

XR Musical Performance: Real-Time MIDI Bridge Enabling Tempo-Synchronized DAW Integration for Immersive Music-Making

Maria Grazia Corino, Ali Al Housseini, Omran Ayoub, Tiziano Leidi
University of Applied Sciences and Arts of Southern Switzerland, Lugano, Switzerland
{mariagrazia.corino, ali.alhousseini, omran.ayoub, tiziano.leidi}@supsi.ch

Abstract—Extended-reality (XR) music applications offer immersive, gesture-driven expression but operate as self-contained instruments, rarely routing performance data into a professional Digital Audio Workstation (DAW) or synchronizing with it. We bridge this gap by treating Musical Instrument Digital Interface (MIDI) as the canonical command language and wireless transports as interchangeable carriers. A mixed-reality instrument on the Meta Quest 3 maps hand gestures to MIDI and dispatches them as Open Sound Control (OSC) over Wi-Fi or as MIDI over Bluetooth Low Energy (BLE-MIDI) to a desktop companion, which routes MIDI into Ableton Live, synchronizes tempo via Ableton Link, and renders step-sequencer patterns locally; state flows back to the headset, keeping the instrument bidirectionally locked to the DAW. A latency evaluation shows the software bridge accounts for 0.07–7.1% of the end-to-end delay, so latency is governed by the wireless transport, with Wi-Fi and BLE-MIDI offering complementary typical-case and worst-case profiles.

Index Terms—extended reality, MIDI controller, digital audio workstation, OSC, BLE-MIDI, Ableton Link, Meta Quest 3, networked music performance

I. INTRODUCTION

The commercial landscape of eXtended Reality (XR) music-making has matured considerably, with platforms such as PatchXR, Virtuoso VR, Soundela, SynthVR, and Tribe XR offering immersive environments for musical expression [1]–[5]. These applications have demonstrated that virtual and mixed-reality spaces can serve as compelling stages for music creation, enabling users to perform gestures that would be impossible with physical instruments, visualize sound spatially, and engage with musical material in ways that are not bound by real-world constraints.

Despite these advances, current XR music applications typically remain self-contained systems [6]. Most focus on implementing virtual instruments and handling sound generation internally, while comparatively little attention has been given to interoperability with established music production ecosystems through external protocols such as Musical Instrument Digital Interface (MIDI) or Open Sound Control (OSC) [7]. Consequently, these systems are well suited for standalone creative exploration but remain isolated from the professional studio ecosystem.

This work is supported by MUSMET project funded by the EIC Pathfinder Open scheme of the European Commission (grant agreement n. 101184379), and has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

The implications of this isolation are significant. A musician performing in an XR environment typically cannot route their MIDI data into a Digital Audio Workstation (DAW) for sound production and arrangement, layer the performance with studio-grade instruments and effects, or synchronize and record it alongside collaborators working in a conventional studio toolchain. For the electronic musicians, producers, and sound designers who build their workflows around a DAW, this separation forces a choice between immersive interaction and professional-grade production, a compromise that caps XR’s potential as a serious music-making tool.

What is needed, therefore, is a communication path that is *standards-compliant*, meaning that MIDI data flows from the XR instrument to the DAW through a standard virtual MIDI port, conforming to the MIDI 1.0 specification [8]; *low-latency*, meaning that software-induced delay remains negligible relative to the underlying wireless transport; and *bidirectional*, meaning that the XR instrument not only sends performance data to the DAW but also receives and reacts to tempo, transport, and state changes originating from it. Such a communication path would bridge immersive XR instruments with established studio workflows, allowing them to function as first-class controllers within existing music production ecosystems rather than as isolated creative applications. To the best of our knowledge, existing XR music systems do not provide this combination of standards-compliant, low-latency, bidirectional integration with external DAWs.

In this work, we introduce a system that addresses this gap by treating MIDI as the canonical command language and wireless transports as interchangeable carriers. The architecture comprises two components: a mixed-reality instrument on the Meta Quest 3 (Unity 6, C#), and a companion desktop application (JUCE, C++, macOS and Windows) that bridges MIDI to Ableton Live. The performer can play live in real time or trigger step-sequencer patterns that stay locked to the session, with gestural input mapped to standard MIDI messages and transmitted as OSC over Wi-Fi or as MIDI over Bluetooth Low Energy (BLE-MIDI). Time-critical work runs on the host, off the wireless link, and transport and state changes flow back to the headset over the same transport, closing the bidirectional loop.

We make two contributions. First, we present a working system that connects a standalone XR headset to a professional

DAW as a bidirectional, synchronized MIDI endpoint: beyond streaming performance data from the headset to the DAW, it closes the return path so that tempo, transport, and state changes flow back to the instrument, letting it behave like a hardware MIDI controller rather than a one-way sender. Second, we introduce a dual-transport design in which either carrier, OSC over Wi-Fi or BLE-MIDI, can be swapped behind a common abstraction without changing the instrument or the DAW-facing interface, showing that MIDI can serve as a transport-agnostic canonical layer while the wireless transport remains an interchangeable implementation choice where platform support allows.

II. RELATED WORK

Connected musical instruments and the Musical Metaverse. A growing body of work envisions musical instruments as networked, interoperable devices rather than isolated tools [10], [11]. The Internet of Musical Things (IoMT) frames instruments as connected entities that exchange musically meaningful data across systems [12], and this vision has recently extended into immersive space under the notion of the Musical Metaverse, where XR environments become venues for connected musical interaction and collaboration [13], [14]. A central premise of this line of work is interoperability: for immersive instruments to participate in real musical practice, their data must flow into and out of the wider ecosystem of production tools. Our system contributes to this direction, connecting an XR instrument to a professional production environment through a standard MIDI interface [15].

XR musical instruments. In practice, however, current XR music platforms fall short of this interoperability. Commercial applications provide immersive environments for performance and sound design [2]–[5], and research has examined the design of virtual-reality musical instruments more broadly [16]. Across both, the headset operates as a self-contained instrument: sound is produced by an internal engine, with little path to route MIDI out to a DAW or to receive tempo and transport feedback from one. Within the Unity ecosystem specifically, existing MIDI plugins either target earlier headsets or handle internal playback without off-device routing [15], [17], [18]. The immersive instrument and the professional toolchain thus remain disjoint.

Wireless MIDI transport. Several protocols carry MIDI over wireless links. OSC [19] is flexible and based on the User Datagram Protocol (UDP), but requires a translation layer for DAW compatibility. BLE-MIDI [20] provides reliable, ordered delivery with direct MIDI 1.0 compatibility. RTP-MIDI [21] and Network MIDI 2.0 [22] offer IP-based alternatives with maturing ecosystem support. These efforts target a single transport; our work instead treats the transport layer as interchangeable, carrying the same canonical MIDI stream over either carrier.

Latency in networked music performance. The networked music performance literature has long studied the effect of transmission delay on ensemble timing [23], and has established perceptual thresholds below which delay is imperceptible

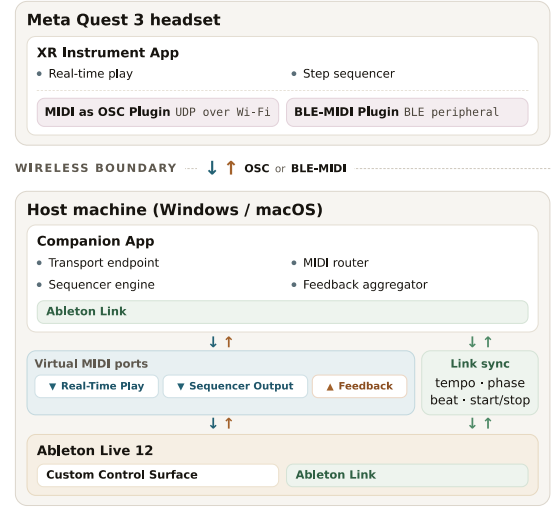


Fig. 1. System architecture bridging a Meta Quest 3 XR instrument to Ableton Live. MIDI is the canonical content; OSC over Wi-Fi and BLE-MIDI are interchangeable wireless carriers. Time-critical sequencer playback runs locally, off the wireless link, and synchronized via Ableton Link.

in interactive musical control [7], [24], [25]. Reported figures place the tolerable one-way latency for tightly coupled interaction in the low tens of milliseconds, which frames the software overhead we introduce as negligible relative to the wireless transport itself. Transport-level benchmarks for BLE-MIDI and OSC/Wi-Fi in performance contexts [26] further inform our mitigation strategy of decoupling the transport threads from the render loop and rendering sequenced patterns locally on the companion machine.

Positioning. Existing work thus separates into two non-intersecting clusters: immersive XR instruments with internal sound engines, and low-level transport libraries without an application-layer instrument, neither of which delivers the interoperability the IoMT and Musical Metaverse vision calls for. We contribute a complete vertical slice, from spatial gesture on the Quest 3, through dual-wireless MIDI transport with Ableton Link synchronization, into Ableton Live via a virtual MIDI port, with bidirectional feedback closing the loop.

III. SYSTEM ARCHITECTURE

This section presents the architecture that translates gestural input on the headset into musical output in the DAW, organized around a single principle: MIDI is the canonical content, while the wireless transport carrying it is interchangeable. We describe the two principal components, the headset application and the desktop companion, followed by the dual-transport bridge, data flow and latency mitigation, and the discovery and cross-platform mechanisms.

A. Architecture Overview

Figure 1 presents the complete signal chain of our proposed architecture. The system connects the Meta Quest 3 headset to Ableton Live through dual wireless transports.

The architecture comprises two principal components, namely, the XR application on the headset and the desktop companion. On the headset, a mixed-reality application

built in Unity 6 (6000.0.58f2) with Meta XR All-in-One SDK v78.0 [27] captures hand gestures and maps them to standard MIDI messages via an interface that abstracts the transport layer, enabling a single MIDI event stream to dispatch over either OSC/Wi-Fi or BLE-MIDI. We leverage a step sequencer on the headset to permit the performer to trigger rhythmic patterns. The desktop companion, built with JUCE v8.0.4 (C++) [28], performs five functions: (a) receives MIDI via the active transport, (b) decodes OSC-encapsulated MIDI or passes BLE-MIDI through directly, (c) writes MIDI to a virtual MIDI port connected to Ableton Live, (d) hosts an Ableton Link v4.0.0b4 instance [9] for tempo and transport synchronization, and (e) runs a local sequencer engine that renders pre-transmitted patterns against the Ableton Link timeline. Ableton Live 12 receives the MIDI stream and produces audio through the configured output device.

B. Dual-Transport MIDI Bridge and Implementation

MIDI is the canonical content; the transport layer is interchangeable. The XR application on the headset carries two native plugins that encode the same MIDI messages onto independent wireless carriers: OSC over Wi-Fi, which favors low latency through best-effort UDP delivery, and BLE-MIDI, which favors reliable delivery. We describe each in turn.

OSC over Wi-Fi. A JUCE C++ library loaded as a native Unity plugin encodes each MIDI message as an OSC packet [19] and transmits over UDP on the LAN via OSC addresses (e.g., `/midi/out/noteon`, `/midi/out/controllerevent`) with arguments such as channel, note, velocity, and pitch-bend value, depending on the message. The companion receives this stream, decodes it back to MIDI, and routes it to the virtual MIDI port. UDP is best-effort: dropped packets are not retransmitted; the next update self-heals state [26]. The transport plugin runs on a dedicated high-priority native thread independent from Unity’s render loop (72–120 fps), so a MIDI message is transmitted the instant it is produced, keeping software-induced delay negligible relative to the wireless transport, as our low-latency requirement demands.

BLE-MIDI. An Android plugin (Java) implements the standard BLE-MIDI service [20] (UUID 03B80E5A-EDE8-4B33-A751-6CE34EC4C700), with the Quest 3 as the BLE peripheral. The companion operates as a BLE central. The plugin serializes MIDI byte streams into BLE-MIDI packets with 13-bit timestamps and applies running-status optimization. BLE provides reliable delivery as an alternative to UDP’s best-effort model. The companion supports BLE-MIDI only on macOS, so the headset can pair only with its macOS version.

C. Data Flow and Latency

Outbound (headset to DAW). A gesture is captured by the XR app and mapped to a MIDI message, which is then handed to the active transport plugin. On the companion, messages are received over either transport, decoded and routed: real-time performance MIDI passes through to the virtual MIDI port

and into Ableton Live; selected command messages (transport controls and Ableton Link controls) are consumed locally. Sequencer patterns follow a distinct path: the entire pattern is sent once as a System Exclusive (SysEx) message, then rendered locally on the companion by a dedicated engine that synchronizes via Ableton Link.

Inbound (DAW to headset). The companion transmits the Ableton Link session state and transport state back to the headset, which renders this as visual feedback. MIDI messages emitted by the DAW, typically state feedback, pass back through the virtual MIDI port to the companion and on to the headset. State feedback originates in a custom Ableton Live control surface, a Python 3 remote script that maps responses to selected Control Change (CC) messages.

Latency mitigation employs two strategies: (1) on the headset, transport plugins run on independent high-priority native threads, so messages are transmitted the moment they are produced; (2) sequencer patterns are sent once as a SysEx message and rendered on the companion, co-located with Ableton Link, the virtual MIDI port and the DAW, eliminating per-note latency and jitter caused by wireless overhead.

Transport choice also carries deployment implications: network transport suits devices separated by distance, as in a studio or live-event setting, while BLE-MIDI avoids firewall and Wi-Fi-congestion issues. BLE-MIDI also works without network connectivity, since Ableton Link runs over localhost on the companion, co-hosted with Ableton Live; a LAN is needed only to synchronize tempo and transport with other devices in an Ableton Link session.

D. Discovery, Connection Management, and Cross-Platform

The companion and headset discover each other automatically over the local network, requiring no manual IP configuration, and a periodic heartbeat monitors connectivity. On disconnection, the companion issues an All-Notes-Off on both transports to prevent hanging notes. The companion runs on macOS and Windows; network transport works identically on both, while BLE-MIDI pairing is macOS-only, as Windows does not expose BLE-MIDI at the OS level. When the headset is tethered via Meta Horizon Link, the companion connects in localhost mode. The companion exchanges MIDI with the DAW through three virtual MIDI ports: two companion-to-DAW (real-time performance notes and sequencer notes), and one DAW-to-companion for state feedback. Port creation is platform-dependent: on macOS the companion creates them directly through JUCE, while on Windows it connects to existing virtual ports, which can be provided by any compatible virtual-MIDI driver.

IV. EXPERIMENTAL SETUP AND RESULTS

A. Setup and Measurement Scope

We used a Meta Quest 3 (Horizon OS 2.3) and three hosts running Ableton Live 12: a Lenovo laptop (Core i7-13700H, 32 GB RAM, Windows 11 Enterprise), an Intel MacBook Pro (Core i7, 2.6 GHz, 16 GB RAM, macOS 15.7.9), and an M1 Pro MacBook Pro (32 GB RAM, macOS 26.2). All devices

TABLE I
FORWARD-PATH LATENCY AND JITTER OF MIDI MESSAGES AT 72 FPS, WITH VALUES IN EACH CELL FOR MEAN PLAYING RATES OF 4/24/200 MSG/S

	Host	Forward latency, XR app → companion (ms; sync error ± 0.16 –0.25)					Jitter (ms)	
		Min	Median	p90	p99	Max	IQR	Delay var. p95
OSC Wi-Fi	Windows 11	0.7/0.7/0.6	1.5/1.5/2.5	4.6/4.7/5.1	10.7/9.2/9.0	100.5/69.7/29.1	1.5/1.6/1.2	4.6/4.8/3.7
	macOS 15, Intel	1.1/1.0/0.9	2.1/1.6/2.8	92.3/2.9/5.2	120.3/34.3/8.0	393.4/126.6/50.8	48.3/0.6/2.4	108.2/2.7/3.4
	macOS 26, M1 Pro	1.3/0.9/0.9	2.3/1.7/2.7	81.9/3.0/5.2	116.4/20.2/8.1	120.7/112.7/13.4	39.0/0.8/2.3	100.8/3.4/3.5
BLE- MIDI	macOS 15, Intel	2.8/3.1/3.2	11.6/11.5/12.5	17.6/18.0/19.2	31.2/31.7/33.0	37.2/44.4/62.0	8.5/8.0/7.9	15.2/14.6/13.8
	macOS 26, M1 Pro	3.7/3.5/3.7	9.0/9.3/10.2	12.1/12.7/13.7	14.6/16.6/22.3	18.4/27.9/36.4	4.0/3.9/3.9	6.6/7.3/7.5

were mains-powered. Windows used the High Performance power plan, Wi-Fi adapter at Maximum Performance; macOS power-management interference was disabled. On Windows, MIDI virtual ports were created with Windows MIDI Services [29].

Wi-Fi tests used a shared 5 GHz access point (4.7–6.3% channel occupancy) with the headset’s low-latency Wi-Fi lock enabled. BLE-MIDI ran on both Macs. A radio-free Windows localhost reference ran the XR app alongside the companion, with wired Meta Horizon Link (USB).

Forward latency spans the XR application’s handoff of generated MIDI to the transport through the companion’s completed MIDI-output-port write. It excludes gesture processing, the pre-send frame wait (up to one frame), delivery into Live, and audio output. One-way latency cannot be measured directly across unsynchronized clocks; a USB clock alignment limits residual uncertainty to ± 0.16 –0.25 ms wirelessly and ± 0.013 –0.014 ms on localhost, not affecting single-device software timings or jitter.

Synthetic MIDI probes used exponentially distributed inter-message intervals. Mean rates of 4/24/200/2000 msg/s used 1000/5000/16 000/16 000 messages; 2000 msg/s was an excessive stress test. Each configuration ran at 72 and 120 fps (48 experiments in total). Latency statistics exclude the first five probes; loss counts include them. Table I covers 72 fps at 4/24/200 msg/s. Jitter comprises inter-quartile range (IQR) and Delay var. p95, the 95th percentile of absolute latency differences between consecutive messages.

B. Forward Latency and Reliability

Software transit. We timed the XR app’s send call on Unity’s main thread and companion reception-to-port-write processing. At 4–200 msg/s, the sum of their per-run medians was 11–51 μ s on macOS and 45–107 μ s on Windows for OSC, and 8–60 μ s for BLE: at most 7.1% of median wireless forward latency.

Transport comparison. At 72 fps and 4–200 msg/s, Wi-Fi forward-latency medians were 1.5–2.8 ms. At 4 msg/s, macOS p90 reached 81.9–92.3 ms, versus 4.6 ms on Windows; the macOS outliers followed idle gaps of approximately 205 ms or longer. Delay var. p95 was at most 4.8 ms except for sparse macOS traffic (100.8–108.2 ms). The radio-free localhost reference yielded medians of 0.20–0.25 ms and p99 of 0.29–0.52 ms at 4–200 msg/s.

The BLE central reported 7.5 ms connection intervals with LE 2M PHY on M1 Pro and 15 ms with LE 1M on Intel. Respective medians were 9.0–10.2 and 11.5–12.5 ms, with IQRs of 3.9–4.0 and 7.9–8.5 ms. No idle-related stall was observed; table extrema are observations, not guaranteed bounds.

Raising the display rate to 120 fps reduced Wi-Fi p90 by 0.7–1.1 ms at 200 msg/s; the sparse macOS stall persisted. Under stress, Wi-Fi medians were 3.8–6.9 ms at 72 fps; BLE p99 increased from 30–38 ms at 72 fps to 36–50 ms at 120 fps.

Delivery. Of 456 000 messages, 13 were lost over Wi-Fi after operating-system handoff. BLE and localhost lost none; received messages remained ordered. An additional Intel-Mac comparison at 18.1% versus 6.3% channel occupancy yielded 68 versus 7 losses per 44 000 messages at 4–200 msg/s; the busier session is excluded from the table and main totals.

C. Feedback and Synchronization

The companion immediately applied and answered XR commands over the same transport. Each round-trip run sent 1000 commands at random intervals averaging 4 msg/s, at 72 or 120 fps. Timing spanned the XR app’s send call to response reception in its main loop, including a frame-boundary wait but excluding Live’s processing.

All localhost responses returned in the next frame; on Windows Wi-Fi, 99.7% did at 72 fps and 96.6% at 120 fps. On macOS Wi-Fi, this share fell from 96–100% after idle gaps below 205 ms to 35–53% after longer gaps, with p90 of 92–101 ms. BLE never returned in the next frame; p99 was 51–56 ms on Intel and 42–43 ms on M1 Pro. One command went unanswered, over Wi-Fi at 72 fps.

Separately, companion-to-Live-to-companion control feedback had medians of 9–10 ms on macOS and 11–12 ms on Windows, using Live 12 with a 64-sample buffer. These control-feedback timings depend on Live’s configuration. Ableton Link synchronization was functionally verified against Ableton’s conformance test plan [30]; no synchronization-error bound is reported.

V. CONCLUSION

Our bidirectional MIDI bridge connects an XR instrument to a professional DAW, with local sequencer playback synchronized via Ableton Link. Timed software contributed a small fraction of wireless forward latency. OSC over Wi-Fi offered lower typical delay; BLE-MIDI avoided sparse-traffic stalls on the tested Macs.

REFERENCES

- [1] PatchXR, “PatchXR: Build your own virtual reality music instruments.” [Online]. Available: <https://patchxr.com>.
- [2] Really Interactive AB, “Virtuoso VR,” 2025. [Online]. Available: <https://virtuoso-vr.com>
- [3] Soundela, “Soundela — Mixed Reality Piano,” 2025. [Online]. Available: <https://soundela.com>
- [4] SynthVR, “SynthVR — Modular Synthesis in Virtual Reality,” 2025. [Online]. Available: <https://synthvr.com>
- [5] Tribe XR, “Tribe XR — VR DJ Academy,” 2025. [Online]. Available: <https://tribexr.com>
- [6] Gómez-Sirvent, José L., et al. “Musical instruments in extended reality: A systematic review.” *International Journal of Human-Computer Interaction* 41.16 (2025): 10143-10162.
- [7] D. Wessel and M. Wright, “Problems and prospects for intimate musical control of computers,” *Computer Music Journal*, vol. 26, no. 3, pp. 11–22, 2002.
- [8] MIDI Manufacturers Association, “MIDI 1.0 Detailed Specification,” 1996. [Online]. Available: <https://midi.org/midi-1-0-detailed-specification>
- [9] Ableton AG, “Ableton Link — Technology for syncing musical beat, tempo, and phase,” 2025. [Online]. Available: <https://www.ableton.com/link/>
- [10] Schörkhuber, Christian, Markus Zaunschirm, and Luca Turchet. “Zero-Delay Spatial Audio Rendering for Immersive Networked Music Performances.” 2025 IEEE 6th International Symposium on the Internet of Sounds (IS2). IEEE, 2025.
- [11] Rottondi, Cristina. “Inclusiveness in remote music teaching and networked music performances: vision, technological enablers and design strategies.” *IEEE Communications Magazine* 62.12 (2024): 34-40.
- [12] Turchet, Luca, et al. “Internet of musical things: Vision and challenges.” *Ieee access* 6 (2018): 61994-62017.
- [13] Buccellato, Pietro, Andrea Bianco, and Cristina Rottondi. “Soundy: A Multimodal Audio Interface for Educational Music Production.” *IEEE Access* (2025).
- [14] Turchet, Luca. “Musical Metaverse: vision, opportunities, and challenges.” *Personal and Ubiquitous Computing* 27.5 (2023): 1811-1827.
- [15] Giudici, Gregorio Andrea, et al. “Connecting MIDI Interfaces to the Musical Metaverse.” 2025 IEEE 6th International Symposium on the Internet of Sounds (IS2). IEEE, 2025.
- [16] S. Serafin, C. Erkut, J. Kojs, N. C. Nilsson, and R. Nordahl, “Virtual reality musical instruments: State of the art, design principles, and future directions,” *Computer Music Journal*, vol. 40, no. 3, pp. 22–40, 2016.
- [17] K. Shoji, “MIDI Plugin for Mobile and Desktop,” Unity Asset Store, 2025. [Online]. Available: <https://assetstore.unity.com/packages/tools/audio/midi-plugin-for-mobile-and-desktop-198917>
- [18] T. Bachmann, “Maestro — MIDI Player Tool Kit Pro,” Unity Asset Store, 2025. [Online]. Available: <https://assetstore.unity.com/packages/tools/audio/maestro-midi-player-tool-kit-pro-115331>
- [19] M. Wright, A. Freed, and A. Momeni, “OpenSound Control: State of the art 2003,” in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Montreal, Canada, 2003.
- [20] MIDI Manufacturers Association, “MIDI over Bluetooth Low Energy (BLE-MIDI) Specification v1.0,” 2015.
- [21] J. Lazzaro and J. Wawrzynek, “An RTP payload format for MIDI,” RFC 4695, 2006. [Online]. Available: <https://tools.ietf.org/html/rfc4695>
- [22] MIDI Manufacturers Association, “Network MIDI 2.0 Specification,” 2020. [Online]. Available: <https://midi.org/network-midi-2-0>
- [23] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, “An overview on networked music performance technologies,” *IEEE Access*, vol. 4, pp. 8823–8843, 2016.
- [24] Boem, Alberto, Matteo Tomasetti, and Luca Turchet. “Issues and challenges of audio technologies for the musical metaverse.” *J. Audio Eng. Soc* 73.3 (2025): 94-114.
- [25] Boem, Alberto, et al. “A Decoupled VR and Real-time Audio System for Distributed Musical Collaboration.” 2025 IEEE 6th International Symposium on the Internet of Sounds (IS2). IEEE, 2025.
- [26] J. Wang, A. Mulder, and M. M. Wanderley, “Practical considerations for MIDI over Bluetooth Low Energy as a wireless interface,” in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Porto Alegre, Brazil, 2019.
- [27] Meta Platforms, Inc., “Meta XR All-in-One SDK v78.0,” 2025. [Online]. Available: <https://developer.oculus.com/documentation/unity/>
- [28] JUCE Team, “JUCE v8.0.4 — Cross-platform C++ framework,” 2025. [Online]. Available: <https://juce.com>
- [29] Microsoft Corporation, “Windows MIDI Services,” 2025. [Online]. Available: <https://github.com/microsoft/MIDI>
- [30] Ableton AG, “Test plan for Link-enabled apps.” [Online]. Available: <https://github.com/Ableton/link/blob/master/TEST-PLAN.md>