Audio Engineering Society

Conference Paper 29

Presented at the AES International Conference on Machine Learning and Artificial Intelligence for Audio
2025 September 8–10, London, UK

This paper was peer-reviewed as a complete manuscript for presentation at this conference. This paper is available in the AES E-Library (http://www.aes.org/e-lib), all rights reserved. Reproduction of this paper, or any portion thereof, is not permitted without direct permission from the Journal of the Audio Engineering Society.

Integrating IP broadcasting with audio tags: Workflow and challenges

Rhys Burchett-Vass[2], Arshdeep Singh[1,2], Gabriel Bibbó[1,2], and Mark D. Plumbley[1,2]

[1]*Centre for Vision Speech and Signal Processing (CVSSP)*
[2]*University of Surrey, UK*

Correspondence should be addressed to Arshdeep Singh (arshdeep.singh@surrey.ac.uk)

ABSTRACT

The broadcasting industry has adopted IP technologies, revolutionising both live and pre-recorded content production, from news gathering to live music events. IP broadcasting allows for the transport of audio and video signals in an easily configurable way, aligning with modern networking techniques. This shift towards an IP workflow allows for much greater flexibility, not only in routing signals but with the integration of tools using standard web development techniques. One possible tool could include the use of live audio tagging, which has a number of uses in the production of content. These could include adding sound effects to automated closed captioning or identifying unwanted sound events within a scene. In this paper, we describe the process of containerising an audio tagging model into a microservice, a small segregated code module that can be integrated into a multitude of different network setups. The goal is to develop a modular, accessible, and flexible tool capable of seamless deployment into broadcasting workflows of all sizes, from small productions to large corporations. Challenges surrounding latency of the selected audio tagging model and its effect on the usefulness of the end product are discussed.

1 Introduction

The audio track contains a wide array of descriptive information about the sound events in a scene. Detecting sound events in real-time could have a number of uses in the broadcasting industry, from aiding operators in programme creation to enhancing end user accessibility. For example, BBC Research and Development [1] employed a sound event detection framework to identify sounds that may disrupt the ambience of a program. This work was targeted at the BBC's *Autumnwatch* programme which uses wildlife cameras capturing the movement of animals. To avoid undesirable noises interrupting the live stream, such as cars passing by or people talking, an icon is overlayed onto the operator's interface, indicating the undesirable sound event so the operator does not to switch to that source. Another use of sound event detection is closed captioning. While the majority of the existing work in captioning broadcast audio has been focused on analysing the speech events [2, 3], only a few attempts of identifying sound events in a real time (live) transmission has been conducted [1]. To achieve full captioning (closed captions of both sound events and speech) within IP broadcasting, general sound event detection models that detect sound events including speech are required.

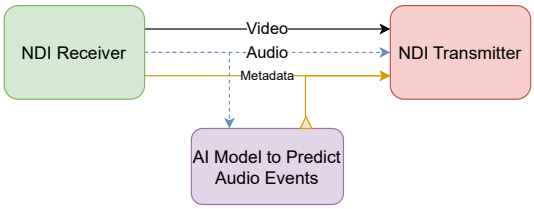


Fig. 1: Basic flow of audio, video and metadata frames travelling through their respective data streams using Network Device Interface (NDI) system.

Internet Protocol (IP) broadcasting describes the process of transmitting audio and video signals from one location to another using IP networking. One approach traditionally used for transmitting audio/video is Serial Digital Interface (SDI), with fixed connections between dedicated hardware devices. IP broadcasting allows software to replace some of these hardware devices, enabling greater scalability and easy re-configuration. Cloud technology and containerisation methods such as Docker [4] can be utilised to take advantage of such scalability.

There are a few challenges in creating software for use in an IP broadcasting environment. First, as with most modern web applications, is scalability and containerisation of software, which allows the infrastructure to adapt depending on the demand on the system by starting up new containers when required. Containerisation also allows for the same task to be conducted independently on different streams or sources by having a container per stream. Another advantage of containerisation is that, if a fault occurs on one container, it does not damage the entire system as a whole and can be fixed independently. The second challenge is handling of the inelastic audio and video traffic, which does not adapt well to changes in network conditions, without introducing delay and jitter to the transmission.

To overcome above challenges and integrate sound event data into IP broadcasting, this paper contributes:

- Containerising audio tagging applications to isolate each component from the other elements of the system, i.e. other processing units, transmission and reception code. Code isolation allows the component to run on any machine. Containerisation makes it easy to create multiple instances of a component if needed, allowing for scalability and the use of cloud platforms.
- Leveraging an Artificial Intelligence (AI) model to generate audio tags to transmit meta-information alongside audio. This added information about the contents of the audio track has a number of uses in the area of automation - from better production tools to improved accessibility via more descriptive automated captioning. An overall proposed framework is shown in Figure 1 and more details including challenges around our approach can be found in Section 5. Our codes are available at Github[1].

2 Related work

2.1 A Brief Overview on IP Broadcasting Technology

There are a few technologies currently used for IP broadcasting. The first of these are described in standards from the Society of Motion Picture and Television Engineers (SMPTE) as the ST 2110 suite of standards [5]. SMPTE standards are used by the industry with examples including the Serial Digital Interface (SDI) standards for transmission between equipment over a direct connection, i.e. coaxial or fibre optic cable. Secondly, the Networked Media Open Specifications (NMOS) from the Advanced Media Workflow Association (AMWA) uses ST 2110 along with other standards to define APIs allowing for the connection of multiple receivers and senders on a network in a vendor agnostic way. NMOS is not software, but a set of specifications aiding development in the software. On the other hand, Network Device Interface (NDI) by NewTek [6] is an open standard with fully developed software and Software Development Kit (SDK), designed to allow for easy integration of IP broadcasting into existing software by utilising the NDI SDK.

2.2 Audio Recognition in Broadcasting

There has been some work conducted in broadcasting related to recognition of audio events [1]. However most examples relate to speech recognition and transcription, commonly used for tagging content for archiving purposes. Recognising speech allows for easy searching without having to manually tag content. For example, Raimond et al. [2] describe a system to automate the tagging of content within the BBC's

[1] https://github.com/Rhysbv/panns_ndi

radio archive based on speech audio. Levin et al. [3] describe a system using automatic speech recognition for captioning of news programming. This system runs against a re-speaker, a person repeating speech in a more readable and understandable way for the system, avoiding the issues surrounding the acoustic environment and overlapping speakers. However, this system only supports the processing of speech and does not consider sound events in general. More modern proprietary solutions do exists [7, 8] which remove the requirement for a re-speaker but are still incapable of including sound events. Additionally, BBC Research and Development [1] have designed an application program (a software tool) to identify sound events for the purpose of audio monitoring.

3 System Design

In contrast to previous work, we separate the audio tagging software from any other application programs. In our work, a modular approach is opted for and a container specifically for general audio tagging is built to allow multiple applications on the network to take advantage of the technology without repeating work. This is helpful considering the computational overhead associated with AI models. Our system can include the environment audio event monitoring system as described by BBC Research and Development [1] in addition to other systems e.g. for captioning.

3.1 Selecting IP Broadcasting Approach

For our work, we need an IP based technology that is both well used by the industry and is simple to implement, allowing for wide adoption of the audio tagging technology. Standards have been created to support new IP based workflows. One example discussed in Section 2.1 is the ST 2110 suite of standards authored by SMPTE. These describes the transport of compressed and uncompressed audio, video and metadata via individual Real-time Transport Protocol (RTP) streams. However, the complexity of understanding the suite of standards means that it is only practical for large corporations to implement. An alternative standard mentioned in Section 2.1 is the Network Device Interface (NDI), an open standard created by NewTek [6] which has an easy to use Software Development Kit (SDK). NDI is available in a wide variety of software and hardware applications, enabling its wide spread adoption in both large and small operations. This has lead to NDI being the selected approach for our work. NDI transports data in the form of frames that contain the relevant data as well as supporting information to aid in its use. There are three types of frames used by NDI: audio, video and metadata (Figure 1). NDI also handles the detection of sources allowing for routing of NDI frames.

3.2 AI Model used for Audio Tagging

To identify audio tags, we use convolutional neural networks (CNNs), that have shown good performance in many audio classification tasks [9, 10]. For example, pre-trained audio neural networks (PANNs) [10] have been widely used to recognize a variety of audio events. A description of the CNN models used in this paper for predicting the audio tags is given below.

Pre-trained Audio Neural Networks (PANNs): CNN14 [10] is a pre-trained audio neural network that is trained on Google AudioSet dataset [11]. CNN14 is trained by extracting the log-mel spectrograms from the audio clips. CNN14 has 81M parameters and it takes 21G multiply-accumulate operations (MACs) to predict tags of the audio of length 10 seconds. The trained CNN14 can predict wide range of sound events such as car passing by, speech, siren, animal etc. This helps identifying sounds in the wide array of possible scenarios the system could be exposed to, i.e. different types of broadcast programming such as news gathering in various locations or a panel show within a studio.

Efficient PANNs: E-PANNs [12] is an efficient version of original PANNs with reduced memory requirement (24M parameters) and a reduced computational complexity (13G MACs per 10 seconds audio). The efficient AI models are beneficial in an IP networking environment, especially one involving inelastic traffic (network traffic that is sensitive to variations in delay, e.g. audio and video streams). This will be explored in Section 5.

3.3 NDI Integration

We use the NDI SDK [13] to create a software module including the PANNs algorithm. Due to the reliance on Python based packages within the PANNs module such as PANNs inference [14], we use Python for implementation. Specifically, community Python bindings [15] to interface between Python and the C++ SDK are used to enable NDI support. An additional Python package

is created to simplify the process of integrating NDI into both the PANNs module and potential proof of concept applications described in Section 4. Our Python package contains three classes: A receiver, transmitter and finder as seen in Figure 1. An application can find NDI network sources using the finder class. Frames can be received from an NDI source using the receiver class. Received Frames can then be processed and transmitted by creating its own NDI source using the transmitter class. An example of how this is used here can be seen in Figure 1. The flow of audio, video and metadata frames is uninterrupted between the receiver and transmitter. Each audio frame is intercepted and a copy is taken for analysis while the original copy is sent straight to the transmitter, minimising delay and jitter. One issue surrounding the community supplied Python bindings were the associated bugs, especially surrounding memory management. This led to having to convert each frame to a Python dataclass so that it could be effectively freed and dealt with by the Python garbage collector, an issue that would not have been encountered using the original C++ SDK.

3.4 Integrating Sound Events Metadata

In order for E-PANNs to produce sound event predictions from the PANNs model and make it compatible with other NDI applications, we follow the pipeline as shown in Figure 3 that takes the incoming audio frames from NDI and creates metadata frames containing the audio tag to be sent across the network.

We use two ring buffers. The first ring buffer stores incoming audio frames. From each audio frame, we extract the individual audio samples and store them in a second ring buffer. Once a sufficient number of samples has been collected in the second ring buffer the entire contents of the ring buffer is fed into the PANNs model. The size of the second ring buffer determines the duration of the audio window that PANNs analyses. The impact of the size of the window on the models latency is discussed in Section 5. To distribute the predicted sound event across the NDI network, we use metadata frames. These frames transport XML data, which can include third-party metadata as used here. The output text from PANNs is inserted into an XML template for transmission. Other NDI applications can then receive this XML via the metadata frames to access the sound event prediction.

A summary of various steps is explained below:

1. Store received audio frames in the first ring buffer.
2. Extract the floating point Pulse Code Modulated (PCM) audio samples from each frame and store these in the second ring buffer.
3. Wait until a given number of samples have been collected.
4. Feed the entire contents of the second ring buffer into the CNN model.
5. Generate a metadata frame containing the prediction from the CNN model.

4 Example Workflow

The proposed containerised component allows for the integration of audio tagging capabilities into a multitude of different systems and use cases. Below, we provide two examples of integration of audio tagging system into existing IP broadcasting framework,

4.1 Audiowatch Example

Figure 2 demonstrates a pipeline inspired by the BBC Audiowatch project, showing an audio tagging module separated from other application programsWe use Docker [4] for containerisation, creating multiple instances of the audio tagging software to analyse several NDI sources simultaneously. A sound event detection front end is a dashboard user interface as shown in Figure 3 and it generates metadata corresponding to input audio. Metadata containing sound event information is then sent to the icon selector module for processing. Next, various icon selector containers extract the sound events from the audio track supplied within the metadata frames. After identifying the unwanted sound events, an appropriate icon overlay is transmitted as an NDI video frame. Next, a video mixing software tool such as Open Broadcaster Software (OBS) [16] is used to superimpose the icon onto the original video source for displaying on the operators multiview, which is used to monitor all video sources.

4.2 Online Closed Captioning

Another example integration could be the use of audio tagging to enhance closed captioning. As discussed in Section 2.2, while work has been conducted to automated closed captioning in real time using automatic

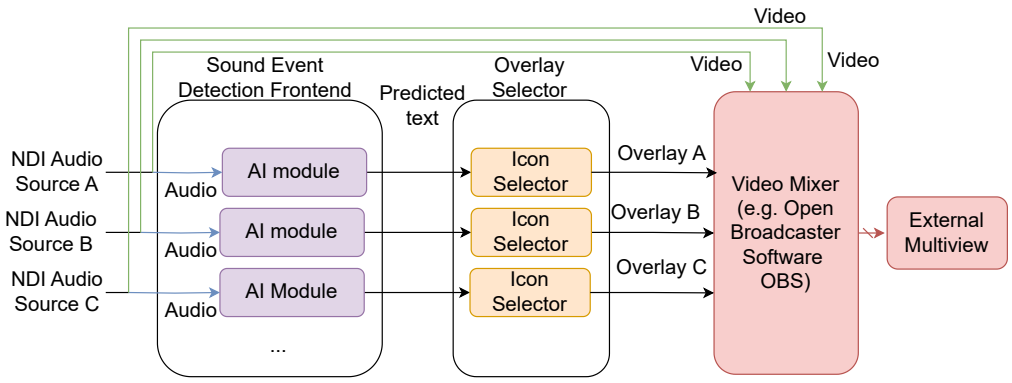


Fig. 2: Proposed integrated pipeline: Audiowatch [1] framework with a separated audio tagging unit.

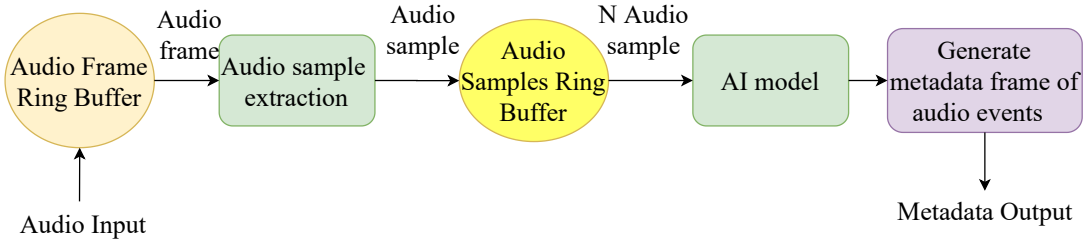


Fig. 3: Metadata generation pipeline.

speech recognition, these do not include descriptions of sound events. By combining the two technologies, full closed captioning could be achieved. This would involve first parsing the audio through the audio tagging model using our container. When the result is returned as human speech, the audio would then be passed through a second speech recognition model to generate subtitles. One major concern would be the accuracy of the audio tagging model. If the speech was not always detected, we would miss large portions of speech text. Additionally the difference in latency between a sound event being inserted and speech going through two models would have to be accounted for.

5 AI model Integration Challenges

There are a number of integration challenges to consider while designing AI based software fit for broadcasting. These challenges include the accuracy of the prediction and the latency of the model delaying the signal. Generally, PANNs and E-PANNs give similar prediction results.

Model latency: The latency of the model here describes the amount of time it takes given a number of audio samples to produce an accurate sound event prediction. Consideration of the model latency is significant given that we are dealing with inelastic audio and video network traffic. This means that any delay in processing contributes to a delay in the resulting transmission depending on the infrastructure. The delay can be mitigated using a design similar to that shown in Figure 1, however there is the issue of predictions being desynchronised from events heard in the audio track Although we have minimal control over the IP network using the audio tagging module, and thus cannot manage the network's latency, we can still select an optimal model that minimises latency while maintaining accuracy.

Buffer size versus model latency: To analyse buffer size and latency of model, we performed experiments using a set of audio recordings with known sound events. The first audio recording is taken directly from the PANNs repository. It involves a telephone ringing followed by human speech and is of seven seconds.

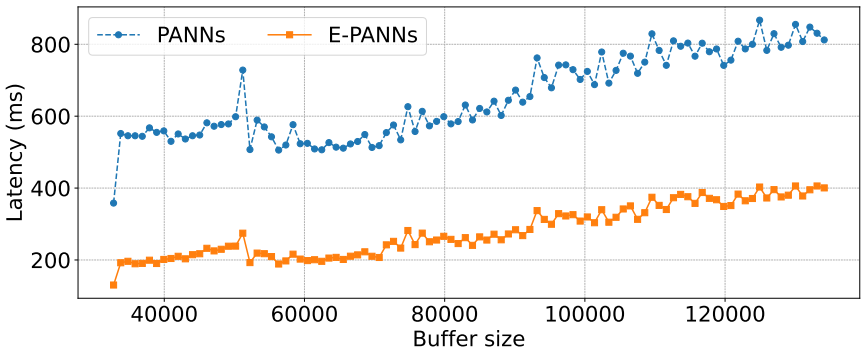


Fig. 4: PANNs/E-PANNs model latency vs buffer size when inputting audio sampled at 48KHz. Experiments are performed a on AMD Ryzen 5 2500U system at 2GHz.

The second audio recording of a car driving into the distance. The third audio recording is created by mixing a car driving and a running river sound events.

Given the audio recordings, we analyse the latency of the CNN model at different number of audio samples. We generate different length audio segments. The audio samples taken are of multiples of 1024 (assuming frames containing 1024 samples are used) and represents the size of the buffer. Given the audio samples of different length, we use the PANNs or E-PANNs model to produce predictions while measuring the time taken for the model to produce a prediction. Figure 4 shows the latency of the PANNs and E-PANNs models at different buffer size. Both PANNs and E-PANNs follow a similar trajectory, with E-PANNs showing a considerable improvement in latency. This suggests that choosing an appropriate model contributes to try to improve latency and hence making integration of audio events more real-time while using less resources.

Experiments found that in our example a buffer size of 47 frames with 1024 samples each This equates to an audio window with a duration of 1.002s sampled at 48KHz, that gives correct results with the minimal latency. Model latency computed on an AMD Ryzen 5 2500U and Intel Core i9-13900HX hardware can be found in Figure 4 and prediction results can be found in Table 1.

Table 1: Comparison of PANNs and E-PANNs predictions across different buffer sizes

Buffer Size	Groundtruth	PANNs	E-PANNs
32768	Tone Car River & Car Phone & Voice	Sine wave Music Waves, surf Telephone bell ringing	Sine wave Music Boat, water vehicle Telephone bell ringing
38912	Tone Car River & Car Phone & Voice	Sine wave Music Vehicle Music	Sine wave Music Boat, water vehicle Telephone bell ringing
48128	Tone Car River & Car Phone & Voice	Sine wave Silence Vehicle Music	Sine wave Vehicle Vehicle Telephone bell ringing
56320	Tone Car River & Car Phone & Voice	Sine wave Vehicle Vehicle Music	Sine wave Vehicle Vehicle Telephone bell ringing
61440	Tone Car River & Car Phone & Voice	Sine wave Vehicle Vehicle Telephone bell ringing	Sine wave Vehicle Vehicle Telephone bell ringing
65536	Tone Car River & Car Phone & Voice	Sine wave Vehicle Vehicle Telephone bell ringing	Sine wave Vehicle Vehicle Telephone bell ringing
102400	Tone Car River & Car Phone & Voice	Sine wave Vehicle Waves, surf Telephone bell ringing	Sine wave Vehicle Vehicle Telephone bell ringing
134144	Tone Car River & Car Phone & Voice	Sine wave Vehicle Vehicle Speech	Sine wave Vehicle Vehicle Speech

6 Discussion and Conclusion

The integration of IP broadcasting with audio tagging offers significant potential for enhancing broadcast workflows, but it also presents several challenges. The transition to IP broadcasting enables a more flexible, scalable, and reconfigurable infrastructure compared to traditional methods based on Serial Digital Interface (SDI). This flexibility is further enhanced by containerisation technologies making the system more resilient and adaptable. However, implementing an audio tagging system introduces challenges primarily related to latency and the accuracy of audio tagging models.

One of the primary challenges discussed is the latency

associated with the audio tagging model. Given the real-time nature of broadcasting, any delays introduced by processing can impact the overall operation. This makes the choice of buffer size crucial. A smaller buffer reduces latency but might compromise the accuracy of sound event detection. Conversely, a larger buffer improves accuracy but increases latency. Experiments conducted (Table 1) show that an acceptable balance can be achieved with a buffer size of 48128 samples, which provides an acceptable latency while maintaining accuracy. The use of Efficient PANNs (E-PANNs) further helps in reducing the computational complexity and memory requirements, making it a suitable choice for real-time applications.

Containerisation offers a robust solution to scalability issues. By isolating the audio tagging functionality into a microservice, it becomes possible to scale the system by simply adding more containers as needed. This isolation also ensures that a fault in one container does not affect the entire system, enhancing overall reliability. The use of Docker to containerise these services allows for easy deployment and management across different network setups. Additionally, the integration with NDI, which is widely adopted in the industry, ensures broad applicability.

Despite these advantages, real-world deployment of such a system is not without hurdles. The reliance on Python bindings to interface with the NDI SDK, while practical, introduces potential issues with memory management that need careful handling.

6.1 Conclusion

Integrating IP broadcasting with audio tagging presents a promising advancement for the broadcasting industry. The use of containerisation and audio tagging for real-time sound event detection can significantly enhance content production and accessibility. However, addressing the challenges of latency, accuracy and real-world deployment is crucial for the successful implementation of this approach. Future work includes re-writing the codebase to use the NDI C++ SDK directly, avoiding the issues surrounding the community Python bindings. Additionally, we would like to analyse more complex models such as transformers [17, 18] within our broadcasting framework. Finally, the creation of the discussed proof of concept applications would allow for full demonstration of the usefulness of this technology.

7 Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/T019751/1] and an Adobe Research Gift. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

References

[1] Ward, S. and Dawes, R., "AudioWatch - Live audio monitoring for Autumnwatch 2021 - BBC R&D," 2021.

[2] Raimond, Y., Lowis, C., Hodgson, R., and Tweed, J., "Automated semantic tagging of speech audio," in *Proceedings of the 21st International Conference on World Wide Web*, pp. 405–408, 2012.

[3] Levin, K., Ponomareva, I., Bulusheva, A., Chernykh, G., Medennikov, I., Merkin, N., Prudnikov, A., and Tomashenko, N., "Automated closed captioning for Russian live broadcasting," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[4] Docker, "Docker: Accelerated Container Application Development," 2022, https://www.docker.com/.

[5] SMPTE, "SMPTE OV 2110-0:2018 - SMPTE Overview Document - Professional Media over Managed IP Networks Roadmap for the 2110 Document Suite," 2018, https://cta-redirect.hubspot.com/cta/redirect/5253154/c8fbb67b-5cba-421d-ad95-29de41239ef7.

[6] NewTek, "NDI 5.6 White Paper," Technical report, NewTek, 2023, https://ndi.video/wp-content/uploads/2023/09/NDI-5.6-White-Paper-2023.pdf.

[7] IBM, "Closed Captioning Software - IBM Watson Media," 2025, https://www.ibm.com/consulting/closed-captioning.

[8] "enCaption: Automated Closed Captioning System | ENCO Systems," 2025.

[9] Purwins, H., Li, B., Virtanen, T., Schlüter, J., Chang, S.-Y., and Sainath, T., "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, 13(2), pp. 206–219, 2019.

[10] Kong, Q., Cao, Y., Iqbal, T., Wang, Y., Wang, W., and Plumbley, M. D., "PANNs: Large-scale pretrained audio neural networks for audio pattern recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28, pp. 2880–2894, 2020.

[11] Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., and Ritter, M., "Audio Set: An ontology and human-labeled dataset for audio events," in *IEEE International Conference on acoustics, Speech and Signal Processing (ICASSP)*, pp. 776–780, 2017.

[12] Singh, A., Liu, H., and Plumbley, M. D., "E-PANNs: Sound Recognition Using Efficient Pre-trained Audio Neural Networks," in *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, volume 268, pp. 7220–7228, Institute of Noise Control Engineering, 2023.

[13] NewTek, "NDI SDK V6.0.1," 2024, https://ndi.video/for-developers/ndi-sdk.

[14] kong, Q., "qiuqiangkong/panns_inference," 2020, original-date: 2020-03-08T06:22:30Z.

[15] Kondo, N., "NDI-Python bindings," 2022, https://github.com/buresu/ndi-python.

[16] Project, O., "OBS studio project," 2025, https://github.com/obsproject/obs-studio.

[17] Gong, Y., Chung, Y.-A., and Glass, J., "AST: Audio spectrogram transformer," *Interspeech*, 2021.

[18] Schmid, F., Koutini, K., and Widmer, G., "Efficient large-scale audio tagging via transformer-to-CNN knowledge distillation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2023.