Isaac Audio Sensors v1.0.0: A Microphone-Array Sensor Frame and Isaac Sim/Lab Integration Package

Patrizio Acquadro

May 26, 2026

Abstract

This technical report documents the final v1.0.0 state of `isaac-audio-sensors`, a reusable Python package for representing and emitting microphone-array observations in robotics simulation workflows. The package defines a stable `AudioSensorFrame` v1 data contract, deterministic geometry and time-difference-of-arrival backends, an optional room-acoustics backend, lazy Isaac Sim stage integration, lazy Isaac Lab sensor integration, package-native JSON/JSONL trace export, and a reference Omniverse Kit extension. The principal contribution is not a complete acoustic physics engine. It is a bounded sensor-frame and integration package that converts simulated sound-source and microphone-array state into explicit bearing, delay, RMS, ambiguity, pose, unit, provenance, and diagnostic records. The report defines the supported v1 scope, distinguishes stable, supported optional, provisional, experimental, and future surfaces, describes the implemented model, summarizes validation evidence, states limitations, and identifies downstream work required for broader acoustic realism, calibration, adapters, and automated live-runtime validation.

Keywords: robotics simulation; Isaac Sim; Isaac Lab; microphone arrays; TDOA; direction of arrival; audio sensor frames; Omniverse; synthetic data.

1 Introduction

Robotics simulators commonly provide structured observations for cameras, depth sensors, lidar, contact forces, inertial sensors, ray casts, and robot state. Audio is less frequently exposed as a compact robotics-style observation contract, especially for microphone-array tasks that require source identity, source pose, array pose, bearing, per-microphone diagnostics, ambiguity handling, and traceable provenance. The `isaac-audio-sensors` package addresses this gap for Isaac-oriented workflows by defining a stable sensor frame and by providing integration paths for Isaac Sim, Isaac Lab, and Omniverse Kit.

The package version documented here is `1.0.0`. The frame schema version is intentionally separate and remains `ias.audio_sensor_frame.v1` for compatible v1 frame records. This separation matters because Python package releases may add compatible optional fields or diagnostics while the serialized v1 frame contract remains stable for consumers. In v1, renaming required public fields, removing required public fields, changing units, changing coordinate semantics, changing provenance values, changing stable backend identifiers, or changing the frame schema version would be breaking changes.

The purpose of the project is to make audio events usable as auditable simulation observations. A frame records what the simulator knew about the scene and array at a given time window, what backend produced the observation, which detections were emitted, which units and coordinate convention were used, and which limitations or ambiguity conditions apply. This design allows

downstream systems to build reinforcement-learning observations, dataset writers, visualization panels, ROS 2 adapters, or task-specific logic without requiring those adapters to be part of the v1 core.

The v1 release is scoped as a package release. It does not depend on downstream validation in SquadBot, Alex, ROS 2, or any specific robot project. It also does not claim real hardware calibration, sim-real transfer, complete material or occlusion acoustics, production beamforming, speech recognition, or complete L3/L4 acoustic fidelity. The package is useful precisely because this boundary is explicit: it provides a stable and testable interface now, while leaving higher-fidelity acoustics and calibration as future layers.

The report is organized as follows. Section 2 positions the package relative to Isaac sensor infrastructure, Isaac Lab sensors, acoustic simulation tools, and audio-visual embodied AI work. Section 4 describes the implemented model and integration design. Section 5 summarizes the validation evidence. Section 6 states limitations. Section 7 discusses future work. Appendix A contains the release-status table and installation commands that are useful for reproducibility but are not central to the paper body.

2 Related Work

Isaac Sim provides sensor simulation extensions for ground-truth perception and physics-based sensors, including camera, depth, RTX lidar/radar, contact, effort, IMU, proximity, tactile, and related non-visual sensor categories [1]. This infrastructure is broad and production-oriented, but the public sensor documentation is organized around visual, ray-based, and physics-based modalities rather than a reusable microphone-array audio frame contract. `isaac-audio-sensors` therefore complements Isaac Sim: it reads USD stage state and authored metadata, but it emits a package-defined audio sensor record rather than replacing existing Isaac sensor systems.

Isaac Lab provides a sensor abstraction centered on `SensorBase` and `SensorBaseCfg`. The base sensor class uses lazy evaluation, updates data at a configured update period, exposes data through a property, and supports reset by environment identifiers [2]. Existing documented Isaac Lab sensors include camera, contact, frame transformer, ray-cast, and IMU surfaces. `isaac-audio-sensors` uses this pattern for audio by exposing an audio array sensor with fixed-shape buffers suitable for vectorized reinforcement-learning tasks. The package also remains import-safe outside an initialized Lab runtime and provides a recovery API so live Lab users can resolve real `SensorBaseCfg`/`SensorBase` subclasses after AppLauncher initialization.

Room-acoustics software such as `pyroomacoustics` provides a strong foundation for acoustic simulation and array processing. Scheibler, Bezzam, and Dokmanic describe `pyroomacoustics` as a Python package for constructing room simulation scenarios with sound sources and microphones, generating room impulse responses through the image source model, and implementing array-processing algorithms [3]. `isaac-audio-sensors` uses this ecosystem conservatively. The v1 `room_acoustics` backend is optional, dependency-gated, and treated as an approximate L2 shoebox-room path. The package does not present `pyroomacoustics` as proof of sim-real calibration or complete material/occlusion realism.

Audio-visual embodied AI work demonstrates the value of simulated acoustic observations. SoundSpaces instrumented Habitat with audio renderings for Matterport3D and Replica environments and enabled arbitrary sound sources for audio-visual navigation research [4]. That line of work is important but has a different target. SoundSpaces focuses on embodied navigation datasets and Habitat instrumentation. `isaac-audio-sensors` focuses on Isaac Sim/Lab integration, a stable frame contract, deterministic microphone-array diagnostics, and source/array metadata authoring

in USD/Kit workflows.

Omniverse Replicator provides writer and annotator infrastructure for synthetic-data workflows; its documentation illustrates access through `WriterRegistry` and annotator registry APIs [5]. The v1 package uses Replicator only as an optional extension capability in the reference Kit UX. The stable recording path for the package remains JSON/JSONL `AudioSensorFrame` output. This distinction keeps the pure Python core independent from Kit and avoids making Replicator availability a release requirement.

The resulting gap is narrow but practical: existing systems provide major simulation frameworks, acoustic libraries, or embodied-audio datasets, while `isaac-audio-sensors` provides a small, reusable, Isaac-oriented audio sensor frame and integration layer. Its contribution is a stable interface and validation boundary rather than a new acoustic solver.

3 Scope and Requirements

The supported v1 surface consists of the stable `AudioSensorFrame` v1 contract, the stable `geometry_only` and `tdoa_synthetic` backends, the supported optional `room_acoustics` backend, supported Isaac Sim and Isaac Lab sensor paths, package-native JSON/JSONL export, and a reference Omniverse extension. Replicator recording is supported only as optional extension functionality inside compatible Kit runtimes.

The pure core must remain importable in a normal Python environment without Isaac Sim, Isaac Lab, Omniverse, `pxr`, `omni`, `pyroomacoustics`, ROS 2, CUDA, or torch. This requirement is architectural rather than cosmetic: downstream users should be able to validate configs, serialize traces, run core backends, and inspect schema files without an NVIDIA runtime. Isaac-specific behavior is therefore loaded lazily, and runtime errors are raised only when the user invokes a function that needs the corresponding optional runtime.

The release also requires clear non-promises. V1 does not include real hardware benchmarks, sim-real calibration, complete L3/L4 acoustic fidelity, realistic occlusion/material acoustics, mandatory ROS 2 adapters, or validation of a specific downstream robot system as a package release gate. These items may be downstream or future work, but they do not define v1 completeness.

4 Model and Implementation

4.1 Architecture

The implementation is organized into four layers, shown in Figure 1. The first layer is the pure Python core. It defines scene snapshots, source specifications, microphone-array specifications, time windows, detections, DOA estimates, sensor frames, backend selection, trace I/O, schema export, and CLI entry points. This layer is deliberately small and dependency-light.

The second layer contains Isaac Sim and Omniverse helpers. It authors and reads ordinary USD custom metadata under the `ias:*` namespace, resolves world poses from USD transform stacks when USD APIs are available, supports fallback attributes for duck-typed tests, discovers arrays and sources semantically, and runs a lifecycle-oriented live sensor object. The third layer contains Isaac Lab integration. It exposes an audio array sensor surface with fixed-shape tensors, supports explicit scene snapshots, stage binding, and entity tensor binding, and follows Lab's lazy-update model. The fourth layer is intentionally external: downstream adapters may convert frames into project-specific observation dictionaries, ROS 2 messages, datasets, or other graph contracts, but those adapters are not required by v1.

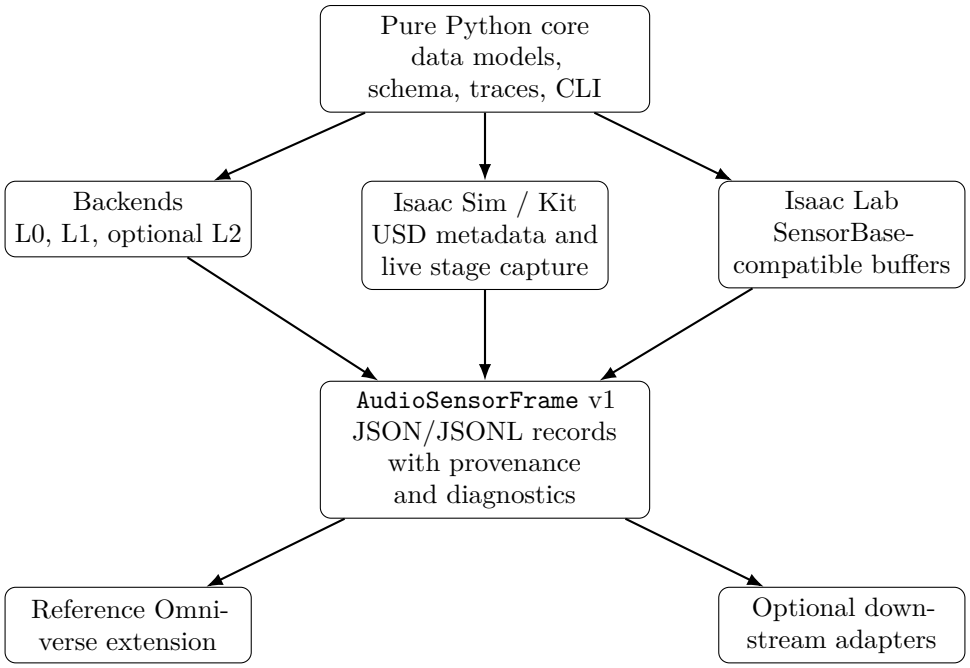

Figure 1: Package architecture. The core frame contract is the common boundary between deterministic backends, Isaac Sim stage capture, Isaac Lab tensorization, the reference extension, and optional downstream adapters.

This layered design is the main mechanism for controlling scope. The core can be tested and distributed independently, while Isaac integrations remain available to users who supply a compatible runtime. It also prevents optional downstream systems from becoming hidden release gates.

4.2 Public Data Contract

The `AudioSensorFrame` is the primary public artifact. A frame represents one array observation over one time window. Its top-level fields include schema version, frame identifier, frame name, timestamp, start and end time, sample rate, frame index, backend identifier, array identifier, array pose, coordinate convention, units, provenance, maximum event count, detections, aggregate per-microphone RMS, waveform paths, and diagnostics. The schema version is fixed at `ias.audio_sensor_frame.v1` for compatible v1 records.

Each detection records a detection identifier, source identifier, class label, detection mode, timestamp, ground-truth bearing when available, source distance, DOA estimate, source pose, per-microphone delay, per-microphone RMS, audio asset path, and diagnostics. A DOA estimate records the estimated bearing when unique, candidate bearings when ambiguity exists, an eight-way sector label, confidence, ambiguity class, and ambiguity reason. A pose records position in meters, quaternion orientation in `xyzw` order, frame name, and coordinate convention.

Table 1 summarizes the major contract groups rather than listing every field individually. The complete schema is exported by the package and checked into the repository as `docs/schemas/audio_sensor_frame.v1.schema.json`.

The coordinate convention is local $+X$ forward, local $+Y$ right, and local $+Z$ up, with bearing measured clockwise from array forward. Bearing sectors use eight half-open bins centered on straight, straight-right, right, behind-right, behind, behind-left, left, and straight-left. The wraparound straight sector includes bearings from 337.5° to 360° and from 0° to below 22.5°. This convention is

Table 1: Major components of the `AudioSensorFrame` v1 contract.

Component	Contract meaning
Frame identity	Stable schema version, frame id/name, backend id, array id, frame index, and deterministic maximum-event behavior.
Time and sampling	Millisecond timestamp, second-based start/end window, and sample rate in hertz.
Geometry	Array pose and source poses in meters, quaternion `xyzw` orientation, and the `x_forward_y_right_z_up_clockwise_bearing` coordinate convention.
Detection state	Source identity, class label, detection mode, distance, bearing, sector, confidence, per-microphone delay, per-microphone RMS, and source metadata.
Ambiguity	Explicit DOA ambiguity fields and candidate bearings, especially for two-microphone front/back ambiguity.
Provenance and diagnostics	Stable provenance values and open-ended diagnostics dictionaries for stage capture, Lab binding, backend evidence, and optional runtime details.

tested because a change in sector semantics would break downstream consumers.

The units dictionary is part of the contract. Positions and distances are meters, time windows are seconds, timestamps are milliseconds, sample rate is hertz, RMS is linear, gain is decibels, and orientation is quaternion `xyzw`. Provenance values are likewise bounded in v1: `synthetic/core`, `room_acoustics`, `isaac_live`, and `replay/trace`. Diagnostics remain open-ended so that new evidence can be added without changing required fields.

The `max_events` field is deterministic. Producers must not emit more detections than the configured limit, and truncation order must be deterministic for the backend and input scene. This makes fixed-shape Lab tensors and downstream trace comparison reproducible.

4.3 Acoustic Backends and Fidelity Levels

The backend design is expressed as a fidelity ladder rather than a single realism claim. Table 2 gives the v1 interpretation. L0 and L1 are stable runtime levels. L2 is supported but optional because it depends on the `room` extra and `pyroomacoustics`. L3 and L4 are documented as future-compatible directions rather than selectable v1 runtime backends.

The `geometry_only` backend uses scene geometry to produce deterministic bearing and distance records. It is useful for stage debugging, trace validation, and tasks that need idealized directional events. The `tdoa_synthetic` backend derives direct-path delays across microphone positions and records per-microphone diagnostics. It includes deterministic stress controls such as delay noise, clock jitter, and gain mismatch, but those controls are not calibrated hardware noise models.

The `room_acoustics` backend is optional. When available, it creates approximate room impulse responses and microphone waveforms through `pyroomacoustics`, derives TDOA through GCC-PHAT, and records diagnostics such as room configuration, RIR length, RIR peak delay, waveform sample count, direct-path comparison, per-microphone RMS, and estimated TDOA matrices. If `pyroomacoustics` is missing, L2 use fails or skips with a clear optional-dependency message; L0, L1, core import, schema export, and trace export remain unaffected.

Table 2: Acoustic fidelity levels in the v1 package boundary.

Level	Backend or family	v1 meaning
L0	`geometry_only`	Stable deterministic bearing, distance, sector, and maximum-event behavior. It is geometry, not acoustic propagation.
L1	`tdoa_synthetic`	Stable direct-path synthetic delay and RMS diagnostics, with explicit ambiguity reporting for two-microphone arrays. It does not model reverberation or occlusion.
L2	`room_acoustics`	Supported optional shoebox-room path using `pyroomacoustics` when installed. It provides RIR/waveform/GCC-PHAT diagnostics but is not a calibrated physical acoustic model.
L3	`advanced_realism`	Provisional future family for richer material, occlusion, directivity, noise, waveform, and estimator behavior. Not a complete v1 runtime backend.
L4	`sim_real_calibration`	Experimental future tooling family for measured array pose, gain, time offset, noise, calibration artifacts, and sim-vs-real comparison. Not a v1 calibration pipeline.

4.4 Isaac Sim Integration

The Isaac Sim layer provides live stage capture without making Isaac a core dependency. It reads authored metadata and native USD-like sound attributes, resolves poses, constructs scene snapshots, runs a selected backend, emits `AudioSensorFrame` records, and optionally writes package-native JSONL traces. The main lifecycle is construction from an explicit stage path or discovered stage, `start()`, repeated `update()` or `capture()`, `get_latest_frame()`, `stop()`, and `close()`.

USD metadata is intentionally ordinary custom metadata, not a custom USD schema. Arrays use metadata such as `ias:array_id`, layout name, sample rate, coordinate convention, microphone ids, and microphone offsets. Source prims use metadata such as `ias:source_id`, class label, audio asset path, start time, duration, gain, and directivity. Microphone child prims may provide explicit array-local offsets. Semantic discovery combines this metadata with type/name signals, native sound attributes, child microphone prims, configured roots, include/exclude filters, and preferred entity selection.

Pose resolution uses USD world-transform APIs when available and fallback attributes in tests or lightweight stages. Nested transform stacks are supported for robot/base-mounted arrays, moving source parents, moving array prims, and microphone child prims. Time-coded USD reads allow live motion to be reflected in successive frames. Active source windows are half-open time intervals, so a source is active only when the requested frame window overlaps its authored active interval.

The live Isaac Sim smoke validates this layer by creating an in-memory USD stage, authoring robot/base, source, array, and microphone-child transform stacks, discovering the stage semantically, running `geometry_only` and `tdoa_synthetic`, verifying changed frame output after movement, verifying inactive-window behavior, producing debug primitives, and writing JSON evidence plus JSONL frames.

4.5 Reference Omniverse Extension

The repository includes a reference Kit extension under `exts/isaac_audio_sensors.omni`. The extension manifest identifies it as *Isaac Audio Sensors* version `1.0.0`. It is a project extension, not an official NVIDIA extension. Its purpose is to provide a practical UX for authoring array/source metadata, binding a live stage, selecting a backend, starting and stopping the sensor, inspecting the latest frame, visualizing structured debug primitives, exporting JSON/JSONL traces, exporting reusable binding/config summaries, and optionally recording through Replicator.

The extension entrypoint is import-safe in ordinary Python. It does not require `omni`, `pxr`, Isaac Sim, a display, CUDA, or a GPU merely to import and instantiate the extension object. Live UI construction and live stage operations are attempted only inside a compatible Kit runtime. The controller records readable errors for missing stage, missing selection, invalid prim path, invalid backend, invalid Replicator output, unavailable Replicator runtime, and other recoverable UX failures.

Package-native JSON/JSONL is the stable recording path. Replicator support is optional and extension-only. When `omni.replicator.core` exposes compatible writer APIs, the extension registers a writer, writes recoverable payloads that contain the full `AudioSensorFrame` object, flushes output, and records status. If a Kit version lacks a compatible writer registry, writer lookup, annotator API, or flush behavior, the extension reports that status while leaving the package JSON/JSONL path available.

4.6 Isaac Lab Integration

The Isaac Lab layer exposes audio observations in a Lab-compatible form. When imported after Isaac Lab and Kit initialization, `AudioArraySensorCfg` resolves as a real `SensorBaseCfg` subclass and `AudioArraySensor` resolves as a real `SensorBase` subclass. Outside an initialized Lab runtime, fallback classes remain import-safe and exercise the same conversion logic without pretending to be real Lab bases. Live users can call `ensure_isaac_lab_sensor_classes()` after AppLauncher initialization to recover proven real classes if fallback classes were imported too early.

The observation surface is fixed-shape for vectorized environments. The sensor data object exposes `event_presence` as a boolean tensor of shape $[N, E]$, `bearing_deg` and `confidence` as float tensors of shape $[N, E]$, `sector_onehot` as $[N, E, 8]$, `per_mic_rms` as $[N, E, M]$, and `ambiguity_mask` as a boolean tensor of shape $[N, E]$. Here N is the number of environments, E is `max_events`, and M is the microphone count. The event-presence mask is the authoritative distinction between real detections and padding.

The binding model supports explicit scene snapshots, provider callbacks, cloned USD stage binding, and scene/entity tensor binding. Stage binding can use explicit relative paths or semantic discovery within clone namespaces. Entity binding is duck typed and supports common Isaac Lab scene and entity patterns, including root/body pose tensors such as `root_state_w`, `root_pos_w`/`root_quat_w`, `body_state_w`, and `body_pos_w`/`body_quat_w`. Entity state is treated as world-frame by default; environment-origin addition is opt-in for env-local state.

Selected-environment update and reset are part of the sensor contract. Calling `update(..., env_ids=[...])` recomputes only the selected rows, and `reset(env_ids=[...])` clears selected rows while preserving other environment rows. This behavior is essential for RL pipelines that update subsets of vectorized environments.

5 Validation and Evidence

Validation is organized around the package boundary. The core release gates test the stable schema, dataclass validation, trace round trips, deterministic L0/L1 behavior, optional L2 behavior, import safety, lint, build, distribution audit, and schema export. Live Isaac checks validate the Isaac Sim, Isaac Lab, and Kit extension paths when a user-managed runtime is available. Missing optional runtimes are not treated as hidden core failures; they must either pass or produce explicit blocker evidence.

Table 3 reports the objective live evidence recorded for the final v1 documentation state. These are machine-local evidence facts, not guarantees that every user's workstation has the same Isaac, Kit, CUDA, or driver configuration.

Table 3: Summary of v1 validation evidence.

Validation area	Status	Evidence summary
Core package checks	Passed in release workflow	Import smoke, pytest, lint, build, distribution audit, schema export, config validation, JSON/JSONL trace checks, and whitespace checks are the documented release gates.
Isaac Sim live smoke	Passed locally	Isaac Sim app `5.1.0`, Kit build `107.3.3+production.229672.69cbf6ad.gl`, Torch `2.7.0+cu128`, CUDA-visible NVIDIA GeForce RTX 4090. Six `AudioSensorFrame` JSONL records were produced: three `geometry_only` and three `tdoa_synthetic`.
Optional room acoustics in live Sim	Skipped in that runtime	`room_acoustics` was skipped because `pyroomacoustics` was not installed in the Isaac Python runtime. This is expected optional-dependency behavior.
Omniverse extension UX	Passed locally	The extension manager enabled `isaac_audio_sensors.omni-1.0.0`; the workflow authored array/source metadata, discovered entities, ran `tdoa_synthetic`, wrote one valid frame, produced seven overlay primitives, exported config/latest-frame files, and exercised Replicator writer start/write/flush/stop.
Isaac Lab GPU smoke	Passed locally	Isaac Lab `0.54.2`, Isaac Sim `5.1.0`, Kit build `107.3.3+production.229672.69cbf6ad.gl`, Torch `2.7.0+cu128`, CUDA device `cuda:0`, RTX 4090. Two environments, two maximum events, four microphones, and all audio/bookkeeping tensors on CUDA.
External consumer smoke	Passed for release candidate	The wheel was installed into an independent temporary consumer workspace with `pip -target`, then generic Isaac Sim and Isaac Lab consumer scripts imported the package without repository-source path leakage or downstream project imports.

The core validation suite checks that the generated JSON Schema exactly matches the checked-in schema, that required frame and detection fields match dataclass serialization, that additive optional fields remain allowed, that trace corpus files round-trip, and that units, provenance, timestamps, coordinate convention, stable backend identifiers, and bearing sectors remain fixed. Backend tests cover canonical geometry sectors, TDOA clean azimuth cases, explicit two-microphone front/back ambiguity, deterministic replay, GCC-PHAT delay signs, optional `room_acoustics` diagnostics through a fake `pyroomacoustics` path, optional dependency errors, and malformed signal rejection.

Import-safety tests run the public imports while blocking `pxr`, `omni`, `isaacsim`, `isaaclab`, `pyroomacoustics`, `scipy`, `soundfile`, protobuf, ROS 2, and torch. Isaac Lab tests cover class-loader behavior, fallback recovery, fixed-shape buffers, selected update/reset, stage binding, entity binding, CUDA placement when available, and error paths. Extension tests cover import-safe startup/shutdown, fake UI construction, authoring, discovery, overlays, latest-frame export, config import/export, Replicator lifecycle with a fake runtime, and readable missing-runtime errors.

Figure 2 shows the validation flow conceptually. The important property is that evidence is generated from artifacts, not prose. The live evidence report parser reads JSON/JSONL outputs from the Isaac Sim smoke, Lab GPU smoke, extension UX smoke, extension config, and Replicator manifest, then generates a local Markdown/PDF evidence report. That generated report may include absolute local Python paths and driver facts; those paths are evidence facts and are intentionally not portable installation instructions.

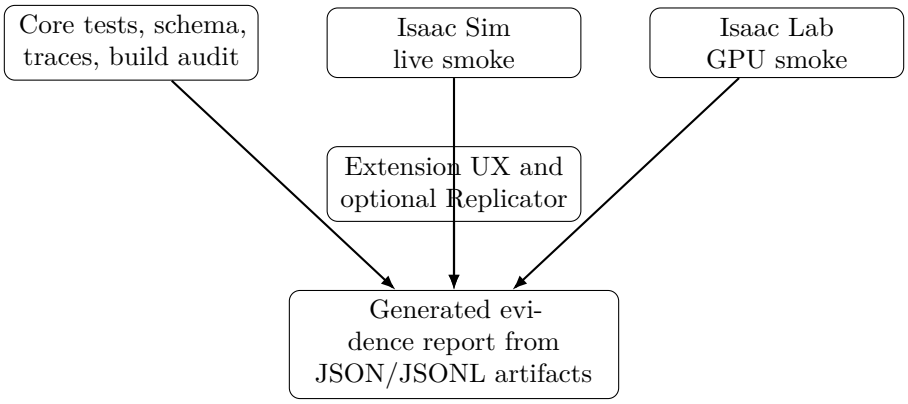

Figure 2: Validation evidence flow. Core checks validate the package contract, while live checks validate optional runtime paths on a configured Isaac workstation.

The validation does not prove acoustic realism. It proves that the public contract is stable, the implemented backends are deterministic, optional dependencies are handled explicitly, Isaac Sim and Isaac Lab integration paths operate on the recorded workstation, and generated artifacts can be parsed back into v1 frames.

6 Limitations

The package is simulation tooling, not a safety-certified perception system. Its stable L0 and L1 backends are deterministic observation generators, not calibrated acoustic models. `geometry_only` computes bearing and distance from known geometry; it does not model propagation. `tdoa_synthetic` computes direct-path synthetic delay and RMS diagnostics; it does not model reverberation, multipath, occlusion, microphone frequency response, clipping, automatic gain control, hardware clock recovery, stochastic electronics noise, or physical self-noise spectra.

The optional `room_acoustics` backend is an approximate L2 path. It can generate room impulse responses and waveform-derived TDOA diagnostics when `pyroomacoustics` is installed, but it does not establish realistic material behavior, source directivity, calibrated microphone response, production beamforming, mixed-source separation, or sim-real transfer. File-backed audio loading is intentionally narrow: public relative paths and matching sample rates are expected.

Array geometry imposes mathematical limits. A two-microphone array cannot uniquely resolve front/back ambiguity from TDOA alone. The package exposes this ambiguity through candidate

bearings, ambiguity class, ambiguity reason, and Lab ambiguity masks instead of hiding it. Four or more non-collinear microphones are recommended for robust DOA demonstrations, but even then the package's L1 estimate remains a synthetic direct-path diagnostic rather than a complete estimator benchmark.

Isaac Sim and Isaac Lab integration require user-managed NVIDIA runtimes for live validation. Pure imports are supported without those runtimes, but live stage capture requires the relevant Isaac/Kit/USD modules. Live GPU Lab validation requires CUDA visibility; the GPU target is expected to fail when audio tensors, timestamp tensors, or outdated masks fall back to CPU. Local live evidence is therefore environment-specific.

The Isaac Sim stage layer supports common metadata, transform stacks, semantic discovery signals, time-coded pose reads, and fallback attributes. It does not infer arbitrary robot articulation semantics, does not register a custom USD schema, and does not claim to support every possible USD variant or project-specific asset convention. The Isaac Lab entity layer supports common root/body tensor patterns through duck typing; unusual task-specific state layouts may require a small adapter or a custom provider.

The reference Omniverse extension is not an official NVIDIA extension. It is included as a reference UX for this project. Its Replicator path depends on Kit's available Python APIs. Writer registration, annotator registration, flush behavior, and viewport screenshot capture may vary by Kit version. When those APIs are unavailable, package-native JSON/JSONL export remains the stable path.

7 Discussion and Future Works

The v1 package establishes a stable interface between simulated audio scene state and downstream robotics consumers. This is the correct foundation for subsequent work because it separates three concerns that are often conflated: the serialized observation contract, the acoustic fidelity of a backend, and the downstream task adapter. A downstream project can now consume `AudioSensorFrame` records or Lab tensors without depending on whether future releases add more advanced acoustic propagation or calibration tools.

The immediate next step is documentation and artifact synchronization. The public showcase should point to the revised paper-style technical report and memo, and the old pre-final PDFs should be replaced. Live evidence should be regenerated whenever the Isaac runtime changes, especially if `pyroomacoustics` is installed in the Isaac Python environment and the optional L2 backend can be live-smoked instead of skipped.

Automation is the next engineering priority. Core validation is already suitable for ordinary CI, while live Isaac Sim and Isaac Lab checks require GPU-capable, licensed, non-sandboxed runners. A practical CI strategy is to keep core contract checks mandatory on every change and run live Isaac checks on scheduled or release-candidate workflows where the required runtimes are available.

Feature work should proceed in layers. L3 advanced realism can add richer wave/RIR behavior, material metadata, occlusion approximations, directivity, noise models, and estimator realism without changing v1 required fields. L4 sim-real calibration should be treated as tooling: measured array pose, gain, time offset, noise artifacts, and sim-vs-real comparisons. ROS 2 and other middleware adapters should remain downstream or optional modules until their contracts are clear. Broader Replicator support can improve dataset workflows, but it should not replace the package-native JSON/JSONL frame trace.

The final assessment is that `isaac-audio-sensors` is v1-complete within its declared scope. It provides a stable frame contract, deterministic backends, optional room-acoustics integration, Isaac

Sim and Isaac Lab paths, a reference Kit UX, validation evidence, and explicit limitations. The package is ready to serve as the audio-sensor substrate for downstream experiments, provided those experiments respect the v1 boundary and do not treat the package as a complete acoustic realism or sim-real calibration system.

A Release Status and Installation Notes

Table 4 records the v1 release status in compact form. The table belongs in the appendix because it is a release-management reference rather than the central model description.

Table 4: V1 release-scope summary.

Surface	Classification	V1 interpretation
`AudioSensorFrame` v1	Stable	Public JSON shape, units, coordinate convention, provenance, ambiguity fields, schema export, and trace round trips are compatibility commitments.
`geometry_only`	Stable L0	Deterministic geometry-only bearing/distance/sector backend.
`tdoa_synthetic`	Stable L1	Direct-path synthetic delay/RMS backend with explicit ambiguity metadata.
`room_acoustics`	Supported optional L2	Dependency-gated optional backend using `pyroomacoustics`.
Isaac Sim sensor path	Supported	Live stage capture and JSON/JSONL output when a compatible Isaac Sim runtime is supplied.
Isaac Lab sensor path	Supported	Lab-compatible sensor surface when a compatible Lab runtime is initialized.
Omniverse extension	Reference UX	Project extension for authoring, binding, inspection, debug overlays, and exports. Not an official NVIDIA extension.
Replicator recording	Optional extension capability	Lazy Kit-only writer path. Not a core dependency and not required for package traces.
L3/L4 acoustic realism and calibration	Future work	Documented directions, not complete v1 runtime systems.

The pure Python core supports Python 3.10 or newer. The base dependencies are intentionally small: `numpy` and `tomli` on Python versions earlier than 3.11. Development validation uses the `dev` extra; room-acoustics support uses the `room` extra, which brings `pyroomacoustics`, `scipy`, and `soundfile`. Isaac Sim, Isaac Lab, Omniverse, Kit, CUDA, and torch are not PyPI dependencies of the core package; live checks should use the Python interpreter supplied by the user's Isaac installation.

```
python3 -m venv .venv
. .venv/bin/activate
```

```
python -m pip install --upgrade pip
python -m pip install -e ".[dev]"
python -m pytest
python -m ruff check .
python -m build
python scripts/audit_distribution.py --dist-dir dist
python -m isaac_audio_sensors.cli export-schema --out
↪  /tmp/audio_sensor_frame.v1.schema.json
```

Optional live checks are run only when the corresponding runtime is available:

```
make live-isaac-sim-audio ISAAC_SIM_COMMAND="$ISAAC_SIM_PYTHON"
make live-omniverse-extension-ux ISAAC_SIM_COMMAND="$ISAAC_SIM_PYTHON"
make live-isaac-lab-audio-gpu ISAAC_LAB_PYTHON="$ISAAC_LAB_PYTHON"
make live-evidence-report
```

B Stable Lab Observation Shapes

Table 5: Isaac Lab audio observation buffers.

Field	Shape	Type	Semantics
`event_presence`	$[N, E]$	Boolean	Authoritative mask for real detections versus padding.
`bearing_deg`	$[N, E]$	Float32	Bearing in degrees; padded/unknown entries use `nan`.
`confidence`	$[N, E]$	Float32	Backend confidence value in $[0, 1]$.
`sector_onehot`	$[N, E, 8]$	Float32	Eight-sector one-hot encoding using the package sector order.
`per_mic_rms`	$[N, E, M]$	Float32	Per-microphone RMS values for each event.
`ambiguity_mask`	$[N, E]$	Boolean	True when the corresponding DOA estimate has non-null ambiguity.

References

[1] NVIDIA Corporation. *Sensors – Isaac Sim Documentation*. Last updated March 18, 2026. `https://docs.isaacsim.omniverse.nvidia.com/latest/sensors/index.html`.

[2] NVIDIA Isaac Lab contributors. *isaaclab.sensors – Isaac Lab Documentation*. `https://isaac-sim.github.io/IsaacLab/main/source/api/lab/isaaclab.sensors.html`.

[3] Robin Scheibler, Eric Bezzam, and Ivan Dokmanic. "Pyroomacoustics: A Python package for audio room simulations and array processing algorithms." arXiv:1710.04196, 2017. https://arxiv.org/abs/1710.04196.

[4] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. "SoundSpaces: Audio-Visual Navigation in 3D Environments." arXiv:1912.11474, 2019. https://arxiv.org/abs/1912.11474.

[5] NVIDIA Corporation. *Omni Replicator Python API Documentation*. https://docs.omniverse.nvidia.com/py/replicator/1.11.14/source/extensions/omni.replicator.core/docs/API.html.

[6] Patrizio Acquadro. *isaac-audio-sensors*. Public source repository. https://github.com/PatrizioAcquadro/isaac-audio-sensors.