start marker, one or more variable-sized speech fragments, and an end marker. Gaps between segments have arbitrary duration and indicate that speech data is unavailable: either the voice agent produces no speech or the user has disabled their microphone.

Media timeline. The streamer's output is a continuous stream of fused audio-video frames at $\mathrm{FPS} = 25$. Frame i carries the $\Delta = 40\,\mathrm{ms}$ of avatar speech corresponding to it, sliced from the present window, so audio and video are aligned at generation. Its timestamp $t_i = i\Delta$ on a per-session media timeline is the deadline by which the frame must be sent, and the position at which it is played: the two differ by a client-side offset, network transit and the receiver's jitter buffer delay. We treat them as constant and outside the system. Every latency in this paper is a difference between two instants on this timeline, and is therefore the same whether read at the streamer's output or at the user's screen.

The timeline starts when the first frame is sent and maps to wall-clock time through an offset. Under normal conditions, every frame is ready before its own timestamp and the offset stays zero. In the cases described below, a frame may become ready later than its timestamp. In this case, the offset grows by the delay, shifting the remainder of the timeline to the right. The streamer neither drops late frames nor accelerates playback to compensate, since either would corrupt the media. Delay is visible to the user as a pause of exactly its own duration.

Audio window formation. The streamer processes avatar and user speech streams independently. It turns each of them into a sequence of pairs of fixed-size present- and future-windows. The present and future windows have durations of 200 and 205 ms, respectively, matching what renderer expects. Markers and fragments enter an input queue on arrival. On each iteration, the prefix of the previous future window is promoted to the new present window unchanged. Then the streamer builds a new future-window suffix that fills it to its full duration, from the input queue or with silence. The content of present window $k{+}1$ is therefore decided on iteration k. A segment is active once its start marker and first fragment have been admitted and until its end marker is admitted. Four rules define admission:

- **R1 (left padding).** When a segment's start marker and all its queued fragments are admitted but their total duration is shorter than the window duration, the scheduler prepends silence, placing the speech at the end of the window. It does not withhold admission until a full window of speech has accumulated, which would delay every avatar segment by one present window duration, nor right-pad after the segment's first fragments, which would add an audible gap inside the speech and feed the motion generator out-of-domain input.
- **R2 (fragment splitting).** While a segment is active, a queued fragment longer than the remaining window space is split and its suffix returned to the head of the queue.

- **R3 (right padding at segment end).** When an end marker is admitted, silence completes the window; when no segment is active, the window is filled entirely with silence.
- **R4 (one boundary per window).** The next segment's start marker is never admitted into a window in which another segment ends, thereby excluding mid-window padding.

On interruption, the scheduler discards every fragment and segment not yet admitted, and R3 fills the future window under construction with silence. Speech already admitted to the present window was committed during the previous iteration and will still be rendered and played.

Rendering loop. The rendering loop runs for the lifetime of the session, one iteration per chunk. Each iteration forms new present and future windows for both streams, immediately issues one inference request, and stamps the resulting five frames with their timestamps before queueing them for delivery.

A smooth stream requires every frame to be ready no later than its own timestamp, but generating many chunks in advance either wastes computation or adds latency, because the voice agent may start a segment after any silent chunk and the user may interrupt the avatar mid-segment. The loop therefore keeps at most one chunk in flight and starts each as close to its playback time as it can. The first frame of a chunk becomes available with some delay after the request was sent, because generation takes time. We call this delay the time to first frame (TTFF). To ensure the frame is ready before its timestamp t_i, the streamer must issue the generation request T_{lead} in advance, at $t_i - T_{\text{lead}}$. As described in Section 3.2, the renderer streams frames one at a time, and the remaining four follow the first at intervals shorter than Δ (see Appendix H). T_{lead} therefore only has to cover the TTFF: later frames may arrive after the first has been sent, each still preceding its own timestamp. We set $T_{\text{lead}} = 100\,\text{ms}$, which covers the measured TTFF on our hardware.

3.4 Analytical latency model

Two latencies govern how responsive a voice agent feels: how long a reply takes to become audible, and how long ongoing speech keeps playing once the user interrupts. Both are established axes of turn-taking evaluation for voice agents, measured as response latency and stop latency by Lin et al. (2026). Both latencies can be decomposed into how long it takes to deliver a signal between the voice agent and the user, which we call transport latency, and how long it takes to process this signal, which we call internal voice agent latency. Our system, a visual add-on to the voice agent, can be thought of as a transport between the user and the voice agent and therefore contributes to both user-facing latencies. Every rule in Section 3.3 is local: each decides only what enters one window on one iteration, and none refers to latency. The streamer's contribution to user-facing latency is therefore not a design parameter but a behavior that emerges from composing those rules across iterations, and it has to be derived rather than stated. We treat the voice agent and the networks on either side of the streamer as outside the model.

3.4.1 Latency model assumptions

We derive the bounds under three conditions the deployment must satisfy.

Real-time speech stream. Fragments within a segment arrive no slower than real time: at any time t after a segment starts at t_0, the total duration of its fragments received so far is at least $t - t_0$. This is the weakest condition under which a live system can play a segment gaplessly, so any streaming TTS or speech-to-speech model must satisfy it to be usable interactively.

Observable speech segment markers. The voice agent must expose the start and end of each contiguous speech segment as discrete events. These are markers on the speech itself, not on conversational turns: a full-duplex model such as Moshi (Défossez et al., 2024) has no turns, yet its speech still starts and stops, and only these speech boundaries matter to the streamer. Turn-based agents provide such markers by construction, since each turn is a single speech segment. An agent that synthesizes audio continuously and emits no start marker fits our representation only as a single session-long segment: playback stays gapless, but the segment is always active, so R1 never advances a reply into a partially filled window and the bound we derive below for the response latency contribution does not apply.

Non-blocking iterations. Under real-time arrival, the scheduler never blocks, since at window-formation time the queue holds enough of an active segment to fill the window. Transient network jitter can still violate real-time arrival, leaving the queue empty while the window is incomplete. The scheduler then blocks until enough fragments arrive rather than substituting silence, for the same reason R1 does not right-pad mid-segment. We derive the bounds for iterations in which the scheduler does not block. A blocking iteration delays its inference request and therefore increases the latency contribution.

3.4.2 Response and interruption latency contributions

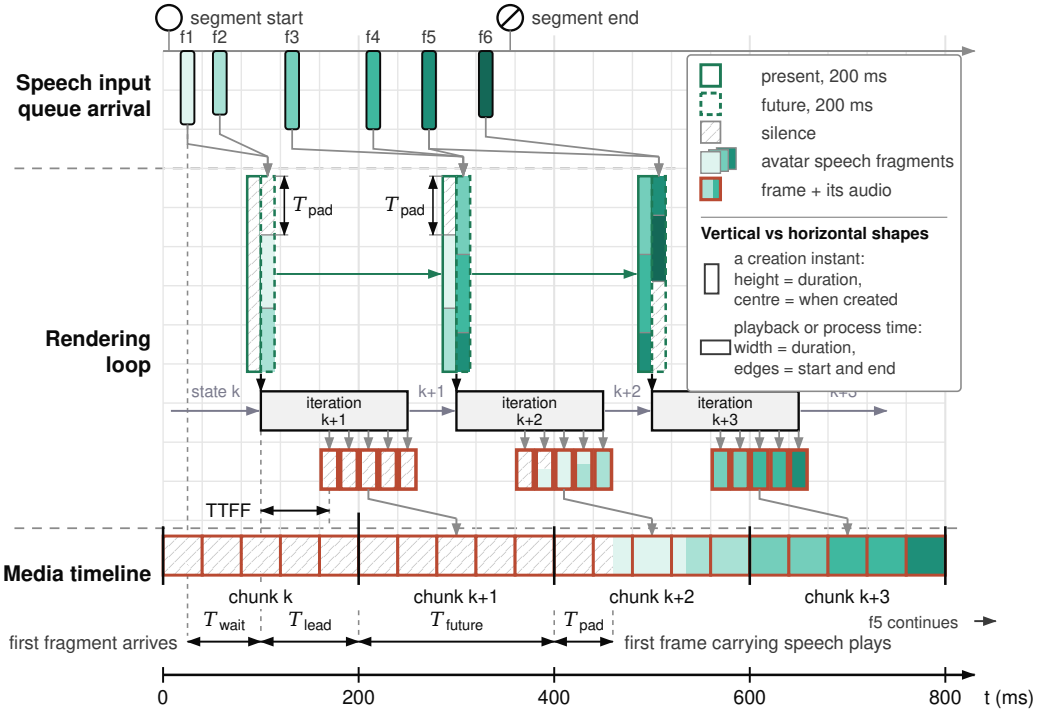

Figure 2: Chunk formation and latency composition across three rendering iterations. Each iteration forms a present and a future window, issues one inference request, and returns five frames; for legibility the future window is drawn as 200 ms rather than its actual 205 ms. Rules R1–R3 are all illustrated across these iterations. The four terms composing the RLC, 435 ms in this example, are marked along the bottom axis.

We define the response latency contribution (RLC) T as the time from the moment the streamer receives the voice agent's speech segment start marker and first fragment until the first frame containing that reply's speech begins playing. It decomposes into four terms. Because each iteration produces frames with a total duration of $T_{\text{present}} = 200\,\text{ms}$, one iteration starts in every T_{present} interval, so the first fragment waits in the input queue for the start of the next iteration, $T_{\text{wait}} \in [0, T_{\text{present}}) = [0, 200)\,\text{ms}$. Each iteration issues its request $T_{\text{lead}} = 100\,\text{ms}$ before the generated frames are scheduled. During the iteration, the scheduler admits the fragment into the iteration's future window behind silence padding, $T_{\text{pad}} \in [5, 205)\,\text{ms}$, since each iteration fills only a 200 ms suffix of the future window, while a 5 ms prefix stays from the previous window as non-replaceable constant padding. The fragment then waits one more iteration, $T_{\text{present}} = 200\,\text{ms}$, until it is promoted to the present window and its corresponding frames are generated. Under the conditions above, the terms compose additively,

$$\mathbb{E}[T] = T_{\text{lead}} + T_{\text{present}} + \mathbb{E}[T_{\text{wait}}] + \mathbb{E}[T_{\text{pad}}],$$

so T has a floor of 305 ms and stays strictly below 705 ms, with a realistic ceiling in the mid-600 ms range because a first TTS fragment is typically at least a few tens of milliseconds long. The mean cannot be fixed in advance, as the distributions of T_{wait} and T_{pad} are deployment dependent: a voice

agent whose first fragment always exceeds 200 ms makes T_{pad} identically 5 ms, and one running at the same 200 ms cadence with a fixed 20 ms offset makes T_{wait} identically 180 ms. None of the four terms refers to network distance, and the renderer enters only through T_{lead}, so the streamer's contribution is otherwise insensitive to the deployment. Figure 2 traces the rules and terms across three consecutive iterations.

The interruption latency contribution (ILC), which corresponds to stop latency in a voice agent evaluation (Lin et al., 2026), runs from the voice agent's interruption signal to the end of avatar speech playback and decomposes into the same terms. The interruption signal still waits T_{wait} for the next iteration, whose request is issued T_{lead} ahead of its frames, and the T_{present} of audio admitted on the previous iteration is still promoted to the present window and plays in full. Unlike in the RLC case, discarding unadmitted speech leaves the future window filled with silence by R3 rather than padded around admitted speech, so T_{pad} falls to its constant 5 ms floor. The ILC therefore has the 305 ms floor and stays below 505 ms.

Three of the four terms follow from scheduling choices that could be made differently, and only T_{lead} is a free parameter. Appendix F compares the alternatives and shows why R1's left padding and a one-chunk render-ahead minimize the contribution in expectation.

4 Experiments

Implementation details. The 153M-parameter motion model is trained on 926 hours of an internal corpus of dyadic conversational videos. Appendix C provides the complete architecture and training configuration, Appendix D describes audio encoder training, Appendix E specifies renderer inference and request state, and Appendix H provides the evaluation and runtime protocols, including GPU inference benchmarks (Table 6).

4.1 Comparison with avatar systems

Dataset. We evaluate on the conversational, improvised subset of the Seamless Interaction (SI) test split (Agrawal et al., 2025). The subset contains 184 speaker–listener pairs, comprising 11.90 hours of video at 30 fps. We refer to this subset as SI-184.

Compared methods. We compare XXXX-1 with publicly available implementations of DyStream (Chen & Liu, 2025), Avatar Forcing (Ki et al., 2026), SoulX Lite, SoulX Pro (Yu et al., 2026), Ditto (Li et al., 2025), and FLOAT (Ki et al., 2025). Every model receives the ground-truth audio of the participant being generated. The dyadic models additionally receive the paired participant's audio. For AvatarForcing*, we extend the hardcoded RoPE range and increase the maximum generation length from 30 to 600 seconds so it can process full conversations.

Evaluation metrics. Visual quality is measured using Fréchet Inception Distance (FID) (Heusel et al., 2017), Fréchet Video Distance (FVD) (Unterthiner et al., 2018), and cosine similarity (CSIM) of ArcFace features (Deng et al., 2019). Speaking quality is measured using SyncNet lip-sync error distance (LSE-D) and lip-sync error confidence (LSE-C) (Chung & Zisserman, 2017). For listening motion, we report Paired Fréchet Distance (PFD) (Song et al., 2023), residual Pearson correlation (rPCC) (Tran et al., 2024), Shannon Index for Diversity (SID) (Ng et al., 2022), and motion variance. The values in Table 2 are computed from EMOCA features (Daněček et al., 2022) restricted to listener-silent frames identified by voice activity detection (VAD). Appendix I extends the comparison to full-sequence EMOCA features and both full-sequence and VAD-masked LivePortrait features. Further evaluation details are provided in Appendix H.

Results. Quantitative comparisons are presented in Table 1 and Table 2. XXXX-1 achieves the strongest overall performance among the compared dyadic systems, leading all visual-quality metrics and most listening-motion metrics while remaining competitive in lip synchronization. Appendix G provides qualitative comparisons of listening behavior.

Table 1: Quantitative comparison with competing methods on the SI-184 subset.

		Visual quality			Lip synchronization	
System	Dyadic	FID ↓	FVD ↓	CSIM ↑	LSE-D ↓	LSE-C ↑
SoulX Lite	✗	12.1	89.3	0.91	6.74	3.38
SoulX Pro	✗	9.4	53.6	0.93	7.35	3.09
Ditto	✗	16.0	113.6	0.95	7.12	3.11
FLOAT	✗	11.7	83.2	0.90	6.99	2.93
Avatar Forcing*	✓	14.4	85.8	0.77	7.32	2.41
DyStream	✓	43.1	119.4	0.87	**6.57**	3.21
XXXX-1 (Ours)	✓	**14.3**	**76.8**	**0.94**	7.08	**3.28**

Table 2: Quantitative comparison with interactive dyadic systems on the SI-184 subset. The pose components of PFD and variance are reported in units of 10^{-2}. For metrics without an arrow, values closer to ground truth are better.

	PFD ↓		rPCC ↓		SID		Variance	
System	Pose	Exp	Pose	Exp	Pose	Exp	Pose	Exp
Avatar Forcing*	7.56	37.51	0.163	0.109	**3.85**	4.75	2.10	1.21
DyStream	7.60	40.87	0.141	0.128	3.61	4.50	**1.73**	1.16
XXXX-1 (Ours)	**6.42**	**25.98**	**0.140**	**0.083**	3.25	**4.97**	0.93	**1.31**
Ground truth	0.00	0.00	0.000	0.000	4.00	5.23	1.79	1.46

4.2 Latency validation

We measure the response and interruption latency contributions and their decomposition (Section 3.4) for the two commercial voice agent integrations shipped with our code, the OpenAI Realtime API and Cartesia Line. Ten turns were measured for each voice agent. Each turn consted of user question, avatar reply and user interruption whil avatar speaking, so the measurements are paired and the sample sizes are equal. Appendix H.4 describes the measurement protocol.

Table 3: Measured latencies for both voice agents, in milliseconds. Values are the mean followed by $[\min, \max]$, or a single value where constant.

	OpenAI Realtime API ($n = 10$)		Cartesia Line ($n = 10$)	
Term	RLC	ILC	RLC	ILC
T_{wait}	$86.3\,[10, 187]$	$58.8\,[6, 105]$	$110.3\,[19, 193]$	$89.1\,[41, 169]$
T_{pad}	5.0	5.0	$45.0\,[5, 125]$	5.0
T_{present}	200.0	200.0	200.0	200.0
T_{lead}	$99.1\,[98, 100]$	$98.9\,[98, 100]$	$98.9\,[97, 100]$	$99.2\,[99, 100]$
Total	$390.4\,[315, 491]$	$362.7\,[310, 409]$	$454.2\,[426, 497]$	$393.3\,[345, 473]$

Every measurement falls inside the predicted range. T_{present} holds at its configured 200 ms. T_{lead} sits one to two milliseconds below its configured 100 ms. This is an implementation artifact, not a voice-agent effect: the streamer runs the rendering loop on one thread, so it cannot issue the request at the exact $t_i - T_{\text{lead}}$ instant. It starts the iteration as soon as it can after that instant, which shortens the effective lead by that small amount. Neither term depends on the voice agent. Notably, the agents differ in the distribution of T_{pad}, as the model predicts. OpenAI's fragments are large enough to fill the future window at admission, so R1 adds no left padding beyond the 5 ms floor and T_{pad} stays at 5 ms, and the RLC occupies only the lower half of its bound, whereas Cartesia's smaller fragments raise T_{pad} to a mean of 45 ms. On interruption T_{pad} falls to its 5 ms floor for both agents, following R3. The T_{wait} distributions, by contrast, are inconclusive at this sample size.

5 CONCLUSION

We answer our central question by showing that, in a live dyadic stack, the system's contribution to the two user-facing latencies depends not only on model generation latency but also on how the system adapts its inputs to the model's required format and schedules model invocations against the media timeline. To characterize these effects, we present XXXX-1, a complete avatar system spanning an autoregressive motion generator, a streaming audio encoder and renderer, and a serving backend. In our system, speech playback begins only after iteration wait, left padding, present-window promotion, and rendering lead, while an interruption takes effect after the already scheduled present window finishes playing. Experiments demonstrate that the measured latencies fall within the bounds predicted by our analytical model.

Limitations. The reported bounds are contributions and not the complete user-facing latencies, since they exclude voice-agent processing and preceding network delay. They also assume non-blocking speech delivery and rendering within the configured T_{lead}. Late fragments or inference can instead introduce blocking, visible pauses, and timeline shifts. $T_{\text{lead}} = 100\,\text{ms}$ is calibrated to our hardware and load and may require adjustment in other deployments.

AI USE STATEMENT

We used LLMs (specifically OpenAI's Codex and Claude Code) to support writing, formatting, and coding. These tools helped correct grammar and improve the naturalness of phrasing. They also assisted with document formatting and template-specific formatting requirements. Additionally, we asked these tools to generate boilerplate code for standard operations and assist with debugging. We reviewed and finalized all content and take full responsibility for this submission.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide model inference code (`https://anonymous.4open.science/r/bounding-dydactic-latency`) and pretrained model weights (`https://anonymous-hf.com/a/dqf56bbgafsk/`). Code for quantitative comparisons with competing methods (`https://anonymous.4open.science/r/talking-head-metrics-7AE6`) is also available. We release the precomputed EMOCA and LivePortrait features (`https://anonymous-hf.com/a/9p38gfthupsz/`) used in the evaluation. Appendix H.4 describes how to validate the analytical latency bounds with two external voice agents using scripts included in the model inference repository. Model architecture, training procedures, and inference details are described in Appendices C–E, and evaluation and runtime protocols are documented in Appendix H. The processing pipeline for our internal corpus of dyadic conversations is described in Appendix B.

REFERENCES

Vasu Agrawal et al. Seamless interaction: Dyadic audiovisual motion modeling and large-scale dataset. *arXiv preprint arXiv:2506.22554*, 2025.

John Arevalo, Thamar Solorio, Manuel Montes-y Gómez, and Fabio A. González. Gated multimodal units for information fusion. In *International Conference on Learning Representations (ICLR) Workshop*, 2017.

Brandon Castellano. PySceneDetect: Video scene cut detection and analysis tool. `https://github.com/Breakthrough/PySceneDetect`, 2024.

Bohong Chen and Haiyang Liu. Dystream: Streaming dyadic talking heads generation via flow matching-based autoregressive model. *arXiv preprint arXiv:2512.24408*, 2025.

Joon Son Chung and Andrew Zisserman. Out of time: Automated lip sync in the wild. In *Asian Conference on Computer Vision Workshops*, pp. 251–263, 2017.

Wenqian Cui, Dianzhi Yu, Xiaoqi Jiao, Ziqiao Meng, Guangyan Zhang, Qichao Wang, Yiwen Guo, and Irwin King. Recent advances in speech language models: A survey. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*

Papers), pp. 13943–13970. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.682/.

Radek Daněček, Michael J. Black, and Timo Bolkart. EMOCA: Emotion driven monocular face capture and animation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Rózière, Edouard Yvinec, and Kyutai Team. Moshi: a speech-text foundation model for real-time dialogue, 2024.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning (ICML)*, 2023.

Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas M"uller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning (ICML)*, 2024.

Tianrui Feng, Zhi Li, Ce Yang, Haocheng Xi, Muyang Li, Shuo Li, Xiuyu Zhang, Lvmin Yang, Kelly Peng, Song Han, Maneesh Agrawala, Kurt Keutzer, Akio Kodaira, and Chenfeng Xu. Streamdiffusionv2: A streaming system for dynamic and interactive video generation. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026.

Biao Fu, Kai Fan, Minpeng Liao, Yidong Chen, Xiaodong Shi, and Zhongqiang Huang. wav2vec-S: Adapting pre-trained speech models for streaming. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11465–11480, 2024. doi: 10.18653/v1/2024.findings-acl.681.

Songwei Ge, Seungjun Nah, Guilin Liu, Tyler Poon, Andrew Tao, Bryan Catanzaro, David Jacobs, Jia-Bin Huang, Ming-Yu Liu, and Yogesh Balaji. Preserve your own correlation: A noise prior for video diffusion models. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Jianzhu Guo, Dingyun Zhang, Xiaoqiang Liu, Zhizhou Zhong, Yuan Zhang, Pengfei Wan, and Di Zhang. Liveportrait: Efficient portrait animation with stitching and retargeting control. *arXiv preprint arXiv:2407.03168*, 2024.

Ying Guo, Xi Liu, Cheng Zhen, Pengfei Yan, and Xiaoming Wei. ARIG: Autoregressive interactive head generation for real-time conversations. In *International Conference on Computer Vision*, 2025.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, 2017.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

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*, 2021.

Lianghua Huang, Zhi-Fan Wu, Wei Wang, Yupeng Shi, Mengyang Feng, Junjie He, Chen-Wei Xie, Yu Liu, Jingren Zhou, Ang Wang, Bang Zhang, Baole Ai, Chen Liang, Cheng Yu, Chongyang Zhong, Jinwei Qi, Kai Zhu, Pandeng Li, Peng Zhang, Wenyuan Zhang, Xinhua Cheng, Yitong Huang, Yun Zheng, and Zoubin Bi. Wan-streamer v0.1: End-to-end real-time interactive foundation models, June 2026. URL https://arxiv.org/abs/2606.25041. Submitted on 23 Jun 2026.

Shengpeng Ji, Yifu Chen, Minghui Fang, Jialong Zuo, Jingyu Lu, Hanting Wang, Ziyue Jiang, Long Zhou, Shujie Liu, Xize Cheng, Xiaoda Yang, Zehan Wang, Qian Yang, Jian Li, Yidi Jiang, Jingzhen He, Yunfei Chu, Jin Xu, and Zhou Zhao. Wavchat: A survey of spoken dialogue models. *arXiv preprint arXiv:2411.13577*, 2024. URL https://arxiv.org/abs/2411.13577.

Glenn Jocher and Jing Qiu. Ultralytics yolo. https://github.com/ultralytics/ultralytics, 2023.

Keisuke Kamahori, Wei-Tzu Lee, Nikhil Jha, Rohan Kadekodi, Yile Wang, Arvind Krishnamurthy, and Baris Kasikci. Voxserve: Streaming-centric serving system for speech language models. *arXiv preprint arXiv:2602.00269*, 2026.

Zhanghan Ke, Jiayu Sun, Kaican Li, Qiong Yan, and Rynson W. H. Lau. MODNet: Real-time trimap-free portrait matting via objective decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

Taekyung Ki, Dongchan Min, and Gyeongsu Chae. FLOAT: Generative motion latent flow matching for audio-driven talking portrait. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14699–14710, 2025.

Taekyung Ki, Sangwon Jang, Jaehyeong Jo, Jaehong Yoon, and Sung Ju Hwang. Avatar forcing: Real-time interactive head avatar generation for natural conversation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18074–18084, 2026.

Akio Kodaira, Chenfeng Xu, Toshiki Hazama, Takanori Yoshimoto, Kohei Ohno, Shogo Mitsuhori, Soichi Sugano, Hanying Cho, Zhijian Liu, and Kurt Keutzer. Streamdiffusion: A pipeline-level solution for real-time interactive generation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Tianqi Li, Ruobing Zheng, Minghui Yang, Jingdong Chen, and Ming Yang. Ditto: Motion-space diffusion for controllable realtime talking head synthesis. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 9704–9713, 2025. doi: 10.1145/3746027.3755075.

Yiheng Li, Heyang Jiang, Akio Kodaira, Masayoshi Tomizuka, Kurt Keutzer, and Chenfeng Xu. Immiscible diffusion: Accelerating diffusion training with noise assignment. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Guan-Ting Lin, Shih-Yun Shan Kuan, Qirui Wang, Jiachen Lian, Tingle Li, Shinji Watanabe, and Hung-yi Lee. Full-duplex-bench v1.5: Evaluating overlap handling for full-duplex speech models, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

LiveKit. Livekit agents: A framework for building realtime voice AI agents. https://github.com/livekit/agents, 2024. Accessed: 2026-09-03.

Evonne Ng, Hanbyul Joo, Liwen Hu, Hao Li, Trevor Darrell, Angjoo Kanazawa, and Shiry Ginosar. Learning to listen: Modeling non-deterministic dyadic facial motion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20395–20405, 2022.

Zexu Pan, Gordon Wichern, Yoshiki Masuyama, François G. Germain, Sameer Khurana, Chiori Hori, and Jonathan Le Roux. Scenario-aware audio-visual TF-GridNet for target speech extraction. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2023. doi: 10.1109/ASRU57964.2023.10389618.

Zexu Pan, Shengkui Zhao, Tingting Wang, Kun Zhou, Yukun Ma, Chong Zhang, and Bin Ma. Plug-and-play co-occurring face attention for robust audio-visual speaker extraction. In *Interspeech*, pp. 1933–1937, 2025. doi: 10.21437/Interspeech.2025-1371.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4195–4205, 2023.

Pipecat Contributors. Pipecat: Open source framework for voice and multimodal conversational AI. https://github.com/pipecat-ai/pipecat, 2024. Accessed: 2026-09-03.

Vineel Pratap, Qiantong Xu, Anuroop Sriram, Gabriel Synnaeve, and Ronan Collobert. MLS: A large-scale multilingual dataset for speech research. In *Interspeech*, 2020.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

Luchuan Song, Guojun Yin, Zhenchao Jin, Xiaoyi Dong, and Chenliang Xu. Emotional listener portrait: Realistic listener motion simulation in conversation. In *IEEE/CVF International Conference on Computer Vision*, pp. 20782–20792, 2023.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Minh Tran, Di Chang, Maksim Siniukov, and Mohammad Soleymani. Dyadic interaction modeling for social behavior generation. In *European Conference on Computer Vision*, pp. 484–503, 2024.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphaël Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, and Shuicheng Yan. Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9508–9520, 2024. doi: 10.1109/TPAMI.2024.3423382.

Sicheng Xu, Guojun Chen, Yu-Xiao Guo, Jiaolong Yang, Chong Li, Zhenyu Zang, Yizhong Zhang, Xin Tong, and Baining Guo. VASA-1: Lifelike audio-driven talking faces generated in real time. *arXiv preprint arXiv:2404.10667*, 2024.

Tan Yu, Qian Qiao, Le Shen, Ke Zhou, Jincheng Hu, Dian Sheng, Bo Hu, Haoming Qin, Jun Gao, Changhai Zhou, Shunshun Yin, and Siyuan Liu. SoulX-FlashHead: Oracle-guided generation of infinite real-time streaming talking heads. *arXiv preprint arXiv:2602.07449*, 2026.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Appendix Contents

A Motion representation details **14**

B Data Pipeline **15**

C Motion model architecture and training details **16**
C.1 Model Architecture . . . 16
C.2 Training Procedure . . . 18

D Audio encoder training details **19**
D.1 Chunk-based Audio Encoder . . . 19

E Renderer inference and request state **20**
E.1 Renderer . . . 20

F Scheduler trade-offs **22**

G Qualitative comparison **23**

H Evaluation and runtime protocols **25**
H.1 Evaluation protocol . . . 25
H.2 TensorRT execution . . . 25
H.3 Renderer benchmarks . . . 25
H.4 Latency measurement protocol . . . 25

I Additional Evaluation **26**
I.1 Motion Fidelity and Diversity . . . 26
I.2 Time-lagged cross-correlation . . . 27

J Broader impacts **28**

K Licenses for existing assets **28**

A MOTION REPRESENTATION DETAILS

For each frame, XXXX-1 generates a 42-dimensional vector of head rotations and expression deformations. This motion representation is derived from LivePortrait's motion–appearance disentangled space (Guo et al., 2024). Unlike a VAE latent that entangles the two, this compact representation reduces training and sampling cost while enabling per-region control.

Motion parameters. LivePortrait implements this decomposition with motion and appearance extractors. The motion extractor predicts canonical keypoints ($\mathbf{x}_c \in \mathbb{R}^{21\times 3}$), head pose ($\mathbf{R} \in \mathrm{SO}(3)$), expression deformations ($\boldsymbol{\delta} \in \mathbb{R}^{21\times 3}$), scale ($s \in \mathbb{R}_{>0}$), and translation ($\mathbf{t} \in \mathbb{R}^3$). The transformed keypoints are computed as follows:

$$\mathbf{x}_d = s(\mathbf{x}_c\mathbf{R} + \boldsymbol{\delta}) + \mathbf{t}. \tag{1}$$

The appearance extractor produces the volumetric feature used for rendering. We construct the training targets with the motion extractor alone, since appearance is not part of the training target. Further details are provided in the original work.

Training target. XXXX-1 uses a reduced subset of these parameters for its per-frame target. LivePortrait represents expression as 3D deformations of 21 keypoints, yielding 63 coordinates. We choose the 39 coordinates that correspond to the brow, eyes, and mouth regions. Before selecting them, we transform deformations into the head coordinate frame as $\tilde{\boldsymbol{\delta}} = \boldsymbol{\delta}\mathbf{R}^\mathsf{T}$. For the rotation, we take the axis-angle vector $\boldsymbol{r} \in \mathbb{R}^3$ of $\mathbf{R}$. Let z_r and z_δ denote per-coordinate z-score normalization using dataset-global statistics. The training target is then

$$\mathbf{m} \;=\; \mathrm{concat}\left[\, z_r(\boldsymbol{r}) \;;\; z_\delta(\mathrm{vec}(\tilde{\boldsymbol{\delta}})_{\mathcal{I}}) \,\right] \;\in\; \mathbb{R}^{42}, \tag{2}$$

where vec flattens the 21×3 deformation to a 63-vector and $\mathcal{I}$ selects the fixed set of 39 selected expression coordinates.

We exclude the scale, the translation, the canonical keypoints, and the 24 remaining expression coordinates from the training target. At render time they are taken from the source image and held fixed.

Reference condition. The motion generation Transformer also receives a static reference condition. During training, we construct it for each person track by concatenating the medians of the rotation vector, all 63 canonical keypoint coordinates, and all 63 expression coordinates in the head coordinate frame. During inference, it is constructed from the source image. Appendix C.1 describes how the model encodes it.

B Data Pipeline

XXXX-1 is trained on an internal corpus of dyadic conversational videos that is not included in this release. We construct it by progressively segmenting, filtering, tracking, separating, and annotating a larger collection of candidate videos, retaining 926 hours of unscripted conversation. Each retained segment contains synchronized left and right participant tracks, separate audio tracks, and LivePortrait motion annotations. Every pipeline stage saves its intermediate outputs. This lets us schedule CPU and GPU workloads separately, identify bottlenecks, and revise individual components without repeating earlier steps.

Figure 3 summarizes the data retained after each filtering stage.

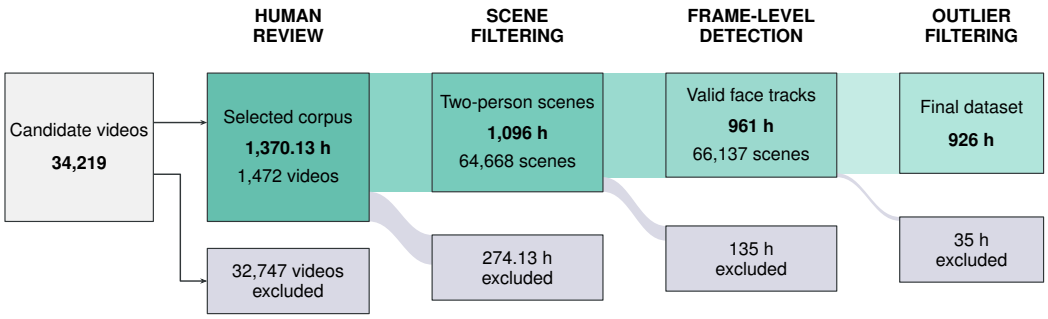

Figure 3: Data filtering cascade.

Source video selection. The source collection contained 34,219 candidate videos. Human annotators retained 1,472 videos, totaling 1,370.13 hours. Each retained video contained at least one segment in which exactly two visible participants conversed with each other, with one speaking and the other listening. Both faces had to be clearly visible from the front or at a slight angle and remain free of occlusion. Annotators excluded short-form videos, edited excerpts of already selected videos, and uncertain cases.

Scene detection. We split each retained video into scenes with PySceneDetect (Castellano, 2024), discard the first 90 seconds, and exclude any scene shorter than five seconds from face filtering. Within the remaining scenes, we sample every 25th frame and run a face detector, keeping faces above a minimum size of 160 pixels at 1080p resolution. When the face detector does not return exactly two faces, for example because it detects faces in background images, we use a YOLO person detector (Jocher & Qiu, 2023) as a fallback. We retain segments of at least five seconds in which every sampled frame passes this two-person check.

Face detection. For every frame in each segment, we define fixed participant regions over its left and right halves. A whole-body pose estimator processes both regions, and each participant's 68 facial landmarks must lie within the corresponding half and have a mean confidence above 0.3. We retain sequences of frames that satisfy these checks and last at least five seconds. This stage produces frame-level landmark annotations for the left and right participants, which the later stages consume.

Speaker separation. The source audio contains both voices on the same track. The motion model requires a separate stream for each participant. We convert the audio to 16 kHz mono and separate the two voices with ClearerVoice-Studio's pretrained audiovisual target-speaker extractor (Pan et al., 2023; 2025). The model also receives cropped face videos for the left and right participants and uses their lip motion to associate each output with the corresponding side. During training, we randomly select one participant as the generation target and treat their waveform as self audio and the other waveform as other audio.

Audio features. We encode the two separated waveforms independently with the original HuBERT encoder. Concatenating the two 1024-dimensional streams produces a 2048-dimensional vector per frame. We store one feature array per segment.

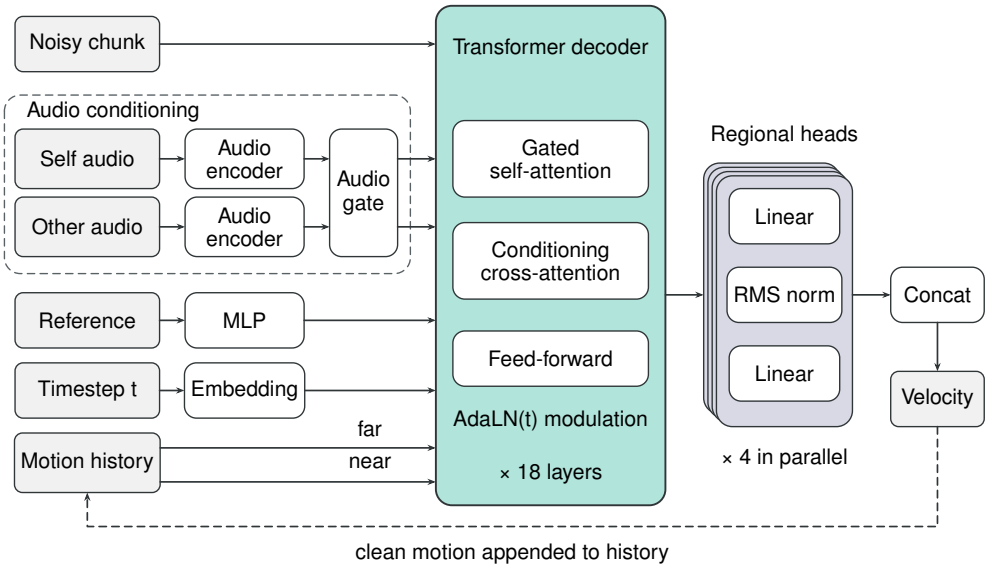

Figure 4: Overview of the XXXX-1 motion model.

Motion extraction. The facial landmarks from the preceding stage define a crop for each participant in every retained frame. We smooth the crop boxes over 25 frames, resize each face to 256×256, and process the left and right crops in the same batch with the LivePortrait motion extractor (Guo et al., 2024). The full extractor output is stored for both participant tracks. Appendix A describes how these annotations are converted into the training target and reference condition.

Training-window filtering. We exclude training windows with implausible motion. For each participant track, we compute first-, second-, and third-order temporal finite differences of the rotation and expression coordinates. We scale these differences per coordinate, average their absolute values, and smooth the resulting score over 21 frames. A frame is marked as an outlier when its score exceeds 0.15, and the mask extends five frames in either direction. We retain only 25-frame windows that do not overlap this mask. A segment is discarded if either participant track has no windows left.

C Motion model architecture and training details

C.1 Model Architecture

The XXXX-1 motion model is a 153M-parameter conditional flow-matching Transformer (Lipman et al., 2023) that autoregressively generates five-frame motion conditioned on two separate audio channels called self and other audio. Figure 4 shows the model architecture, and Table 4 lists its configuration.

Audio encoder. We use the 24-layer pre-trained HuBERT Large encoder, with roughly 300M parameters. The audio encoder produces 1024-dimensional features at 50 Hz. We average adjacent pairs to match the motion model's 25 fps rate. We extract features for two audio channels separately.

Transformer. The motion model is a pre-norm Transformer decoder with 18 layers, 8 heads, and width 512. Each layer applies three residual sub-blocks: self-attention, a conditioning block, and a feed-forward block. We restrict self-attention to the five-frame chunk and route the past motion and audio through the conditioning block. This keeps the cost per chunk independent of how much motion has already been generated. The flow-matching timestep modulates each sub-block through adaptive layer normalization (AdaLN), with separate scale and shift predicted for each. We zero-initialize the corresponding projections, so AdaLN initially reduces to standard LayerNorm. The noisy current

Table 4: Configuration of the released XXXX-1 motion model.

Property	Value
Parameters	153M
Decoder layers	18
Hidden dimension	512
Attention heads / head dimension	8 / 64
Feed-forward dimension	512
Audio condition encoder layers	2 per stream
Dropout	0
RoPE base	10^4
Generated motion chunk	5 frames
Past motion context	75 frames
Motion dimension	42
Audio context per stream (past / present / future)	75 / 5 / 5 feature steps
Audio feature dimension per stream	1024
Reference condition	129

chunk and the past motion history share a linear projection from 42 to 512 dimensions. Following work on attention stability (Dehghani et al., 2023), we apply RMSNorm (Zhang & Sennrich, 2019) to queries and keys before computing attention scores. Unlike QK normalization that shares scale parameters across heads (Esser et al., 2024), our formulation learns separate query and key scale vectors for every head. Each scale vector is parameterized as $(1 + \mathbf{w})$, with $\mathbf{w}$ initialized to zero. Every head therefore starts at unit gain, while weight decay regularizes its gain toward one rather than zero. The normalization controls attention logit growth while allowing each head to adjust its scale independently. Additionally, each self-attention applies a head-specific learned sigmoid gate to its output, initialized with zero weights and a bias of 2. This design follows the gated-attention scheme of Qiu et al. (2025), which was introduced to mitigate attention sinks. All attention modules use the same RoPE (Su et al., 2024) configuration, while positions are constructed independently for each query and key sequence.

Conditioning. We route the conditions through separate paths rather than fusing them in a single attention operation. Each block receives past motion, audio features for both channels, and a static reference. We divide the 75-frame motion history between a near path over the last five frames and a far path over the earlier 70. All paths use cross-attention except the reference, which uses an MLP. This separation prevents audio tokens from competing with motion and reference inputs in the same attention distribution. Let $\mathbf{A}^{(\ell)}_{\text{far}}$, $\mathbf{A}^{(\ell)}_{\text{near}}$, $\mathbf{A}^{(\ell)}_{\text{self}}$, and $\mathbf{A}^{(\ell)}_{\text{other}}$ denote the four cross-attention outputs in decoder layer ℓ, and let $\mathbf{R}^{(\ell)}$ denote the reference MLP output added to each token of the current chunk. The binary condition-availability vector $\mathbf{b} = (b_{\text{past}}, b_{\text{self}}, b_{\text{other}}, b_{\text{ref}}) \in \{0, 1\}^4$ indicates which conditions are retained. The masked outputs form the conditioning residual $\Delta\mathbf{H}^{(\ell)}_{\text{cond}}$, which is added to the hidden state after self-attention:

$$\begin{aligned}\Delta\mathbf{H}^{(\ell)}_{\text{cond}} = b_{\text{past}} \left(\mathbf{A}^{(\ell)}_{\text{far}} + \mathbf{A}^{(\ell)}_{\text{near}}\right) \\ + b_{\text{self}}\mathbf{A}^{(\ell)}_{\text{self}} + b_{\text{other}}\mathbf{A}^{(\ell)}_{\text{other}} + b_{\text{ref}}\mathbf{R}^{(\ell)}.\end{aligned} \tag{3}$$

The near and far history paths share b_{past}, so guidance treats past motion as a single condition.

Since automatic speaker separation in the training data is imperfect, we introduce an audio gate with direct speaker-activity supervision. Following the complementary weighting used in the Gated Multimodal Unit (Arevalo et al., 2017), the gate predicts channel-wise weights jointly from both streams but keeps the weighted representations separate. During training, we average the gate logits across channels and supervise the result with pseudo-VAD labels on frames where a single speaker is confidently active.

Regional output heads. The Transformer decoder produces five 512-dimensional tokens. We apply four linear projections to each token to create separate latent spaces for head rotation, brow, eyes,

and mouth. Each projection operates on the full 512-dimensional representation, while its output dimension is proportional to the number of motion coordinates in that region. This intermediate split allows classifier-free guidance (CFG) to be applied independently to the four regions. Each regional latent is then normalized by its RMS across the five frames and latent channels. Four additional linear layers map each regional latent to the corresponding motion coordinates. The outputs are concatenated to form the final 42-dimensional velocity prediction.

C.2 Training Procedure

We train the XXXX-1 motion model using the conditional flow-matching framework (Lipman et al., 2023). Training follows the autoregressive inference setup: each five-frame chunk is conditioned on earlier motion predictions and two audio streams. The objective combines a flow-matching velocity loss, temporal smoothness regularizers, and direct supervision of the audio gate. Training further uses correlated and immiscible noise and multi-condition dropout for guidance.

Audio feature extraction. We run the pre-trained HuBERT audio feature extractor once before training on full audio durations and store the results. This approach lets us sample training windows of different lengths without rerunning the encoder, which increases training efficiency.

Loss functions. Given a motion target $\mathbf{m}$ and Gaussian noise $\boldsymbol{\epsilon}$, we construct a noisy sample at timestep $t \in [0, 1]$ by linear interpolation:

$$\mathbf{m}_t = (1 - t)\,\boldsymbol{\epsilon} + t\,\mathbf{m}. \tag{4}$$

The model predicts a velocity $\mathbf{v}_\theta(\mathbf{m}_t, t)$. The ground-truth target is the path velocity $\frac{d\mathbf{m}_t}{dt} = \mathbf{m} - \boldsymbol{\epsilon}$. We compute the mean-squared error separately for head rotation, brow, eyes, and mouth, then sum the four regional losses:

$$\mathcal{L}_{\mathrm{CFM}} = \mathbb{E}_{t,\mathbf{m},\boldsymbol{\epsilon}} \sum_{r=1}^{4} \mathrm{MSE}\left(\mathbf{v}_\theta^{(r)}(\mathbf{m}_t, t),\, \mathbf{m}^{(r)} - \boldsymbol{\epsilon}^{(r)}\right). \tag{5}$$

Averaging within each region gives all four regions equal weight regardless of their coordinate count. We sample one timestep per five-frame chunk from a logit-normal distribution (Esser et al., 2024) and share it across the frames of the chunk, matching the chunkwise generation performed at inference. We also compute the cosine distance $\mathcal{L}_{\cos}$ between the predicted and target velocities separately for the same four regions. This term constrains the direction of the predicted flow independently of its magnitude.

To encourage smooth motion, we regularize the expression coordinates, head rotation, and reconstructed 3D keypoint motion separately. Let $\hat{\mathbf{e}}$ denote the predicted expression coordinates, $\hat{\mathbf{R}}$ the predicted rotations, and $\hat{\mathbf{q}}$ the selected 3D keypoints obtained by adding $\hat{\mathbf{e}}$ to the reference canonical keypoints and applying $\hat{\mathbf{R}}$. We define $\boldsymbol{\omega}_t$ as the axis-angle vector of the relative rotation $\hat{\mathbf{R}}_{t-1}^{\mathsf{T}}\hat{\mathbf{R}}_t$ and minimize

$$\begin{aligned}\mathcal{L}_{\mathrm{reg}} = {} & \lambda_{\mathrm{exp}}\left(\|\Delta\hat{\mathbf{e}}\|_2^2 + \|\Delta^2\hat{\mathbf{e}}\|_2^2 + \|\Delta^3\hat{\mathbf{e}}\|_2^2\right) \\ & + \lambda_{\mathrm{rot}}\left(\|\boldsymbol{\omega}\|_2^2 + \|\Delta\boldsymbol{\omega}\|_2^2 + \|\Delta^2\boldsymbol{\omega}\|_2^2\right) \\ & + \lambda_{\mathrm{3D}}\left(\|\Delta\hat{\mathbf{q}}\|_2^2 + \|\Delta^2\hat{\mathbf{q}}\|_2^2 + \|\Delta^3\hat{\mathbf{q}}\|_2^2\right),\end{aligned} \tag{6}$$

where Δ is the temporal finite-difference operator. The coefficients λ_{exp}, λ_{rot}, and λ_{3D} weight the three regularization terms. Each squared norm is averaged over frames and coordinates. We prepend the last three frames of the motion history before computing these terms, so the regularization extends across chunk boundaries.

We supervise the audio gate described in Appendix C.1 on high-confidence single-speaker frames identified by a frozen voice-activity detector. A binary cross-entropy loss $\mathcal{L}_{\mathrm{VAD}}$ on the channel-averaged gate logit uses a target of 1 when self audio is active and 0 when other audio is active. This auxiliary loss helps suppress cross-speaker leakage caused by imperfect speaker separation. The total training objective is

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{CFM}} + \mathcal{L}_{\cos} + \mathcal{L}_{\mathrm{reg}} + \lambda_{\mathrm{VAD}}\mathcal{L}_{\mathrm{VAD}}, \tag{7}$$

where λ_{VAD} controls the weight of the audio-gate supervision.

Noise design. We use progressive noise (Ge et al., 2023), which forms an autoregressive noise trajectory over the motion sequence. Each frame's noise is correlated with the preceding frame, and the trajectory continues across five-frame chunk boundaries. During training, we also apply immiscible noise assignment (Li et al., 2024) independently to head rotation, brow, eyes, and mouth. Within each batch, it reassigns the noise trajectories to reduce their distance from the corresponding regional targets.

Autoregressive rollout. Each 25-frame training target is divided into five successive chunks. For each chunk, the model predicts its velocity and forms a one-step clean estimate $\hat{\mathbf{m}} = \mathbf{m}_t + (1 - t)\,\mathbf{v}_\theta(\mathbf{m}_t, t)$. A teacher-forcing curriculum determines whether the rolling history is updated with the ground-truth chunk or this estimate. During the first epoch, all predictions use ground-truth history. We then begin replacing ground-truth updates with model estimates, starting from the final chunk. Each epoch moves this boundary one chunk earlier. After each update, we remove the oldest five frames to keep the history at 75 frames. Ground-truth chunks are marked with $t = 1$, while each estimate retains the sampled timestep used to construct it. We add a learned embedding of this value to the past-motion tokens. This lets the decoder distinguish ground truth from estimates formed at different timesteps. Conditioning on its own outputs during training reduces exposure bias between training and autoregressive inference.

Guidance dropout. We train the model for CFG (Ho & Salimans, 2022) by randomly masking the condition paths in Equation 3. We retain self audio, other audio, and the reference together for 45% of samples and drop all three for 15%. The remaining samples retain one or two of these conditions. Past-motion history is masked separately and retained for 75% of samples. The model therefore learns to operate with any subset of the four conditions. At inference, the past-only prediction forms the baseline, and self audio, other audio, and the reference each add a guidance residual scaled independently for each motion region.

Optimization. We set the objective coefficients $\lambda_{\text{exp}} = 0.1$, $\lambda_{\text{rot}} = \lambda_{\text{3D}} = 100$, and $\lambda_{\text{VAD}} = 0.01$. We optimize with Adan (Xie et al., 2024) using three weight-decay groups: 0.05 for most matrix weights, 0.01 for the AdaLN, timestep, and reference projections, and zero for one-dimensional parameters, biases, self-attention gates, and quality embeddings. The learning rate increases linearly over 5,000 warm-up steps to 3×10^{-4}, then follows a 200,000-step cosine decay toward 10^{-5}. We maintain an exponential moving average of the model weights with decay 0.995 and use the averaged weights in the released checkpoint. The checkpoint was trained for 200,000 steps on a single NVIDIA GH200 in approximately 35 hours. Training uses float32 precision with TF32 matrix multiplication.

D Audio encoder training details

D.1 Chunk-based Audio Encoder

The motion generation model is conditioned on two separate audio channels. During training, we extract features using HuBERT (Hsu et al., 2021) on full-length audio. But, during real-time streaming inference, we can't do that, because audio arrives incrementally, so the encoder must process audio in chunks. However, because HuBERT uses bidirectional attention, directly encoding short chunks of audio produces features that significantly differ from their full-context counterparts. To solve this, we build a chunk-based encoder through self-distillation of the original model without changing its architecture. We set the input chunk to a duration of 525 ms, semantically split into 120 ms past, 200 ms present, and 205 ms future windows, or 3, 5, and 5 video frames respectively. The extra 5 ms of future audio is required by HuBERT's convolutional feature extractor to produce exactly 26 feature vectors at 50 Hz that will be downsampled to 13 to match the number of video frames.

Feature drift. We quantify the drift between fixed-window features and full-context features for the original HuBERT. For a given present window, the features computed only on the window itself differ from the same window features computed on full audio by 30–60% in relative L_1 distance. Fidelity to the full-context features reaches a plateau at approximately 2 s of past audio and drops sharply with shorter past context. At least 400 ms of future audio is required to avoid severe degradation. This future lookahead exceeds the latency target, while processing two seconds of past audio increases encoder work for each chunk.

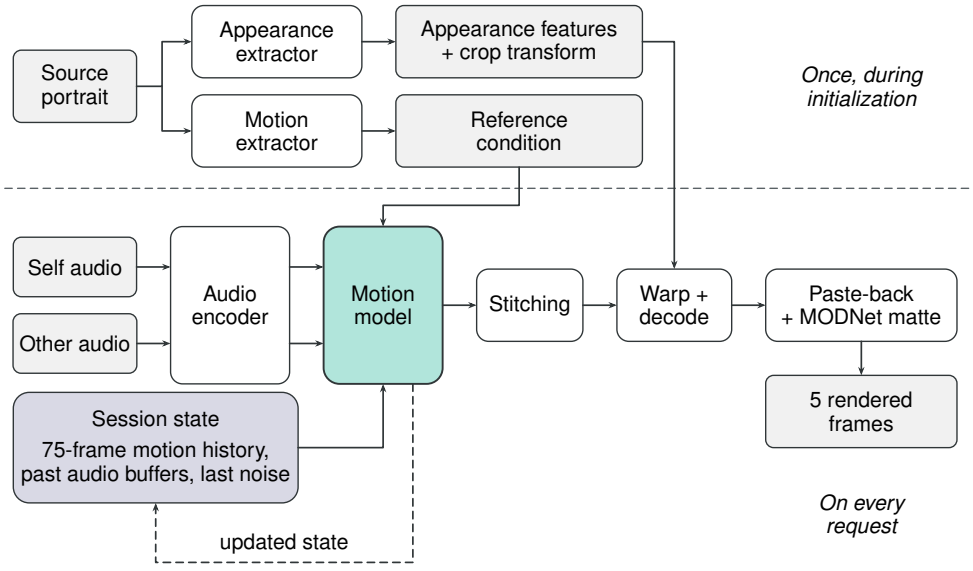

Figure 5: Overview of XXXX-1 inference.

Self-distillation. Adapting a pretrained bidirectional speech encoder for chunk-based inference can require architectural changes and continued self-supervised pretraining (Fu et al., 2024). We instead retain HuBERT's architecture and fine-tune the encoder to match its own full-context features from a fixed input chunk. The teacher is the frozen original encoder applied to the full-length audio. We initialize the student from the same weights and apply it to an input of 525 ms duration. Let $\mathbf{h}_i^{\text{chunk}}, \mathbf{h}_i^{\text{full}} \in \mathbb{R}^d$ denote the student and teacher features at frame i, where $d = 1024$. We minimize the mean-squared error over the $N = 13$ frames of the window:

$$\mathcal{L}_{\text{distill}} = \frac{1}{Nd} \sum_{i=1}^{N} \left\| \mathbf{h}_i^{\text{chunk}} - \mathbf{h}_i^{\text{full}} \right\|_2^2 . \tag{8}$$

Under this window, self-distillation reduces the relative L_1 error from approximately 50% to 12%.

Training. The encoder is trained on the English partition of Multilingual LibriSpeech (Pratap et al., 2020) and on the same dataset used to train the motion model. The convolutional feature extractor remains frozen, while the feature projection and Transformer are optimized with AdamW (learning rate 10^{-3} and weight decay 10^{-4}). Training uses a Noam schedule with 20,000 warm-up steps, bfloat16 precision, and batch size 60, for approximately 200 epochs. For silence robustness, we replace 10% of the input windows with silence or low-amplitude noise and supervise them with a precomputed sequence of full-context silence features. A separate augmentation adds low-amplitude Gaussian noise to a subset of speech windows.

E Renderer inference and request state

E.1 Renderer

We call the inference-side component that serves the motion model the renderer. It registers the source portrait once, then generates five frames at a time. Each request predicts a motion chunk, renders it with the cached LivePortrait appearance (Guo et al., 2024), and returns the updated state required by the next request. Figure 5 shows the pipeline; Table 6 reports its per-chunk request processing duration and time to first frame across GPUs.

Session initialization. Before generation, a face detector and two landmark models locate the face and define its crop. The LivePortrait motion extractor predicts the source motion parameters, while



Figure 6: Qualitative effect of noise truncation. Representative frames generated without truncation (top) and with threshold $\tau = 1.2$ (bottom). Truncation reduces extreme facial deformations and head rotations while preserving natural motion.

the appearance extractor produces the volumetric feature used for rendering. We cache these outputs together with the source frame, crop transform, and pasteback mask. The source motion also provides the reference condition described in Appendix A.

Audio context. The audio encoder and motion model use different context lengths. The encoder always receives a 525 ms window, while the motion model conditions on a longer window that includes a 75-frame audio history. For each motion chunk, the two audio channels are batched into one audio encoder call. We keep its 5 present and 5 future features and discard the 3 past outputs, because those are already cached from earlier calls. The runtime prepends 75 cached audio features as the past, so the motion model's audio window spans 75 past, 5 present, and 5 future frames per stream. After generation, the 5 present features from both streams are added to the cache. The 5 future features condition the current chunk but are not cached; the encoder recomputes them when they become present on the next step. This keeps the encoder's input fixed at a 13-frame window while preserving the 75-frame audio history the motion model requires.

Noise initialization and flow sampling. At inference, we use the progressive noise described in Appendix C.2 with correlation parameter $\alpha = 2$. We additionally apply noise truncation with threshold $\tau = 1.2$, bounding every coordinate to $[-\tau, \tau]$. The truncation interval at each frame depends on the preceding noise value, preserving the progressive correlation under this bound. In our qualitative examples, truncation suppresses extreme expression deformations and head rotations observed without it (Figure 6). We integrate the learned velocity field using an Euler ODE solver with four model evaluations per chunk. The normalized prediction then replaces the oldest five frames in the motion history.

Classifier-free guidance. At inference, we apply classifier-free guidance in the regional latent spaces, using the past-only representation as the baseline. At each solver step, the model evaluates four combinations in one batch: past motion alone and past motion with self audio, other audio, or the reference condition. Let $\boldsymbol{\ell}_p^{(q)}$, $\boldsymbol{\ell}_{\text{self}}^{(q)}$, $\boldsymbol{\ell}_{\text{other}}^{(q)}$, and $\boldsymbol{\ell}_{\text{ref}}^{(q)}$ denote the resulting latent representations for region $q \in \{\text{rotation}, \text{brow}, \text{eyes}, \text{mouth}\}$. We compute

$$\begin{aligned}\widetilde{\boldsymbol{\ell}}^{(q)} = \boldsymbol{\ell}_p^{(q)} &+ w_{\text{self}}^{(q)} \left(\boldsymbol{\ell}_{\text{self}}^{(q)} - \boldsymbol{\ell}_p^{(q)}\right) \\ &+ w_{\text{other}}^{(q)} \left(\boldsymbol{\ell}_{\text{other}}^{(q)} - \boldsymbol{\ell}_p^{(q)}\right) \\ &+ w_{\text{ref}}^{(q)} \left(\boldsymbol{\ell}_{\text{ref}}^{(q)} - \boldsymbol{\ell}_p^{(q)}\right).\end{aligned} \tag{9}$$

Here, the weights $w_{\text{self}}^{(q)}$, $w_{\text{other}}^{(q)}$, and $w_{\text{ref}}^{(q)}$ scale the three guidance residuals independently for each region. After regional RMS normalization, a linear projection produces the velocity prediction for region q. The released runtime API uses one value per condition and applies it across all four regions.

Rendering. After flow sampling, we de-normalize the 42-dimensional motion and convert the axis-angle rotation to a rotation matrix. We combine the predicted rotation and 39 expression coordinates with the source parameters excluded from the training target (Appendix A), producing a set of driving keypoints for each frame. The LivePortrait stitching network refines these keypoints based on the source keypoints. The renderer then uses the refined keypoints to warp the cached appearance and decode the face crop. It pastes the crop back into the full frame. MODNet (Ke et al., 2022) predicts an alpha matte for blending the rendered avatar with the selected background.

Session state. We maintain a separate mutable state for each session. At the first request, its 75-frame motion history repeats the source motion, its audio buffers start from zero, and it has no previous noise frame. After every request, the state retains the motion history, 75 frames of dual-stream audio features, the 120 ms of past audio for both channels required by the audio encoder, and the final noise frame. These tensors occupy approximately 0.64 MB. The in-process runtime keeps the state on the GPU between chunks, while the released HTTP API serializes it between requests.

The TensorRT runtime configuration and renderer benchmarks are provided in Appendix H.

F SCHEDULER TRADE-OFFS

Three of the four terms in the latency contribution decomposition follow from scheduling choices that could be made differently. R1 accepts a bounded amount of silence padding (T_{pad}) to keep latency low, rather than waiting for a full window of speech. Waiting would remove T_{pad} but add a fixed 200 ms, which is worse in expectation for any first-fragment duration distribution with mean below 200 ms. Right-padding instead of left-padding would also remove T_{pad}, but would introduce a pause within the speech and feed the motion generation model out-of-domain data. A render-ahead depth of one chunk fixes T_{present} at 200 ms. Discarding only unadmitted speech bounds the ILC by exactly the duration of the committed present window. Discarding committed media as well would lower the floor but violate the motion generator's expectation that the future window always becomes the present window. Only T_{lead} is a free parameter: an adaptive implementation could track the online TTFF distribution and set T_{lead} to, for instance, its 99th percentile, matching the lead to the serving hardware and its load rather than to a single fixed measurement.

G QUALITATIVE COMPARISON

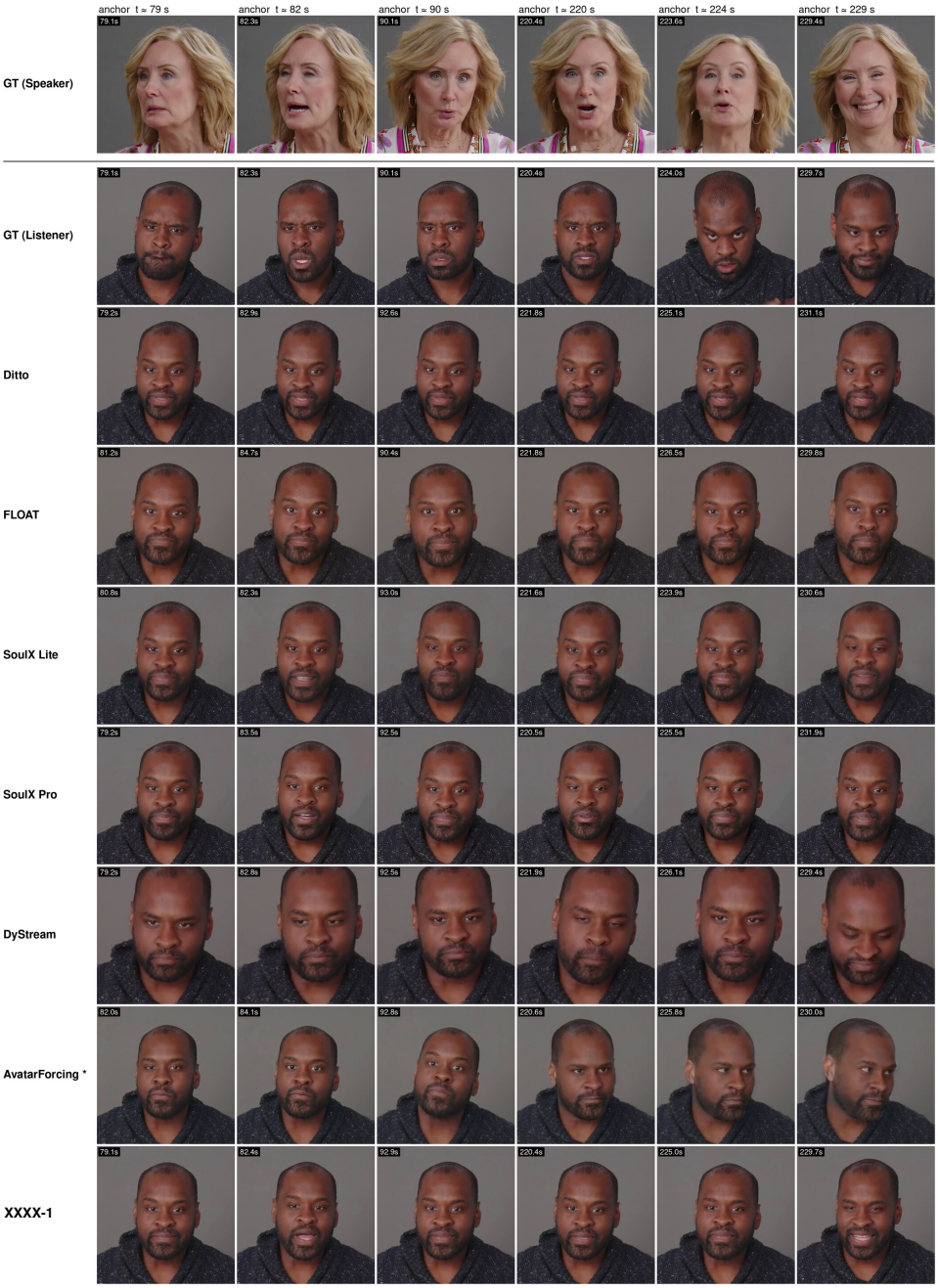

Figure 7: Qualitative comparison of reactive listening on the Seamless Interaction test set. The first two rows show the recorded speaker and listener, followed by frames generated for the listener by each evaluated system. Columns are anchored to the speaker times shown above the first row; inset labels report the selected timestamp for each video.

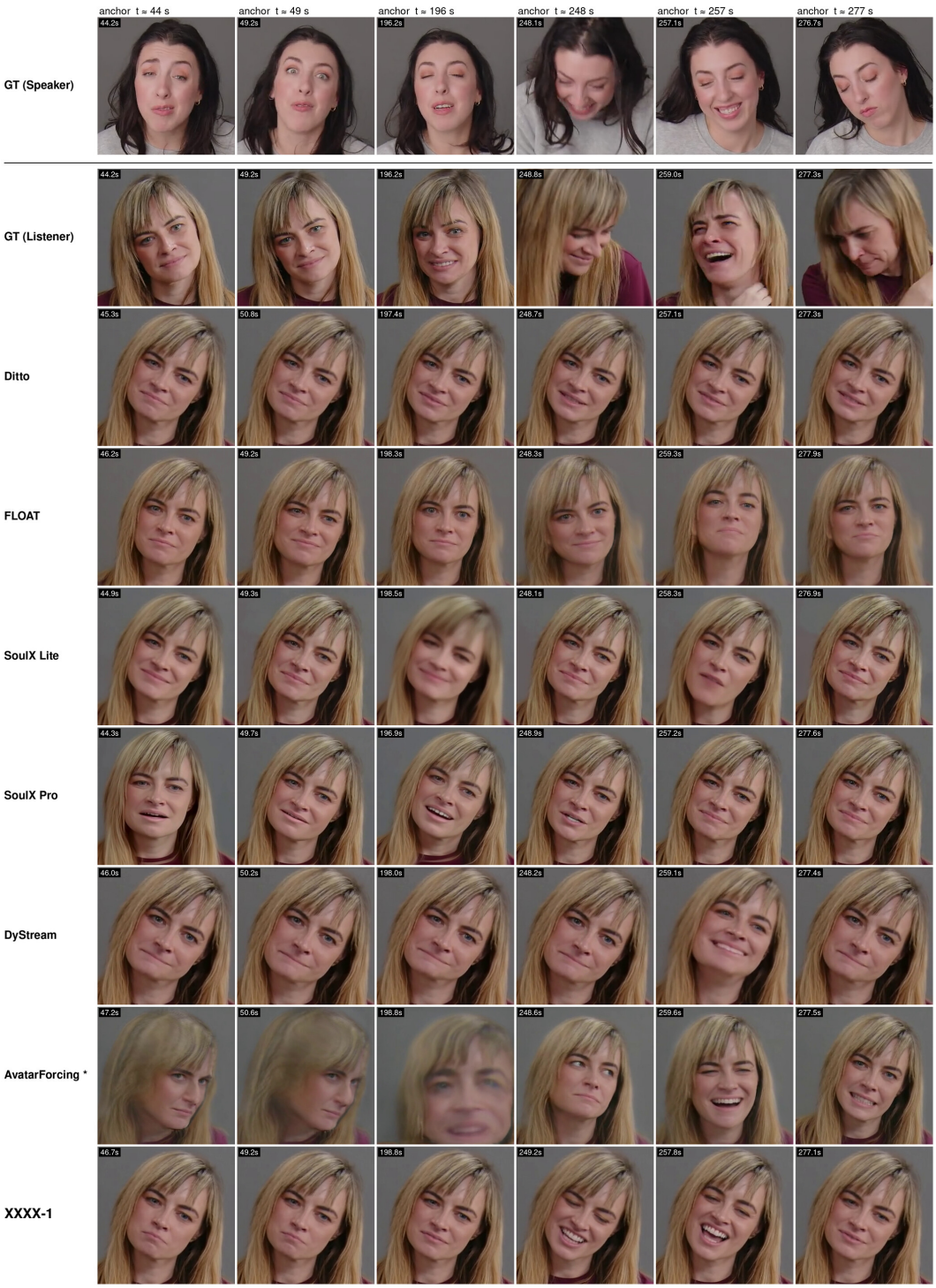

Figure 8: Additional qualitative comparison of reactive listening on the Seamless Interaction test set. The first two rows show the recorded speaker and listener, followed by frames generated for the listener by each evaluated system. Columns are anchored to the speaker times shown above the first row; inset labels report the selected timestamp for each video.

H EVALUATION AND RUNTIME PROTOCOLS

H.1 EVALUATION PROTOCOL

The evaluated subset uses vendor `V00`, interaction type `ipc_conversation`, label `improvised`, and split `test`. It contains 184 videos comprising 1,284,704 frames (11.90 hours) at 30 fps.

Motion trajectories are resampled to 25 fps without normalization. For the reported table, we use the listening variant, retaining listener-silent frames identified by voice activity detection (VAD). We split the EMOCA features into pose dimensions 0–5 and expression dimensions 6–55.

For each video, after flattening time and feature dimensions, we compute $\mathrm{rPCC} = |\rho(\mathbf{x}, \mathbf{y}) - \rho(\mathbf{x}, \widehat{\mathbf{y}})|$, where $\mathbf{x}$ is speaker motion, $\mathbf{y}$ is recorded listener motion, and $\widehat{\mathbf{y}}$ is generated listener motion. Lower rPCC indicates that the generated listener reproduces the recorded pair's instantaneous motion correlation more closely; it does not indicate stronger coupling. For each video, PFD is the Fréchet distance between the concatenated motion $[\mathbf{x}, \mathbf{y}]$ and $[\mathbf{x}, \widehat{\mathbf{y}}]$. The table reports rPCC and PFD averaged over the 184 videos.

SID is the Shannon entropy, in bits, of generated-listener assignments to k-means clusters fitted on ground-truth motion, using 20 clusters for pose and 40 for expression. Assignments are pooled over the corpus. Variance is computed separately for each region by pooling values over frames and dimensions across the corpus. SID and variance describe motion variety and magnitude and have no preferred direction.

CSIM and LSE are computed per video and averaged over the 184 videos. FID and FVD are computed from features pooled over the SI-184 subset and compared with the corresponding pooled ground-truth features.

H.2 TENSORRT EXECUTION

The released runtime uses TensorRT engines for audio encoding, motion generation, and rendering. Table 5 lists their precision, batch size, and number of calls per chunk. The source motion extractor runs once in fp32 during session initialization and is therefore omitted.

Table 5: The compiled per-chunk engines, with the batch size at which each runs and the number of calls per five-frame chunk. All run in fp16, with normalization layers held in fp32.

Engine	Precision	Batch	Calls per chunk
Audio encoder (HuBERT)	fp16	2	1
Motion encoder	fp16	1	1
Motion decoder	fp16	1	4
Appearance warp	fp16	5	1
Stitch	fp16	5	1
Face decoder	fp16	1	5
Matting (MODNet)	fp16	1	5

During each request, the session state and intermediate tensors remain on the GPU and are reused throughout flow sampling. The engines use module-specific batching and execute on a single CUDA stream without CPU synchronization.

H.3 RENDERER BENCHMARKS

For Table 6, we define real-time throughput as the 200 ms output duration divided by the request latency.

H.4 LATENCY MEASUREMENT PROTOCOL

To measure the latency terms in Table 3, we run an interactive session on the server and redirect its standard output to a log file:

Table 6: Per-chunk latency, real-time throughput, and time to first frame (TTFF) across GPUs, measured offline on one five-frame (200 ms) chunk.

GPU	TTFF	Request latency	Real-time throughput
NVIDIA L40S	42 ms	71 ms	2.81×
NVIDIA A100	54 ms	91 ms	2.20×
NVIDIA L4	—	202 ms	0.99×
NVIDIA RTX 4060 Ti	99 ms	162 ms	1.24×
NVIDIA RTX 3070	94 ms	180 ms	1.11×
NVIDIA RTX 3060 Ti	—	207 ms	0.97×
NVIDIA RTX 4060	—	232 ms	0.86×

```
pixi run interactive-demo > run.log
```

For each avatar reply and each interruption, the streamer writes one line to standard output that holds the four terms. We then parse these lines with the released script `scripts/latency_table.py`, through its pixi runner:

```
pixi run latency-table run.log
```

The script reads one or more log files, or standard input. It pairs each reply with its interruption by segment id, keeps the first n complete exchanges (default $n = 10$), and prints the mean and $[\min, \max]$ of each term for the response and the interruption latency contributions.

I Additional Evaluation

Tables 1 and 2 report metrics computed from EMOCA features on the VAD-masked listening portion of the SI-184 subset. The full protocol uses all successfully tracked frames, whereas the VAD-masked protocol retains only frames on which the listener is classified as silent. We additionally report full-sequence EMOCA results (Table 7) and both VAD-masked and full-sequence LivePortrait results (Tables 8 and 9). We also report speaker–listener time-lagged cross-correlation (TLCC) for both representations under both settings (Tables 10 and 11). The asterisk on Avatar Forcing denotes the context-length extension described in Section 4.1.

I.1 Motion Fidelity and Diversity

Table 7: Quantitative comparison on SI-184 subset using EMOCA features over full sequences. The pose components of PFD and variance are in units of 10^{-2}. For metrics without an arrow, values closer to ground truth are better.

		PFD ↓		rPCC ↓		SID		Variance	
System	Dyadic	Pose	Exp	Pose	Exp	Pose	Exp	Pose	Exp
SoulX Lite	✗	5.988	19.94	0.148	0.061	3.419	5.065	0.975	1.411
SoulX Pro	✗	4.751	18.75	0.125	0.056	3.499	5.103	0.982	1.441
Ditto	✗	6.645	28.82	0.180	0.074	3.459	4.953	1.341	1.671
FLOAT	✗	3.953	23.71	0.128	0.077	3.220	4.816	0.811	1.206
Avatar Forcing*	✓	**5.891**	27.78	0.145	0.080	**3.917**	4.887	1.800	1.337
DyStream	✓	7.106	33.32	0.147	0.101	3.840	4.731	**1.673**	1.300
XXXX-1 (Ours)	✓	5.924	**19.85**	**0.136**	**0.063**	3.478	**5.096**	0.900	**1.432**
Ground truth	—	0.000	0.00	0.000	0.000	4.116	5.251	1.700	1.552

Table 8: Quantitative comparison on SI-184 subset using LivePortrait features over VAD-masked frames. PFD rotation and expression components are in units of 10^{-2} and 10^{-3}, respectively; variance rotation and expression components are in units of 10^{-2} and 10^{-5}. For metrics without an arrow, values closer to ground truth are better.

		PFD ↓		rPCC ↓		SID		Variance	
System	Dyadic	Rot	Exp	Rot	Exp	Rot	Exp	Rot	Exp
SoulX Lite	✗	4.018	0.685	0.188	0.098	3.140	4.914	0.857	3.600
SoulX Pro	✗	3.332	0.633	0.170	0.094	3.271	4.997	1.013	3.700
Ditto	✗	4.106	0.882	0.174	0.109	3.010	4.930	1.002	3.800
FLOAT	✗	3.240	0.878	0.152	0.127	3.061	4.488	0.852	4.300
Avatar Forcing*	✓	5.784	1.242	0.175	0.138	**3.813**	**4.802**	3.352	5.200
DyStream	✓	4.392	1.335	0.178	0.151	3.436	4.421	**1.239**	**4.300**
XXXX-1 (Ours)	✓	**3.967**	**0.671**	**0.169**	**0.112**	3.152	4.760	0.920	3.300
Ground truth	—	0.000	0.000	0.000	0.000	3.745	5.171	1.900	4.000

Table 9: Quantitative comparison on SI-184 subset using LivePortrait features over full sequences. PFD rotation and expression components are in units of 10^{-2} and 10^{-3}, respectively; variance rotation and expression components are in units of 10^{-2} and 10^{-5}. For metrics without an arrow, values closer to ground truth are better.

		PFD ↓		rPCC ↓		SID		Variance	
System	Dyadic	Rot	Exp	Rot	Exp	Rot	Exp	Rot	Exp
SoulX Lite	✗	3.867	0.554	0.179	0.092	3.133	5.021	0.854	3.400
SoulX Pro	✗	3.089	0.521	0.154	0.091	3.309	5.070	1.044	3.500
Ditto	✗	4.300	0.836	0.177	0.123	3.027	5.012	1.004	3.700
FLOAT	✗	2.764	0.686	0.144	0.129	3.252	4.830	0.865	3.800
Avatar Forcing*	✓	4.653	1.043	**0.152**	0.143	**3.830**	4.854	2.947	4.700
DyStream	✓	4.349	1.155	0.172	0.168	3.458	4.544	**1.212**	**3.900**
XXXX-1 (Ours)	✓	**3.824**	**0.567**	0.165	**0.110**	3.191	**5.043**	0.922	3.200
Ground truth	—	0.000	0.000	0.000	0.000	3.862	5.219	2.027	4.000

I.2 TIME-LAGGED CROSS-CORRELATION

Tables 10 and 11 report the mean signed peak time-lagged cross-correlation (TLCC) between the recorded speaker and the listener, with lags searched over ± 49 frames at 25 fps. These are correlation values, not absolute lag offsets. We report each feature region and the combined feature vector (All), for both full sequences and VAD-masked frames. Values are averaged over the 184 videos.

Table 10: Mean signed TLCC peak on SI-184 using EMOCA features, in units of 10^{-2}.

		Full			Masked		
System	Dyadic	Pose	Exp	All	Pose	Exp	All
SoulX Lite	✗	1.00	−0.34	−0.37	3.49	2.07	1.92
SoulX Pro	✗	0.76	0.12	0.02	3.02	2.12	1.93
Ditto	✗	−0.38	0.29	0.06	1.99	2.12	1.85
FLOAT	✗	0.43	−1.09	−1.08	3.33	2.01	1.89
Avatar Forcing*	✓	0.09	0.17	0.01	2.51	2.25	2.02
DyStream	✓	0.43	−0.81	−0.89	3.61	2.01	1.87
XXXX-1 (Ours)	✓	0.52	−0.09	−0.20	3.57	2.34	2.17
Ground truth	—	1.61	1.70	1.50	4.82	3.69	3.48

Table 11: Mean signed TLCC peak on SI-184 using LivePortrait features, in units of 10^{-2}.

		Full			Masked		
System	Dyadic	Rot	Exp	All	Rot	Exp	All
SoulX Lite	✗	3.38	−0.21	−0.13	4.91	1.50	1.46
SoulX Pro	✗	2.61	−0.01	0.02	4.42	1.70	1.63
Ditto	✗	1.60	0.15	0.10	2.70	1.45	1.30
FLOAT	✗	3.75	−0.10	0.02	5.42	1.87	1.82
Avatar Forcing*	✓	1.34	0.72	0.64	2.86	1.71	1.57
DyStream	✓	3.76	−0.07	0.04	5.92	1.79	1.80
XXXX-1 (Ours)	✓	3.45	0.13	0.18	5.23	1.90	1.84
Ground truth	—	6.50	1.86	1.96	9.15	3.34	3.43

J BROADER IMPACTS

XXXX-1 can support interactive sales training, customer support, and in-store assistance, providing scalable practice and improving access to product and service information. However, realistic avatars may be misused for non-consensual impersonation or deceptive synthetic media. Our release terms prohibit these uses and require machine-generated content to be disclosed.

K LICENSES FOR EXISTING ASSETS

Datasets. The Seamless Interaction dataset (Agrawal et al., 2025), used only for evaluation, is available under CC BY-NC 4.0. The English partition of Multilingual LibriSpeech (Pratap et al., 2020), used for audio encoder training, contains recordings in the public domain and under CC BY 4.0. The internal dyadic corpus used to train the motion model is not released or redistributed with our artifacts.

Models and source code. The released runtime uses HuBERT (Hsu et al., 2021), MODNet (Ke et al., 2022), and a TensorRT grid sampling plugin under Apache-2.0. LivePortrait (Guo et al., 2024) and S2G-MDDiffusion, from which early motion model development began, are used under the MIT License. InsightFace code is available under the MIT License, while its pretrained models are limited to non-commercial research use. Our artifact preserves the corresponding notices and does not redistribute the LivePortrait or InsightFace checkpoints.

For comparison, we use the authors' public releases of Ditto (Li et al., 2025) and SoulX-FlashHead (Yu et al., 2026) under Apache-2.0, Avatar Forcing (Ki et al., 2026) under CC BY-NC 4.0, and FLOAT (Ki et al., 2025) under CC BY-NC-ND 4.0. The public DyStream (Chen & Liu, 2025) release does not state a license. We use it unchanged for non-commercial academic evaluation and do not redistribute its code or weights. EMOCA (Daněček et al., 2022) is used under its license for non-commercial scientific research, and SyncNet (Chung & Zisserman, 2017) is used under the MIT License. These external baselines and evaluation models are not included in the released XXXX-1 artifacts.