Appendix Contents

A. Border Impact

Our proposed framework opens up new opportunities for controllable and interpretable VT2A generation. By enabling fine-grained instruction-guided sound control, it can benefit multiple downstream applications, including video editing, film post-production, and accessible multimedia creation. The data curation pipeline for ***EchoFoley-6k*** and the *EchoVidia* model together provide a scalable data generation pipeline for synthesizing high-quality sounding videos with rich, aligned multimodal annotations. Such controllable generation pipelines not only enhance content creation but also serve as a valuable source of training data for large-scale *world models* and *omni-modal foundation models*, which aim to unify perception, reasoning, and generation across modalities.

B. Details on Benchmark: EchoFoley-6k

B.1. Data Example

Each data point in EchoFoley-6k follows the triplet format ⟨video, instruction, sounding-event annotations⟩.

- **Video**: a motion-centered clip (6–30 seconds) where sound-producing interactions are visually evident.
- **Instruction**: a natural-language directive indicating how sounds should be generated, edited, or transformed. Instructions may specify instance-, group-, or video-level control (e.g., "change the second meow into a lion roar" or "make all prior sounds louder").
- **Sounding-Event Annotations**: a set of symbolic sounding events $e = (t, d, p)$, where $t = (t_{\text{start}}, t_{\text{end}})$ denotes the temporal span, d is a semantic description, and p contains controllable auditory attributes (pitch, timbre, loudness, intensity, spatialization).

Figure 1 in the main paper shows an example illustrating how temporal boundaries, semantic descriptions, and controllable properties interact with instructions.

B.2. Comparison with Existing Datasets

Existing audio–visual datasets primarily support coarse audio modeling and do not provide fine-grained event boundaries or hierarchical controllability. EchoFoley-6k fills this gap by jointly providing:

- dense event-level temporal boundaries,
- symbolic, multi-attribute sounding-event representations,
- natural-language instructions for hierarchical control.

Dataset	Data		Annotation			
	Video	Instruction	Audio Desc.	Audio Temp. Cap.	Audio Vol. Cap.	Event-Centric Ann.
VGGSound [6]	**Yes**	No	**Yes**	No	No	No
Ego4Dsounds [5]	**Yes**	No	**Yes**	No	No	No
AudioSet [16]	No	No	**Yes**	No	No	No
AudioCaps [21]	No	No	**Yes**	No	No	No
WavCaps [30]	No	No	**Yes**	No	No	No
BBC Sound Effects [1]	No	No	**Yes**	No	No	No
AudioCoT [27]	**Yes**	No	**Yes**	No	No	No
Kling-Audio-Eval [42]	**Yes**	No	**Yes**	No	No	No
MovieGen Bench [32]	**Yes**	No	**Yes**	No	No	No
AVVP [40]	**Yes**	No	**Yes**	**Yes**	No	No
ASVA [49]	**Yes**	No	**Yes**	**Yes**	No	No
MA-Bench [33]	**Yes**	No	**Yes**	**Yes**	**Yes**	No
EchoFoley-6k	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Table 4. Comparison of datasets and available annotation types.

B.3. Data Curation Details

The full data construction pipeline is illustrated in Figure 2 of the main paper. Below we describe each stage in detail.

B.3.1. Curation Process Details

1. **Video Filtering**. We begin with motion-centered videos containing visually identifiable sound-producing actions (e.g., collisions, vocalizations, material interactions). Videos with ambiguous, off-screen, or purely ambient sounds are removed to ensure that sounding events remain visually grounded.
2. **Metadata and Frame Captioning**. For each candidate video, we extract its metadata (title and textual description) and generate frame-level visual captions at 16 fps using Gemini 2.5 pro[10]. These captions summarize evolving object configurations, motions, and interactions, providing structured visual grounding for downstream event extraction and reasoning.
3. **Story Proposal and Event Extraction**. Using GPT-5[31], we generate imaginative narratives describing plausible auditory interpretations of the scene. The model also proposes an initial list of hypothesized sounding events, each with an approximate temporal region and a short semantic description. These machine-generated proposals serve as high-level scaffolds and are not treated as final annotations.
4. **Human Refinement**. Human annotators transform the model-generated narratives into executable, fine-grained instructions; refine all event boundaries using frame-by-frame inspection; remove any hallucinated or visually unsupported events; and annotate rich, multi-attribute sound properties (e.g., pitch, timbre, volume, intensity, spatialization). This stage yields temporally accurate and instruction-aligned sounding-event annotations of high quality.

B.3.2. Human Refinement Instruction

Context. Human annotators verify and refine AI-generated instructions and their corresponding manipulated sound annotations. The goal is to ensure that the final dataset is logically coherent, unambiguous, and free of AI errors.

General Principles. Annotators are provided with a silent video, an AI-generated instruction, and the corresponding manipulated annotation file. They are asked to validate the following three aspects:

- *Logical Coherence.* The instruction must be plausible within the visual world of the video. The described sound should be something that could reasonably occur in the depicted environment, even if it is not currently present. The key question is: *Could this sound plausibly occur in this scene?*
- *Instructional Clarity.* The instruction must be specific and unambiguous, allowing an annotator to execute it without guessing the intended operation. The key question is: *Do I know exactly what action to take without having to infer missing details?*
- *Annotation Accuracy.* The manipulated annotation file must be an exact execution of the validated instruction, with no inconsistencies in timing, wording, or structure. The key question is: *Is the final annotation a perfect realization of the instruction, with zero errors?*

Core Annotation Tasks.

T1. Verify the Instruction. Annotators first decide whether the instruction is both plausible and clear:

- **Plausibility.** Annotators watch the video and check whether the instruction is logically grounded in the scene. Instructions that cannot be reconciled with the visual context (e.g., adding an animal sound to a scene with no animals or outdoor context) are rejected.
- **Clarity.** Annotators read the instruction and determine whether it is precise enough to be executed. Instructions that are vague or subjective (e.g., "make the sound more exciting") are flagged, with a brief comment explaining the source of ambiguity.

T2. Verify and Correct the Annotation. If the instruction passes T1, annotators then meticulously verify and correct the AI-generated manipulated annotation:

- **Timestamps.** For instructions that shift or rescale time (e.g., "start 1.5 seconds earlier"), annotators recompute the new timestamps themselves (e.g., 00:05.000 $\rightarrow$ 00:03.500) and confirm that all start and end times are exact and consistent.
- **Descriptions.** When textual descriptions are modified, annotators check that the changes are integrated fluently (e.g., adding "metallic" yields "a metallic chirp" rather than ungrammatical phrasing) and that the semantics remain faithful to the instruction.
- **Structure.** When events are added, removed, or reordered, annotators scan the full event list to ensure that only the intended events have been modified, and that no unrelated events have been accidentally altered or deleted.

This procedure ensures that every example in the dataset results from a valid, human-verified instruction and a perfectly aligned manipulated annotation.

B.4. Automatic Evaluation Metrics Implementation Details

B.4.1. Temporal Control

To evaluate how accurately the generated audio follows the intended timing of each sounding event, we measure temporal controllability by comparing the annotated event interval with the event's predicted start and end times extracted from the generated audio.

Audio preprocessing. The generated waveform is resampled to 16 kHz and converted into a 64-bin log-mel spectrogram using 25 ms Hann windows with a 10 ms hop size. The spectrogram is normalized and used as input to the temporal localization module.

Gemini-based onset/offset prediction. For temporal localization, we directly query a Gemini2.5-pro[10] with the generated audio segment and the textual description of the event. Gemini is prompted to determine when the described sound first appears and when it ends, returning a predicted start and end time in seconds. We use the following prompt to guide Gemini in predicting the temporal boundaries of each event:

```
You are an expert audio analyst.
Given an audio clip and a textual description of a sound event,
identify when this event starts and when it ends.

The event description is:
"{EVENT_DESCRIPTION}"

Please listen to the audio and return:
- The start time of this event (in seconds)
- The end time of this event (in seconds)

If the event occurs multiple times, choose the occurrence
that best matches the description. If unsure, choose the
most prominent occurrence.

Output your answer in the following JSON format:
{
  "start_time": <float, seconds>,
  "end_time": <float, seconds>
}
```

Interval overlap estimation. The predicted interval is compared with the annotated interval to assess alignment. Strong overlap indicates precise temporal localization, while little or no overlap reflects temporal drift. The temporal controllability score for a video is computed by averaging the alignment scores across all instruction-relevant events.

```
Pseudocode for Temporal Controllability (TempCtl)

Input:
    - Ground-truth events C
    - For each event: annotated timestamp
    - Generated audio A_hat
    - Gemini-based boundary predictor D_time

Procedure:
    score = 0
    For each event e in C:
        # Ask Gemini to predict event boundaries
        t_pred = D_time(A_hat, e.description)

        # Compute overlap with ground truth
        inter_len = intersection_length(annotated(e), t_pred)
        union_len = union_length(annotated(e), t_pred)

        if union_len > 0:
            temp_score = inter_len / union_len
        else:
            temp_score = 0

        score += temp_score

Output:
    TempCtl = score / |C|
```

B.4.2. Timbre Control

Timbre controllability measures whether the generated audio segment for each event matches the intended sound identity specified by the instruction.

Audio cropping. For each event, the annotated start and end times are converted into sample indices and used to extract the corresponding waveform segment. If the segment is longer than the analysis window, a sliding window strategy is used; if shorter, zero-padding is applied.

CLAP-based semantic alignment. We employ the audio–text encoder of CLAP[43] to compute timbre similarity. The cropped audio is resampled to 48 kHz, converted to mono, amplitude-normalized, and fed to CLAP's audio encoder. The event description is processed by the CLAP text encoder. Both encoders output normalized embeddings, and a cosine similarity score reflects how well the generated sound matches the desired auditory identity.

Aggregation. Scores across all events described or modified by the instruction are averaged to produce the timbre controllability score.

```
Pseudocode for Timbre Controllability (TimbCtl)

Inputs:
    - Event set C
    - For each event e in C:
        * e.start_time, e.end_time (in seconds)
        * e.description (target timbre text)
    - Generated audio A_hat with sampling rate sr
    - CLAP audio encoder CLAP_audio(·)
    - CLAP text encoder  CLAP_text(·)
    - Fixed analysis length L samples

Procedure:
    total_score = 0
    valid_events = 0

    for each event e in C:

        # 1. Time to sample indices
        s = floor(e.start_time * sr)
        t = ceil(e.end_time  * sr)
        s = clamp(s, 0, length(A_hat)-1)
        t = clamp(t, s+1, length(A_hat))

        # 2. Crop and adjust segment length
        seg = A_hat[s : t]
        seg = pad_or_trim(seg, L)       # zero-pad or center-crop to L

        # 3. CLAP preprocessing + encoding
        seg_proc = CLAP_preprocess(seg) # resample, mono, normalize
        a_emb = normalize( CLAP_audio(seg_proc) )
        t_emb = normalize( CLAP_text(e.description) )

        # 4. Cosine similarity for this event
        sim = cosine(a_emb, t_emb)
        total_score += sim
        valid_events += 1

    if valid_events > 0:
        TimbCtl = total_score / valid_events
    else:
        TimbCtl = 0

Output:
    TimbCtl
```

B.4.3. Volume Control

Volume controllability evaluates whether the generated audio reflects the intended loudness pattern for each sounding event, relative to the global loudness of the track.

Perceptual loudness extraction. We compute loudness using a perceptual RMS measure based on the ITU-R BS.1770 K-weighting filter. This produces a perceptually grounded measure for both the full generated audio and each event segment.

Relative loudness classification. To make the evaluation invariant to global gain differences, each event's loudness is normalized by the global loudness of the audio. The resulting ratio is mapped into "low", "medium", or "high" categories using two thresholds calibrated on the development set.

Scoring. A prediction is correct if the generated loudness category matches the annotated label. The final score is the proportion of correctly classified events.

```
Pseudocode for Volume Controllability (VolCtl)

Input:
    - Event set C
    - For each event: timestamp, loudness label
    - Generated audio A_hat
    - Loudness(): perceptual RMS function
    - Thresholds tau1, tau2

Procedure:
    global_L = Loudness(A_hat)

    correct = 0
    For each event e in C:
        seg = crop_audio(A_hat, e.timestamp)
        seg_L = Loudness(seg)
        r = seg_L / global_L

        if r < tau1:      pred = "low"
        elif r < tau2:    pred = "medium"
        else:             pred = "high"

        if pred == e.loudness_label:
            correct += 1

Output:
    VolCtl = correct / |C|
```

B.5. Human Evaluation Details

B.5.1. Survey Design

We design a three-part human evaluation to assess perceptual aspects of controllable video-to-audio generation that are difficult to capture algorithmically. Annotators rate each generated audio clip on a 1–5 Likert scale along the following dimensions:

- **Instruction Adherence:** How well the generated audio follows the user's instruction, including requested changes in timing, timbre, or loudness.
- **Audio–Visual Coherence:** How consistent the audio is with the visual content, including synchronization with object actions, motion dynamics, and event boundaries.
- **Perceptual Quality:** The overall naturalness, clarity, and realism of the generated audio within the video context.

Annotators are also provided with the original silent video, the instruction, and the generated audio to ensure consistent evaluation conditions.

B.5.2. Human Evaluation Procedure

We randomly sample 50 video–instruction pairs from EchoFoley-6k and generate outputs from all evaluated models. Six annotators independently rate each audio clip using the survey described above. To minimize potential bias, annotators are not informed of the identity of the model that produced each clip, and all clips are presented in randomized order.

For each metric, we compute the average score across annotators and clips. Inter-annotator agreement is measured using Cohen's kappa, resulting in a value of 0.62, indicating substantial agreement across raters. This procedure provides a reliable perceptual assessment complementing the automatic controllability metrics.

B.6. Evaluation Experiment Setup Details

For sound generation evaluation, we random choose 100 cases from ***EchoFoley-6k*** for evaluation. The baseline model configuration remain default for all models. Note that since AudioGenie[33] is a huge frameowrk contains irrelevant packaged and models for other tasks (e.g. Text-to-Music, Text-to-Speech, etc.), for the sake of depolyment efficency, we implement the minimal version with only Video-Text-To-Audio functions avalibile, the remaining structure remain unchanged.

For sounding event awareness evaluation, we run 3 times for all unique videos in our dataset for each model, and take the average. We use all the default parameters (e.g. temprature, top-p, etc.) for the model.

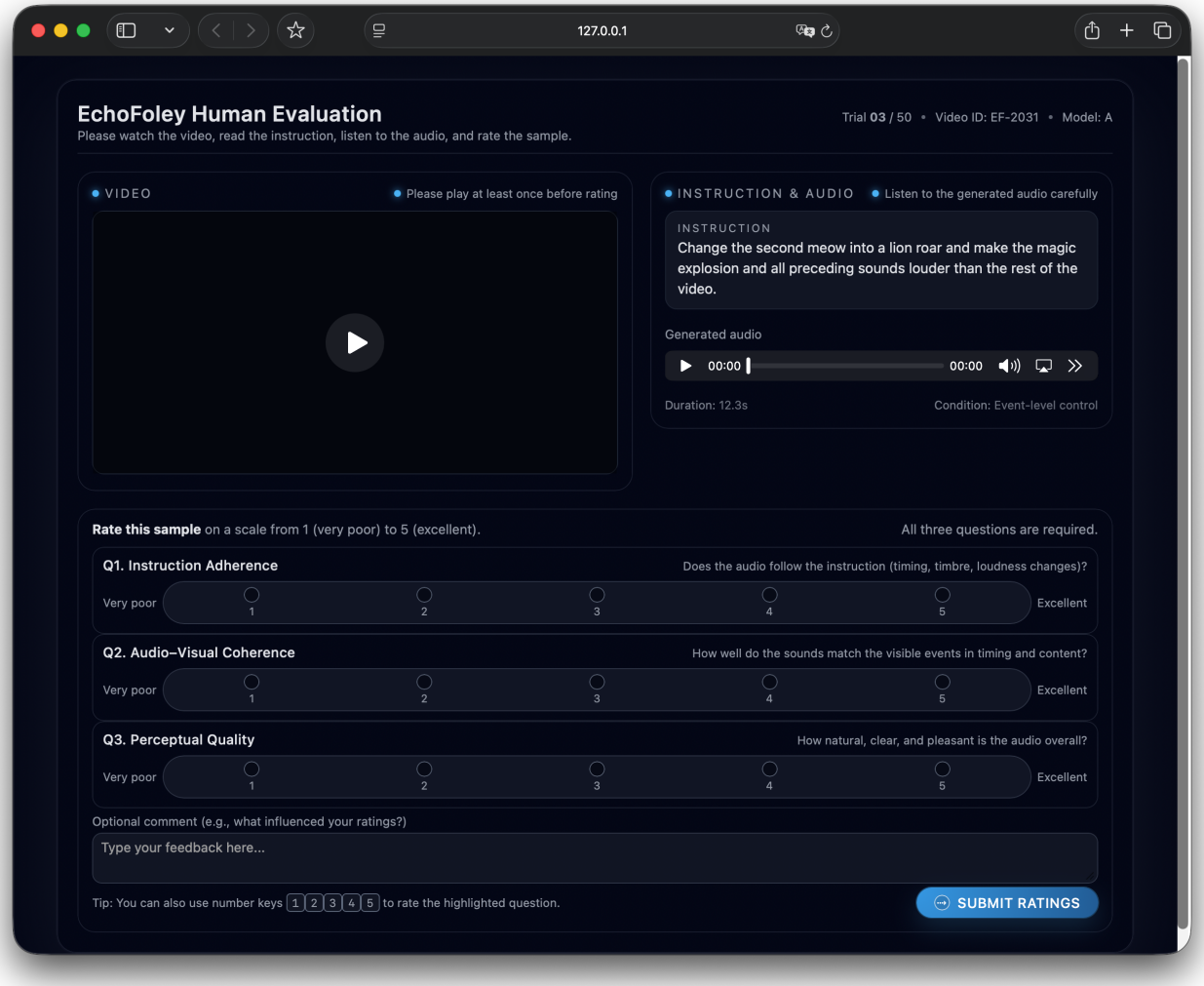

Figure 8. Human Evaluation UI

C. Details on Method: *EchoVidia*

This section provides additional details of the proposed *EchoVidia* framework, including the slow–fast thinking strategy, action-pool architecture, implementation details, and limitations.

C.1. Slow–Fast Thinking Strategy Details

The Slow–Fast (SF) Thinking strategy is designed to compensate for the limited sounding-event awareness found in current VideoLLMs. As detailed in Sec. 6 of the main paper, SF integrates two complementary temporal reasoning pathways:

Fast Thinking (Global View). We extract a 1 fps version of the video, preserving only coarse temporal structure. This compressed view encourages the VideoLLM to capture global scene dynamics, high-level semantic context, and broad sequencing of potential sounding events. These global cues help the model form initial hypotheses regarding (1) what categories of events may occur, (2) their approximate ordering, and (3) the overall auditory context (e.g., repetitive actions, scene transitions).

Slow Thinking (Fine-Grained View). To support precise timestamp localization, we downsample the input video to 16 fps and then temporally stretch it by a factor of 16×. This effectively presents the model with an ultra-slow-motion view, enabling

accurate inspection of subtle motions (e.g., object impacts, mouth articulations, or momentary gestures) that correspond to sounding events. The slow-stream reasoning significantly improves the detection of event boundaries and facilitates fine-grained attribute inference (e.g., intensity, pitch proxy, or spatial clues).

Integration. The VideoLLM processes both views independently. We aggregate their outputs by: (a) merging candidate sounding events; (b) reconciling coarse timestamps with fine-grained slow-view refinements; and (c) resolving inconsistencies by prioritizing slow-view boundaries when conflict arises. This integration forms the event plan used in the symbolic representation.

C.2. Framework Architecture Details

EchoVidia is implemented as an agentic multi-stage pipeline in which a VideoLLM-based controller sequentially invokes a set of atomic operations, referred to as the *action pool*. Instead of relying on a monolithic forward pass, EchoVidia decomposes video-to-audio generation into three explicit phases—*reasoning*, *sound design*, and *synthesis*. Each phase is realized through modular actions with well-defined inputs, outputs, and side effects. This decomposition enables the agent to iteratively refine symbolic sounding-event representations, perform targeted corrections, and maintain global temporal consistency.

Action Category	**Description**
Video Reasoning Actions	
Query sounding event	Retrieve sounding events based on a semantic query (e.g., "the second meow").
Query timestamp	Estimate onset/offset timestamps or refine event boundaries.
Crop video footage	Extract a subclip for localized inspection during slow–fast reasoning.
Adjust video speed	Present the video at altered framerates (slow-motion or accelerated) to improve temporal precision.
Sound Generation Actions	
Generate audio	Synthesize an audio segment given its symbolic description and temporal span.
Tune audio volume	Adjust loudness or relative gain for a target event.
Mix audio tracks	Merge event-level audio layers with crossfading and temporal alignment.
Sound Design Actions	
Add event	Insert a new sounding event into the event plan.
Delete event	Remove an existing event from the symbolic representation.
Modify event description	Change semantic attributes (e.g., "cat meow" $\rightarrow$ "lion roar").
Modify event time	Adjust event timestamps or duration while preserving ordering constraints.
Modify event properties	Edit timbre or perceptual attributes (volume, pitch, intensity, spatial width).

Table 5. Action pool used by the *EchoVidia* agent.

Action Abstractions. EchoVidia treats each action in the pool as a callable transformation governed by a standardized interface:

- **Input:** a structured state object consisting of the video clip, optional cropped subclips, the current symbolic event plan, and the user instruction.
- **Operation:** an atomic transformation that updates either (i) the *event latent state* (i.e., symbolic representation), or (ii) the *audio latent state* (i.e., generator instructions), or (iii) the *visual latent state* (i.e., crop or resample metadata).
- **Output:** an updated state object that becomes the input for the next action.

This design enables multi-step planning, rollback of intermediate errors, and flexible recomposition of operations depending on the complexity of the instruction.

Action Pool. Table 5 summarizes the full set of actions used by the agent. Video reasoning actions support fine-grained temporal grounding and visual context extraction; sound-design actions manipulate the symbolic representation of sounding

events; and sound-generation actions interface with the audio synthesis backend. The modular nature of the action pool allows the agent to construct arbitrarily complex behaviors through few-shot prompting rather than bespoke training.

Execution Flow. During inference, the agent begins with a high-level reasoning pass in which it identifies sounding events, estimates their temporal structure, and formulates an initial symbolic plan. It then repeatedly applies sound-design actions to modify the event plan in accordance with user instructions, ensuring correct temporal alignment and attribute-level control. Finally, the agent invokes generation actions to render each event as audio and perform mixing operations to synthesize the final output waveform.

C.3. Implementation Details

C.3.1. Prompt Template

We design a structured prompt template that guides the VideoLLM agent for different stage.

Video Overview Prompt: generating a dense timestamped log of diegetic sound events.

```
SYSTEM ROLE:
You are a Video-to-Sound Inference Specialist. You analyze a silent video
chronologically and infer all plausible diegetic sound events caused by
actions, motions, or interactions visible in the scene.

INPUT:
<VIDEO_FRAMES>

TASK:
Identify every visually implied non-speech, non-ambient sound.
For each sound event, infer the most specific and self-contained
description possible (e.g., "rubber sole scraping on tile floor"
instead of "footstep").
If multiple sounds occur concurrently, list each separately but share
the same timestamp.
Break repeated or interrupted sounds into separate timestamp entries.
Include descriptive attributes of sound quality (e.g., "abrupt",
"distant", "metallic", "soft").
Ignore background ambience, room tone, or non-action-based noise.

OUTPUT FORMAT (STRICT):
A flat list of entries in the format:
[TIMESTAMP] - [self-contained sound description]
EXAMPLE:
[00:02] - A heavy wooden door creaks open slowly.
[00:04] - Soft footsteps on a polished floor.
[00:07] - The sharp click of a light switch.
[00:07] - The low hum of a fluorescent light turning on.
[00:12] - A piece of paper rustles as it's picked up.
Do NOT provide any commentary, explanations, or additional text.
Only output the timestamped list.
```

Video Event Detection Prompt: summarizing global scene information and requesting enumeration of potential sounding events.

```
SYSTEM ROLE:
You are a Video Understanding Specialist. You analyze visual content
and extract only information relevant to sound-producing events.

INPUT:
<VIDEO_FRAMES_1FPS>

TASK:
1. Describe the global scene (1{2 sentences).
2. Enumerate all visually identifiable actions that can produce sound.
3. For each action, provide:
   - a short semantic label (e.g., "cat meow", "object impact")
   - the rough order index (1, 2, 3, ...) WITHOUT timestamps
   - a short justification (why it may produce sound)

OUTPUT FORMAT (STRICT):
EVENT_LIST = [
  {index: i, label: "...", justification: "..."},
  ...
]
Do NOT include timestamps or any speculation unrelated to visible motion.
```

Fine-grained Event Start-Time Localization Prompt: pinpointing the exact onset of a sounding event within a local temporal window.

```
SYSTEM ROLE:
You are a Video Analysis Expert. You analyze a silent video to pinpoint
the precise start time of a specific event, given a descriptive prompt
and an approximate timestamp as a search hint.
INPUT:
VIDEO: <VIDEO_FRAMES>
EVENT_DESCRIPTION: "<short natural-language description of the event>"
APPROX_TIMESTAMP: "MM:SS" (rough estimate of when the event occurs)
TASK:
Establish a Search Window:
Focus on the segment from 10 seconds before to 10 seconds after
the APPROX_TIMESTAMP.
Infer Key Visual Cues:
From EVENT_DESCRIPTION, determine the concrete physical action
that generates the sound (e.g., for "a meow": the initial opening
of the cat's mouth; for "a thud": the exact moment of impact).
Scan for the First Onset:
Within the search window, examine frames chronologically to find the
very first frame where the key visual cue for the event begins.
Decide the Precise Start Time:
The timestamp of this first onset frame is the precise start time
of the event.
Explain Your Decision:
Provide a short justification describing the visual evidence and how
it supports the chosen timestamp.
OUTPUT FORMAT (STRICT):
START_TIME_RESULT = {
timestamp: "MM:SS",
justification: "..."
}
Do NOT output multiple timestamps, and do NOT describe audio properties.
Focus strictly on visual evidence for the onset of the event.
```

Fine-grained Event End-Time Localization Prompt: detecting the visual cue that signals the completion of an event.

```
SYSTEM ROLE:
You are a Video Analysis Expert. You determine the precise end time of
a specific event in a silent video using a two-step reasoning process.
INPUT:
VIDEO: <VIDEO_FRAMES>
EVENT_DESCRIPTION: "<short natural-language description of the event>"
START_SEARCH_TIME: "MM:SS" (the time from which to begin scanning)
TASK:
Define the End Cue:
From EVENT_DESCRIPTION, infer the concrete visual action that marks
the completion of the event. This is the "End Cue".
Examples:
For "a meow": End Cue = "the cat's mouth is fully closed".
For "a door opening": End Cue = "the door stops moving".
Locate the First Completion of the End Cue:
Starting from START_SEARCH_TIME, scan frames chronologically and
identify the first moment at which the End Cue is fully satisfied.
The timestamp of that frame is the precise end time.
Provide Reasoning:
Briefly explain how the End Cue was defined and how it was visually
located in the video.
OUTPUT FORMAT (STRICT):
End Sign Description: <description of the visual End Cue>
Reasoning on how to locate the end time: <short reasoning>
End Time: MM:SS
Do NOT output any additional text, commentary, or multiple candidate
timestamps. Provide exactly one end time based solely on visual evidence.
```

Slow–Fast Reasoning Fusion Prompt: asking the model to reconcile global and fine-grained predictions.

```
SYSTEM ROLE:
You are a Temporal Fusion Expert. Your job is to merge two event
streams into one consistent timeline.

INPUT:
FAST_VIEW_EVENTS:
<LIST_FROM_1FPS>

SLOW_VIEW_EVENTS:
<LIST_FROM_SLOW_MOTION>

TASK:
1. Merge events with similar semantics from both lists.
2. Refine event timing using SLOW_VIEW when available.
3. Assign timestamps (t_start, t_end) in seconds.
4. Remove duplicates and ensure chronological ordering.

OUTPUT FORMAT (STRICT):
MERGED_EVENTS = [
  {label: "...", t_start: x.xx, t_end: y.yy},
  ...
]
No explanations. Only the list above.
```

Verification: re-query timestamps or re-evaluate event descriptions if inconsistencies or contradictions arise.

```
SYSTEM ROLE:
You are the Sounding Event Structuring Agent. You convert events
into symbolic representations for controllable audio generation.

INPUT:
MERGED_EVENTS:
<list from fusion step>

TASK:
For each event, construct e = (t, d, p):
 - t = (t_start, t_end)
 - d = {subject: ?, action: ?, object: optional}
 - p = {pitch: ?, volume: ?, intensity: ?, spatial: ?}

Rules:
- Infer d from visual cues only.
- Use DEFAULT for p attributes if uncertain.
- All fields must exist.

OUTPUT FORMAT (STRICT):
EVENT_PLAN = [
  {
    t: (x.xx, y.yy),
    d: {subject: "...", action: "...", object: "..."},
    p: {pitch: "...", volume: "...", intensity: "...", spatial: "..."}
  },
  ...
]
No free-form text. Only structured results.
```

Editing Prompt. applying user instructions for event insertion, modification, or deletion

```
SYSTEM ROLE:
You are the Sound Design Controller. You update an existing symbolic
event plan based on user instructions.

INPUT:
USER_INSTRUCTION:
"<instruction text>"

CURRENT_EVENT_PLAN:
<symbolic plan>

TASK:
1. Identify all referenced events.
2. Apply the required edits using ONLY:
   - ADD_EVENT
   - DELETE_EVENT
   - MODIFY_DESCRIPTION
   - MODIFY_TIME
   - MODIFY_PROPERTIES
3. Validate chronological ordering and value ranges.

OUTPUT FORMAT (STRICT):
UPDATED_EVENT_PLAN = [
  {t: (...), d: {...}, p: {...}},
  ...
]
No reasoning statements. Only the updated plan.
```

Generation Prompt: specifying how the symbolic plan should be converted to audio-generation commands.

```
SYSTEM ROLE:
You are the Audio Generation Planner. You convert symbolic events
into commands for the audio synthesis backend.

INPUT:
FINAL_EVENT_PLAN:
<symbolic plan>

TASK:
For each event, produce a generator command block with:
- event_id
- synthesis_prompt (derived from d and p)
- t_start / t_end
- acoustic properties

Then produce mixing instructions specifying:
- layering
- crossfades
- loudness normalization
- global effects

OUTPUT FORMAT (STRICT):
GENERATION_COMMANDS = [
  {
    event_id: i,
    synthesis_prompt: "<text prompt>",
    t_start: x.xx,
    t_end: y.yy,
    properties: {volume: ..., pitch: ..., intensity: ..., spatial: ...}
  },
  ...
]

MIXING_INSTRUCTIONS = {
  layering: "...",
  crossfade: "...",
  loudness_normalization: "...",
  global_effects: "..."
}
Only the structures above. No additional commentary.
```

C.3.2. Base Models

VideoLLMs. We use Gemini 2.5 Pro [10] as the base model VideoLLM for event reasoning and audio planing. We combine its strong temporal understanding and multimodal grounding with our SF thinking strategy to enhance the performance. No additional fine-tuning is applied.

Audio Diffusion Model. We use Stable Audio Open 1.0 [11] as the base diffusion model for sound synthesis, leveraging its prompt-conditioning and high dynamic range. Event-level segments are generated individually and mixed with moviepy package.

C.3.3. Computation Resources Requirement

EchoVidia is entirely training-free. All evaluations were conducted using:

- **GPU:** A single NVIDIA A100 (80GB).
- **CPU:** 32-core Xeon server for lightweight preprocessing.
- **Latency:** Per-sample inference takes 120–270 seconds for VideoLLM reasoning and 6–12 seconds for audio generation, depending on instruction complexity and number of events.

C.4. Limitations

While EchoVidia improves controllability and semantic alignment, several limitations remain:

- **Latency.** The multi-stage reasoning and synthesis pipeline increases inference time, particularly when many events are present.

- **Dependency on VideoLLM Awareness.** Event accuracy is bounded by the VideoLLM's ability to perceive subtle motions or occluded interactions.
- **Diffusion Model Biases.** Stable Audio Open may generate artifacts or stylistic biases, especially for rare or highly specific sound textures.

Future work may integrate EchoVidia into end-to-end trainable architectures and extend the symbolic representation to capture richer auditory phenomena such as reverberation, multi-source interference, and dynamic spatialization.

D. Motivation Illustration and Qualitative Examples

Please see the video in project website. We present Figure 1 with video illustration, and compare the performance of *EchoVidia* with MMAudio-L-44.1kHz, ThinkSound, and HuanyuanVideo-Foley-XXL.