2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #53066

Scalable Proctoring for Concurrent VR Training: A WebSocket-Orchestrated Framework for Multimodal Data Streaming

Nicholas John Allegretti, University of Missouri - Columbia
Fang Wang, University of Missouri - Columbia

Fang Wang is an Associate Teaching Professor in the Engineering and Information Technology Department. Her research interests include the design, evaluation, and application of virtual and augmented reality, game-based and mobile technologies for healthcare and education, and computational modeling and simulation in engineering. Her research has been funded by the National Science Foundation and the Coulter Biomedical Accelerator Program.

Scalable Proctoring for Concurrent VR Training: A WebSocket-Orchestrated Framework for Multimodal Data Streaming

Abstract

Virtual reality (VR) training simulations are increasingly used in high-consequence settings such as semiconductor cleanroom operations. Although VR reduces the cost and risk of early procedural practice, scaling instruction remains difficult because proctors must continuously monitor learner progress, interpret where attention is directed, and intervene when errors or confusion occur. This challenge becomes more acute when one instructor supervises multiple learners at the same time.

This paper presents a WebSocket-orchestrated framework for streaming multimodal data from multiple concurrent VR learners to a browser-based proctor dashboard without requiring the proctor to enter the headset experience. The framework streams egocentric video, gaze-derived fixation events, and step-level training-log events so that an instructor can view learner context, attention, and progress from a single interface.

We focus on the streaming architecture, runtime model, and scalability of the orchestration layer. Using synthetic clients and two RTX 3090-class machines, the system maintained sub-100 ms p95 round-trip latency through 32 concurrent learners under baseline gaze and training-log traffic (37 ms gaze and 70 ms training-log RTT at the highest baseline rate). Adding Meta Quest-like video streaming (480x270 JPEG at 8 fps) increased p95 latency only modestly (64–65 ms for gaze/training-log RTT and 35.7 ms for video ACK RTT). Continuous 10 Hz audio streaming represents a worst-case stress condition and can dominate the server event loop at high concurrency, motivating batching, voice-activity gating, and selective worker offload for larger deployments.

1 Introduction

Immersive VR simulations are an efficient and practical method of teaching complex laboratory and manufacturing procedures. This is in part due to their ability to provide repeatable practice of procedures with reduced risk, cost and facility constraints. For example, recent work in semiconductor education has explored VR-based microfabrication training to help learners familiarize themselves with photolithography and cleanroom operations prior to entering the physical facility through clearly outlined instructions [1]. However, the presence of instructional material inside a VR simulation does not eliminate the need for proctor guidance. Learners come from a variety of backgrounds and have different levels of understanding the procedures within the simulation, as well as operating a VR device in general. This can lead to novice learners experiencing more uncertainty, equipment handling errors, and misinterpretations of visual cues.

In many training contexts, proctors operate as cognitive apprenticeships, modeling expert prac-

tice, observing learner actions, and providing scaffolding and coaching as learners build competence [2]. In VR, the proctor's limited visibility into the learner's view and attention hinders their ability to mentor effectively. This issue is even more prevalent when the proctor must assist multiple learners, infer who needs help, and deliver intervention in a manner that does not disrupt autonomy or overwhelm the learner with unsolicited instruction.

This paper addresses the scalability gap between single-learner and multi-learner VR proctoring. We present a browser-based dashboard and WebSocket orchestration layer that externalize the key observables a proctor needs: what the learner sees, where the learner is looking, and which procedural step the learner has reached. The emphasis of this paper is the streaming substrate and its scalability rather than automated assistance or pedagogical outcome measurement. Our framing is informed by multimodal learning analytics (MMLA), but we use MMLA here only to motivate why video, gaze, and progress logs should be streamed together [3].

The contributions of this paper are: (1) a low-latency WebSocket orchestration service and typed message protocol for streaming synchronized VR telemetry from multiple concurrent learners; (2) a browser-based proctor dashboard that supports rapid learner assessment without headset immersion; and (3) an experimental characterization of scalability and robustness across 1–32 concurrent clients, including baseline gaze/training-log traffic, video overhead, audio stress conditions, and worker failover.

The remainder of this paper is organized as follows. Section 2 summarizes existing related work on VR instruction, instructor dashboard systems, and multimodal sensing. Section 3 describes the motivating use case and design requirements. Sections 4 and 5 explain the proposed architecture, telemetry protocol, and implementation. Section 6 presents the scalability experiment design and results using synthetic clients. Section 7 discusses implications, limitations, and ethical considerations. Lastly, Section 8 concludes and outlines future work.

2 Background and related work

This system sits at the intersection of immersive VR training, instructor monitoring interfaces, and multimodal sensing. To show the motivation of our design we briefly review each of these topics.

2.1 VR procedural training and instructor support

An ever-expanding body of research utilizes immersive environments as a platform for skill acquisition in engineering and laboratory settings, including recent cleanroom microfabrication training systems which integrate guided instruction and demonstrate improvements in procedural knowledge [1]. While many VR training systems are designed for individual practice, institutional onboarding often involves numerous learners requiring multiple-hour sessions. With limited proctor availability this becomes a scaling challenge.

2.2 Dashboards and instructor awareness

A common goal of learning analytics dashboards is to capture and visualize traces of learning activities in order to promote awareness, reflection, and sense-making [6]. In VR settings, learner monitoring is further complicated by limited access to learner context inside the headset experience. Previous work has explored interfaces that surface student status signals to instructors, including visual indicators intended to help a VR-immersed teacher monitor student actions and attention in a multi-student virtual classroom [7]. Other classroom-related VR systems have also utilized dashboards that visualize learner eye gaze to support remote teaching and attention management [8]. These existing studies show the value of externalized attention signals for reducing instructor workload.

2.3 Multimodal learning analytics and eye tracking

MMLA argues that learning-relevant constructs often cannot be inferred from a single data stream, and that multimodal capture can mitigate the "streetlight effect" of only analyzing what is easy to log in computer-mediated systems [3]. Eye tracking functionality is becoming increasingly available in consumer HMD displays, enabling attention estimation and fine-grained behavioral measures in VR [4]. However, raw gaze points are difficult to interpret at scale and consequently higher-level summaries are often needed to support instructor awareness.

2.4 Gaze entropy for attention dispersion and efficiency

Entropy-based gaze metrics have been proposed as a compact way to describe the distribution and transitions of visual attention. A comprehensive review distinguishes stationary gaze entropy (dispersion across regions) and gaze transition entropy (unpredictability of scanpath transitions) and discusses how these measures can relate to visual scanning efficiency and task demands [5]. These ideas suggest that interpretable summaries of attention, such as entropy, dwell, and switching rates, could eventually support instructor triage. In the present paper, however, we focus on the underlying streaming and monitoring substrate rather than validating derived analytics.

3 Motivating use case and design requirements

Our toolset was developed to support proctoring multiple learners in procedural VR modules for semiconductor cleanroom onboarding. In the motivating scenario, learners complete ordered task steps in a headset while interacting with virtual tools, equipment, and highlighted scene objects. The proctor supervises these learners from a browser dashboard rather than from inside VR. This context matters because learners may become stuck, misread a visual cue, or advance through the wrong procedural step even when instructional content is already embedded in the simulation.

In our target deployment, each learner runs a Unity-based VR module on a standalone Meta

Quest Pro with eye tracking. The module streams three core data channels used throughout this paper: egocentric video, gaze fixation events, and step-level training-log events. A training-log event is a discrete procedural status message such as step start, step completion, or error. One or more proctors supervise the active sessions from a browser-based proctor panel on the local network.

We refer to these seven design requirements as Requirement 1 through Requirement 7.

- **Requirement 1 (Concurrent oversight):** Must support simultaneous monitoring of multiple learners without forcing the proctor to join VR sessions.
- **Requirement 2 (Fast learner assessment):** The system must support proctors in quickly identifying which learners need attention and why through the use of interpretable indicators.
- **Requirement 3 (Context fidelity):** The system must expose what the learner sees and the learner's attention to reduce ambiguity in diagnosing issues.
- **Requirement 4 (Low-latency intervention):** Must allow timely proctor actions (e.g. speak, nudge, or advance scenario state) with minimal session disruption.
- **Requirement 5 (Learner autonomy):** The system should prefer learner-initiated help over unsolicited interruptions. The assistance provided should be minimal yet actionable.
- **Requirement 6 (Scalability and robustness):** The system must allow computationally heavy components to scale with hardware availability and degrade gracefully if workers are unavailable.
- **Requirement 7 (Privacy-aware deployment):** The system must support local, on-premises processing so that sensitive video, audio, and behavioral data do not leave the facility.

4 System architecture and telemetry protocol

The proposed system is an event-driven multimodal VR proctoring pipeline organized around a central WebSocket orchestration service. VR client devices stream telemetry to the orchestrator through a lightweight bus module. The orchestrator maintains per-learner session state, optionally routes selected workloads to worker devices, and broadcasts the relevant views to the proctor dashboard. In the baseline configuration analyzed in this paper, the streamed data include egocentric video, gaze events, training-log events, and optional audio. Figure 1 summarizes this data flow, and Table 1 lists the primary message types used to move these data through the system. Egocentric video provides learner context, gaze fixations summarize attention, and training-log events capture procedural progress over time.

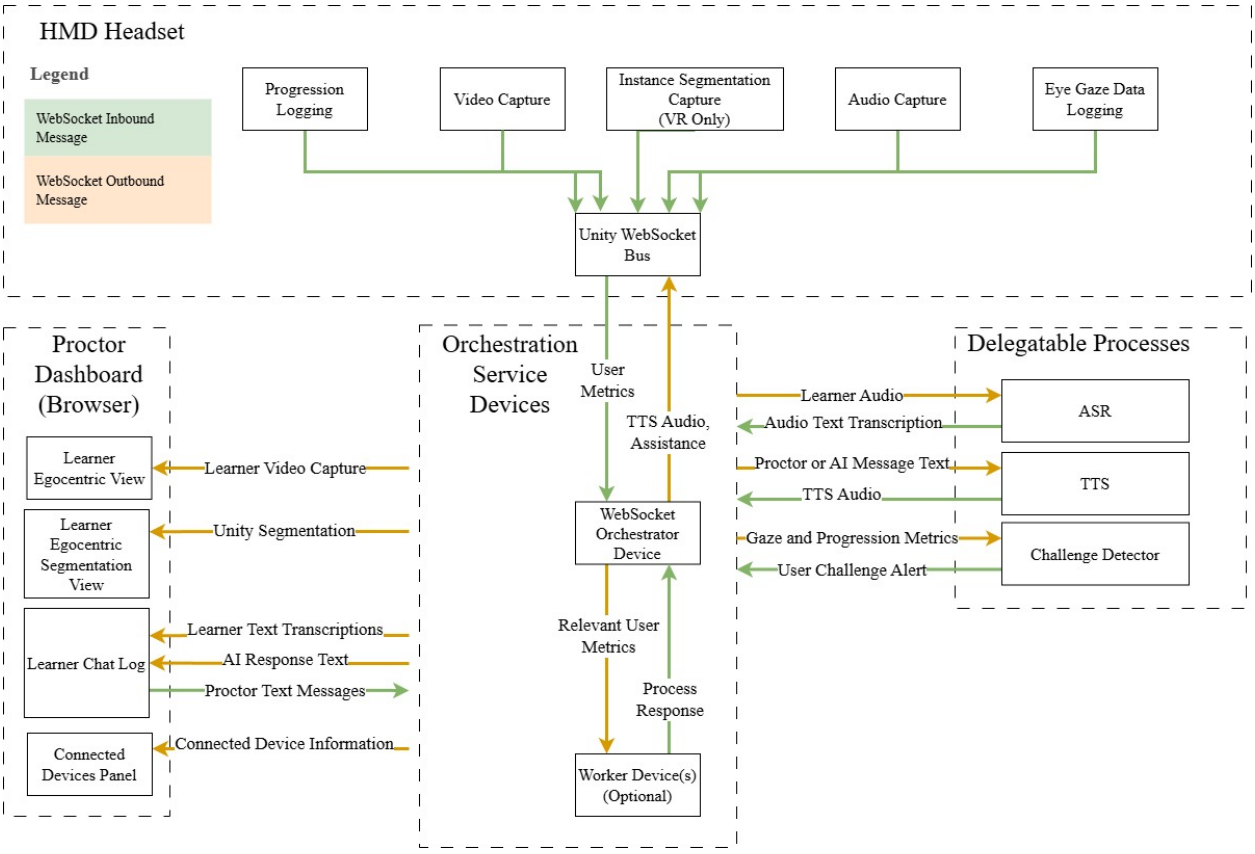

Figure 1: High-level system architecture and data flow.

4.1 Unified WebSocket message model

All components of the system communicate using a typed JSON message envelope with a small set of common fields (e.g., message type, session identifier, and timestamps). Binary artifacts, such as JPEG video frames and PCM16 audio chunks, are carried through the same channel by base64-encoding the payload and attaching modality-specific metadata such as frame index or sample rate. This simplifies deployment while still allowing heterogeneous telemetry to remain synchronized. Figure 1 shows how those messages flow through the architecture, and Table 1 summarizes the message types that are most important to the VR streaming workflow described in this paper.

4.2 Session lifecycle

When a VR client first connects to the orchestrator, it sends a “hello” handshake message announcing the device and its intended module. The orchestrator then assigns or confirms a session identifier, initializes the per-learner state, and begins accepting telemetry. Proctor clients connect separately through the hosted web application and subscribe to the set of active sessions. Proctor actions such as advancing a scenario step are sent as commands to the orchestrator, which then forwards them to the relevant VR client.

4.3 Per-learner runtime state

For each active learner client, the orchestrator maintains a compact in-memory state object. This includes the most recent video frame, recent gaze events, the current training step, and optional modality buffers when audio or segmentation are enabled. This state serves two purposes. First, it supports rapid rendering of live views in the proctor dashboard. Second, it preserves enough recent context for the proctor to interpret what a learner is doing without joining the headset session directly.

Table 1: Primary telemetry and control messages used in the system.

Modality / Control	Message type	Purpose	Key payload fields
System	hello	Session handshake and metadata	device_id, module_name, session_id
Video	video_frame	Stream learner egocentric view	base64 JPEG, frame_idx, ts
Gaze	gaze_event	Stream fixation and attention-target events	target_label, duration_ms, reliability
Progress	training_log	Stream step-level progress and event updates	step_id, event_type, success / fail

Modality / Control	Message type	Purpose	Key payload fields
Audio in	audio_chunk	Stream optional microphone audio	base64 PCM16, sample_rate, ts
Segmentation (optional)	seg_frame	Stream optional visual overlay data	base64 PNG, frame_idx, ts
Segmentation (optional)	seg_legend	Provide ID-to-label and color mapping	entries[{id, label, color_hex}]
Proctor control	skip_step	Advance scenario to maintain flow	session_id, step_id

5 Implementation and methodology

This section summarizes the concrete implementation of the toolset and the design choices that enable low-latency multi-learner proctoring. The implementation is organized around two layers: a Unity VR client instrumentation layer and a Python orchestration server with optional worker processes for selected high-overhead tasks.

5.1 Orchestration server and runtime model

The orchestration service is implemented primarily in Python. It uses an asynchronous event loop to concurrently manage WebSocket connections from multiple VR clients, worker devices, and the proctor dashboard. Each connected learner is represented by a per-device state object that stores the latest video frame, recent gaze events, and training progression. Messages received from the VR clients are normalized into internal events, update the relevant state object, and are then selectively broadcast to dashboard subscribers.

The orchestration service is designed to run on standard workstation hardware and to support local, privacy-preserving deployment. When additional resources are available, selected higher-overhead processing stages can be routed to one or more worker machines connected to the orchestrator. If a worker becomes unavailable, the system can continue operating with reduced capability rather than dropping the learner session entirely. In the present paper, the main concern is how this design affects message transport and proctor visibility under load.

5.2 VR client telemetry instrumentation

The Unity VR client includes modular streamers for the primary telemetry channels, allowing each modality to be enabled or disabled per module. The WebSocket bus is instantiated in the main menu scene and persists across scenes, giving each module a common interface for emitting progress logs and contextual events. Gaze data are derived from eye-to-object raycasts against

tagged scene objects. The client also captures the learner's egocentric view for video streaming. When additional visual overlays are needed, the system can stream corresponding segmentation artifacts, but those overlays are not the focus of the present paper.

5.3 Real-time cognitive monitoring via struggle detection

To reduce proctor burden, the orchestration server computes interpretable struggle indicators over rolling windows of VR client telemetry. The current implementation uses a hybrid of progress- and gaze-based triggers:

- **Progress stagnation (no-success timeout):** If no successful progress events are observed for a configurable interval (e.g., 90 seconds), emit a struggle event.
- **Long dwell on a single target:** If a fixation remains on one labeled target beyond a dwell threshold (e.g., 10 seconds), emit a struggle event.
- **Rapid switching / volatility:** If attention target transitions exceed a short-horizon threshold (e.g., >18 label switches within 10 seconds), emit a struggle event.
- **High gaze entropy:** If the entropy over recent attention targets exceeds a threshold (e.g., 1.6 bits over a 15-fixation history), emit a struggle event.

To prevent alert fatigue, struggle events are rate-limited to one alert per learner every 20 seconds, although this is configurable. Each alert includes a compact summary of the recent context including the current step, recent gaze targets, and timestamps. These summaries are intended to support rapid proctor response.

5.4 Proctor dashboard and intervention controls

The proctor dashboard subscribes to the set of active sessions and renders a cohort overview with per-learner status, along with individual learner views showing live egocentric video, step progress, and recent gaze targets. This workflow supports monitoring many learners in parallel while reducing the need for continuous first-person observation. The same orchestration service supports a lightweight control plane. Proctors can issue scenario control actions such as skip_step to advance a learner past non-instructional blockers while preserving training flow.

Figure 2 shows the proctor dashboard used to monitor concurrent learners. The interface presents the connected-device list, egocentric video, recent gaze information, and progress status so the proctor can quickly determine who needs attention and why.

Figure 2: Proctor dashboard used for multi-learner supervision, including the connected-device list, egocentric view, recent gaze information, and progress status.

6 Experiments and results

6.1 Experimental setup

We evaluated the scalability of the WebSocket orchestration server using synthetic VR clients (no human subjects). Synthetic clients implement the same protocol handshake and telemetry message types used by the Unity client, including client_hello/session_start and the core telemetry streams gaze_event and training_log with corresponding server ACKs. In this section, training-log traffic refers to discrete procedural progress messages emitted by the VR module, such as step start, step completion, and error events. Optional audio and video paths were exercised using the same message formats as the deployed system, with ACK timing measured when enabled.

Runs were executed on Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz, GeForce RTX 3090, 128GB 3200MHz RAM, Windows 10, and Python 3.11.14. Unless otherwise noted, synthetic VR clients were run on the same host as the orchestration server and connected via local loopback (`ws://127.0.0.1:8765`) to isolate server-side orchestration overhead from network variability. Worker processes ran on a second RTX 3090 host and communicated with the server over the local network. Video payload experiments additionally ran synthetic clients on the second host over the local network so that the measurements included JPEG payload transmission. Each condition swept N = 1, 2, 4, 8, 16, 24, and 32 concurrent clients with a 5 second warmup, a 30 s measurement window, and 3 repetitions per N (unless otherwise specified for single-N tests). Metrics were measured on the client side using round-trip time (RTT) from send timestamp to receipt of the corresponding server ACK.

6.2 Workload profiles

We used three baseline telemetry loads and modality add-on profiles:

- **Low:** gaze 5 Hz, training-log 1 Hz
- **Medium:** gaze 10 Hz, training-log 2 Hz
- **High:** gaze 20 Hz, training-log 4 Hz
- **Audio (local processing baseline):** medium baseline + audio chunks 10 Hz (ACKs measured on start/end). The deployed client ordinarily transmits audio only when speech is detected, so this profile represents an upper-bound stress case in which all learners are continuously transmitting.
- **Audio (worker offload):** the same workload with speech processing routed to a worker when available.
- **Video (Meta Quest-like):** medium baseline + 480x270 JPEG frames at 8 fps (quality 45, average frame size approximately 14 KB).
- **Video (stress point):** medium baseline + 1280x720 JPEG frames at 15 fps (quality 45, average frame size approximately 54 KB), evaluated up to N = 16.

6.3 Metrics

For each ACK type, we report median and p95 RTT and throughput in messages per second. Throughput is computed as the number of ACKs observed divided by the measured duration. For failover behavior, we additionally analyze per-second RTT summaries to visualize transient degradation and recovery.

6.4 Core scalability (gaze + training-log)

Across the low, medium, and high baseline profiles (gaze + training-log only), throughput scaled linearly with N and closely matched the configured per-client rates. At N = 32, observed throughput was approximately 160.1 gaze msg/s and 32.6 training-log msg/s (low), 320.7 gaze msg/s and 64.3 training-log msg/s (medium), and 640.6 gaze msg/s and 128.2 training-log msg/s (high). Latency increased gradually with N. At N = 32, p95 RTTs were 14 ms (gaze) and 13 ms (training-log) for the low profile, 17 ms (gaze) and 35 ms (training-log) for the medium profile, and 37 ms (gaze) and 70 ms (training-log) for the high profile. These results indicate that the core event loop maintains sub-100 ms p95 latency at 32 concurrent learners even under the high baseline rate.

Table 2 summarizes the N = 32 baseline results, while Figures 3 and 4 show how latency and throughput vary with concurrency under the high baseline profile. Together, these results show gradual RTT growth and near-linear throughput scaling as the number of learners increases.

Figure 3 shows how p95 RTT changes with concurrency under the high baseline profile (20 Hz gaze and 4 Hz training-log traffic per client). As the number of concurrent learners increases, both curves rise gradually rather than abruptly, which indicates that the orchestration layer degrades predictably under heavier baseline load. The training_ack curve increases more sharply than the gaze_ack curve at higher client counts, suggesting that training-log processing is somewhat more sensitive to congestion and queueing overhead in this configuration.

Figure 4 complements the latency result by showing throughput under the same high baseline profile. Throughput increases nearly linearly with the number of concurrent learners for both gaze and training-log ACKs, which indicates that the orchestration service continues to process messages at the expected aggregate rate as load grows. This is consistent with the configured per-client message rates: each additional learner adds a fixed amount of gaze and training-log traffic, and the measured throughput tracks that increase closely.

Table 2: Baseline scalability results at $N = 32$ (throughput and p95 RTT) for the low, medium, and high profiles.

Profile	Gaze (Hz)	Training-log (Hz)	Gaze msg/s	Training-log msg/s	Gaze p95 (ms)	Training-log p95 (ms)
Low	5	1	160.1	32.6	14.0	13.0
Medium	10	2	320.7	64.3	17.0	35.0
High	20	4	640.6	128.2	37.0	70.0

6.5 Modality overhead: audio and video

6.5.1 Realistic video streaming (Meta Quest-like profile)

Using a Meta Quest-like profile (480x270 JPEG at 8 fps, quality 45, average frame size 14,265 bytes), the orchestration server maintained stable latency up to N = 32. At N = 32, p95 RTTs were 64.0 ms for gaze_ack, 65.0 ms for training_ack, and 35.7 ms for video_ack, while throughput closely matched configured rates (approximately 321 gaze ACK/s, 64 training-log ACK/s, and 257 video ACK/s). Based on payload size, this corresponds to an estimated per-client video payload rate of approximately 0.91 Mbps raw (approximately 1.22 Mbps with base64 expansion), or approximately 39 Mbps aggregate at N = 32 (base64 payload only).

We also evaluated a stress point at 1280x720 JPEG at 15 fps (quality 45, average frame size 54,140 bytes) up to N = 16. At N = 16, p95 RTTs remained stable: 64.3 ms (gaze_ack), 59.9 ms (training_ack), and 34.0 ms (video_ack). This stress point corresponds to approximately 6.5

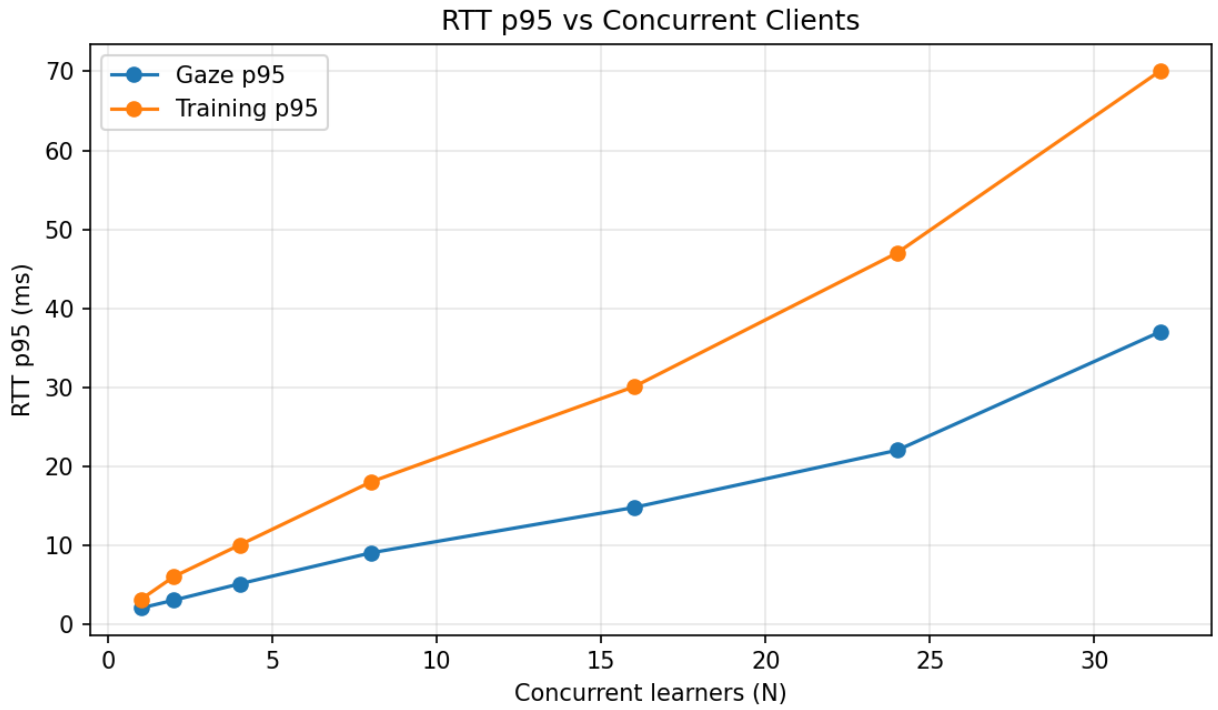

Figure 3: RTT p95 vs concurrent clients for gaze_ack and training_ack under the high baseline profile (20 Hz gaze, 4 Hz training-log).

Mbps raw per client (approximately 8.7 Mbps base64), or approximately 139 Mbps aggregate at N = 16 (base64 payload only).

Figure 5 shows that adding Quest-like video increases gaze and training-log RTT only modestly relative to the gaze-plus-training-log baseline, even at N = 32. The curves remain comparatively flat as concurrency increases, and the video_ack p95 stays lower than the corresponding gaze and training-log RTTs. In practical terms, Figure 5 indicates that periodic egocentric video can be added to the proctor dashboard without fundamentally changing the low-latency behavior of the orchestration layer under the tested local-network conditions.

6.5.2 Audio streaming (local processing baseline)

Audio streaming has a pronounced effect on latency at higher concurrency when audio handling includes local processing in the orchestration path. Under the audio profile (medium baseline + audio chunks 10 Hz), p95 RTTs at N = 32 increased to approximately 2.34 s (gaze_ack) and 2.44 s (training_ack), while audio_ack_end p95 was 324 ms (audio_ack_start p95 was 1.86 s). This behavior indicates that continuous audio can dominate the server loop at scale when compute-heavy and/or blocking operations occur in the orchestration host's hot path. In typical deployments, audio is gated to speech segments by the VR client. We nevertheless evaluate continuous concurrent audio to characterize an upper-bound load condition.

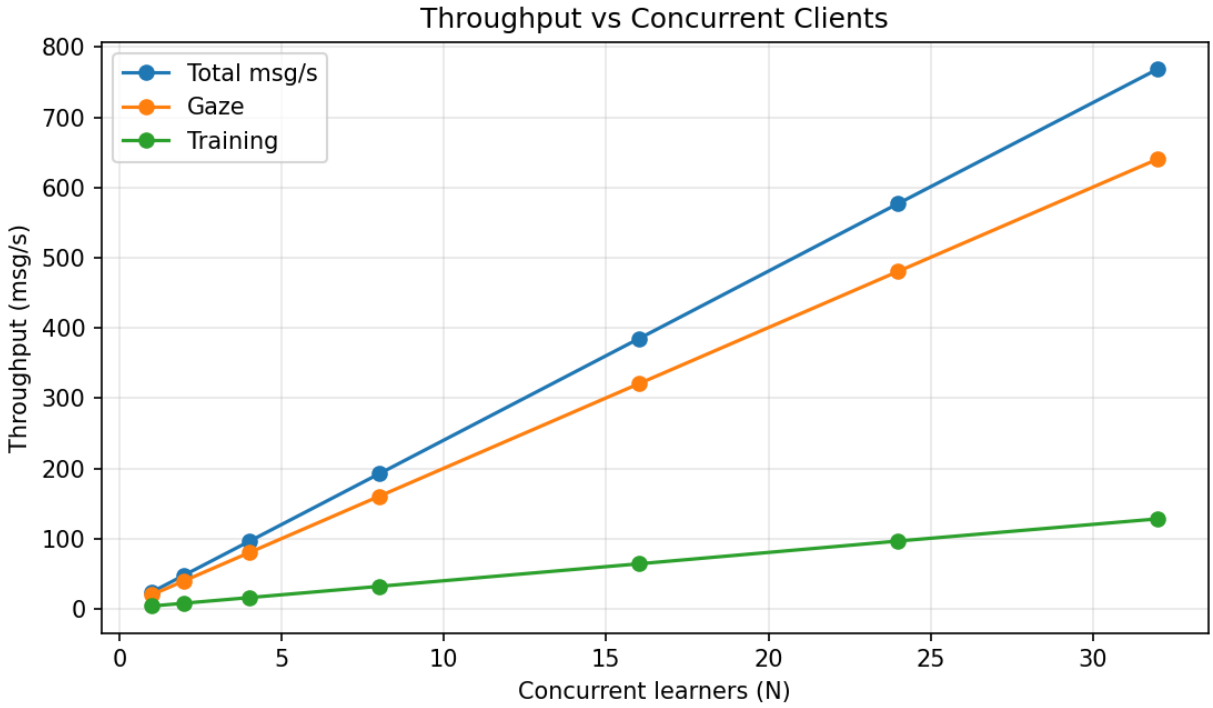

Figure 4: Throughput vs concurrent clients under the high baseline profile (20 Hz gaze, 4 Hz training-log).

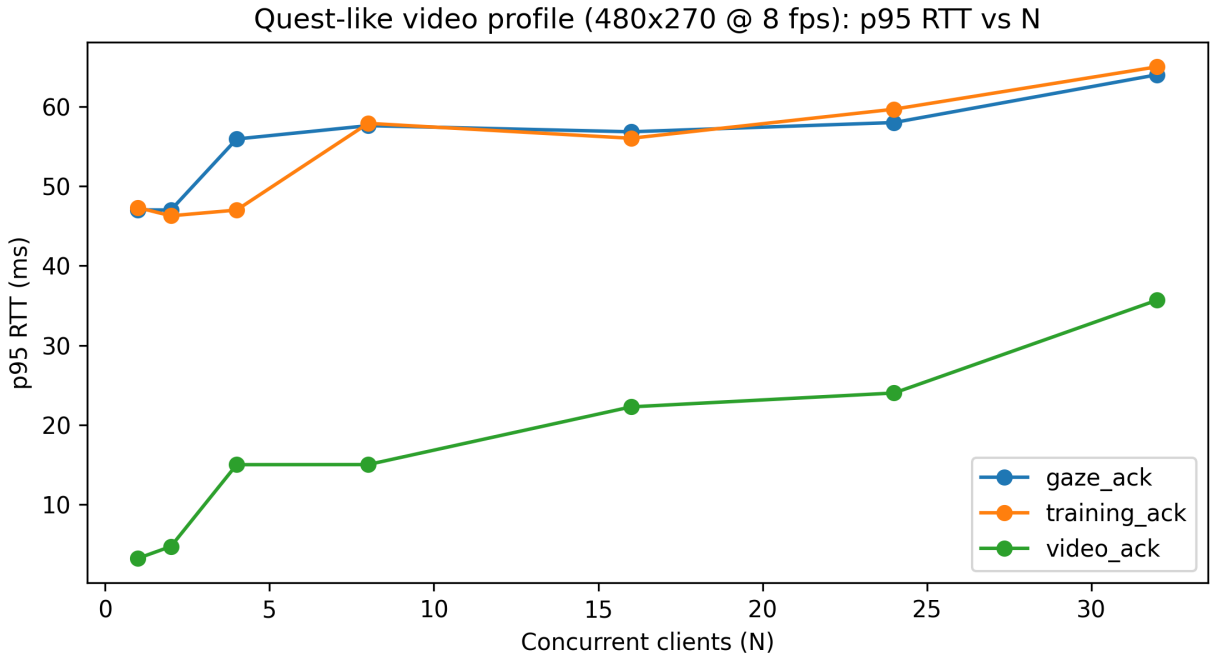

Figure 5: RTT p95 vs concurrent clients for gaze_ack, training_ack, and video_ack under the Quest-like video profile (480x270 @ 8 fps).

6.5.3 Audio mitigation via worker offload

To evaluate scalability and robustness under audio load, we repeated the audio profile with speech processing routed to a worker when available. In the tested configuration, worker offload preserved low latency at N = 24: gaze_ack p95 was 18.0 ms, training_ack p95 was 16.0 ms, and audio_ack_end p95 was 20.0 ms. At N = 32, however, gaze and training-log p95 RTTs increased to multi-second levels (approximately 2.16 s and 2.17 s, respectively). These results should therefore be interpreted as improved performance at 24 concurrent clients in the tested setup, not as evidence of a hard 24-user system limit.

Table 3 compares the main add-on conditions and shows that audio, not video, is the dominant scalability stressor in the tested configurations.

Table 3: Selected modality add-on results.

Condition	N	Add-on	Gaze p95 (ms)	Training-log p95 (ms)	A/V p95 (ms)
Video (Quest-like)	32	480x270 JPEG @ 8 fps (q=45)	64	65	video_ack 36
Audio (local baseline)	32	Audio 10 Hz (start/end ACK)	2337	2441	audio_end 324 (audio_start 1862)
Audio (worker offload)	24	Audio 10 Hz + worker speech processing	18	16	audio_end 20
Audio (worker offload)	32	Audio 10 Hz + worker speech processing	2162	2172	audio_end 181

6.5.4 Audio bottleneck decomposition (local vs offload; disk I/O on/off)

To better isolate which factors drive the audio-induced latency increase, we ran a four-condition decomposition: local vs worker multiplied by disk I/O on vs off. Results show that (1) local audio handling produces multi-second p95 RTT at N >= 24 regardless of disk I/O toggling, and (2) worker offload restores low-latency operation at N = 24, but (3) at N = 32, gaze/training-log p95 can still rise to approximately 2 s even with worker offload. This third result does not imply a hard 24-user system limit. Rather, it indicates that under the specific worst-case condition of continuous 10 Hz audio streaming from all 32 learners, a single-worker offload strategy is not sufficient to preserve low latency on the tested hardware. Baseline gaze/training-log traffic and the Quest-like video condition both remain responsive at N = 32, so the degradation is modality-dependent rather than a global concurrency ceiling.

Table 4 shows that worker offload reduces the N = 24 case from roughly 1.8 s p95 RTT to approximately 20 ms, but the N = 32 case remains near 2 s regardless of whether disk I/O is enabled. This pattern suggests that, at N = 32 under continuous audio load, the dominant bottleneck is no longer speech-processing alone; instead, queueing, per-message overhead, and orchestration-

path contention likely become the limiting factors. In practical terms, the current system supports 32 concurrent learners for baseline telemetry and realistic video, but only 24 concurrent learners for low-latency operation under the tested continuous-audio stress case. Supporting 32 learners under that stress condition would likely require additional worker capacity, lower audio duty cycle, batching, or stronger backpressure mechanisms.

Table 4: Audio decomposition results (p95 RTT, ms).

Condition	N=24 Gaze p95	N=24 Training-log p95	N=32 Gaze p95	N=32 Training-log p95
Local speech + disk on	1831	1858	2159	2331
Local speech + disk off	1781	1816	2110	2054
Worker speech + disk on	21.0	15.1	1993	2039
Worker speech + disk off	20.0	16.0	1992	1899

6.5.5 Audio message-rate sensitivity (constant bitrate sweep at N=32)

To isolate the effect of message-rate and batching separate from changes in total audio throughput, we ran a constant-bitrate sweep at N = 32 holding audio throughput approximately constant while varying the chunk rate: 10 Hz x 3200 B, 5 Hz x 6400 B, and 2 Hz x 16000 B (approximately 32,000 B/s/client in all cases). Across this range, orchestration RTT was not materially sensitive to chunk rate: gaze_ack and training_ack p95 remained in the 20–26 ms range for all three conditions.

Table 5 shows that, at fixed bitrate, changing the audio chunk rate from 2 to 10 Hz has little effect on gaze or training-log RTT.

Table 5: Audio message-rate sensitivity at N=32 (constant bitrate).

Audio rate (Hz)	Chunk bytes	Gaze p95 (ms)	Training-log p95 (ms)	audio_end p95 (ms)
10	3200	20.0	24.0	18.0
5	6400	22.0	26.0	20.0
2	16000	25.0	23.0	20.0

6.5.6 Worker failover behavior (robustness)

Finally, we evaluated robustness under worker failure using a single-N run (N = 32, 60 s measurement) with worker offload enabled, then terminating the worker mid-run. Over the full run, overall p95 RTT rose to 6.03 s (gaze_ack) and 6.16 s (training_ack), dominated by the transient disruption window. A per-second analysis shows stable low RTT at the beginning of the run (typical per-second medians 8–10 ms), followed by a multi-second degradation window immediately

after worker termination (per-second medians peaking at approximately 7.5 s), and subsequent recovery to a slower but stable regime (approximately 150–180 ms medians near the end). This demonstrates that the system continues operating under worker loss, but failover is not seamless and can produce a noticeable recovery window.

Figure 6 illustrates the transient latency spike immediately after worker termination and the slower but stable regime that follows recovery.

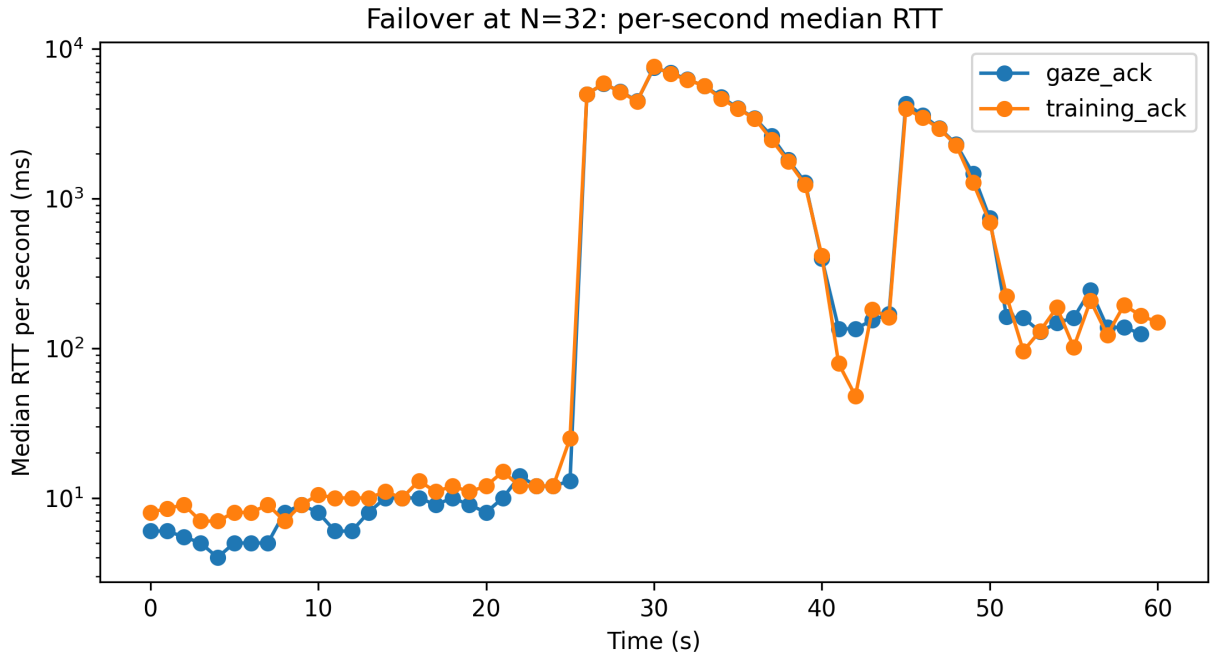

Figure 6: Failover time series at N=32: per-second median RTT for gaze_ack and training_ack during worker termination and recovery.

6.6 Resource utilization

Resource logs sampled the server process once per second during each run. Baseline profiles remained CPU-light at N = 32 (average CPU 4.6% low, 7.8% medium, and 14.8% high), consistent with the low-tens-of-milliseconds RTTs observed. In the local audio profile, average CPU increased to approximately 94% at N = 24 and 98% at N = 32, consistent with the sharp increase in RTT observed under local audio handling. These results support the architectural claim that compute-heavy and/or blocking modalities are primary scalability bottlenecks and motivate worker offload and adaptive load management for larger learner groups.

6.7 Summary and implications

Overall, the orchestration server scales linearly in throughput and maintains low tens-of-milliseconds p95 RTT for gaze and training-log telemetry under the low, medium, and high baseline profiles up to 32 concurrent clients. With realistic video payloads (Meta Quest-like 480x270 at 8 fps), the system maintained stable latency at N = 32, indicating that periodic dashboard video updates are feasible within this WebSocket-based message model. In contrast, continuous audio streaming can substantially increase RTT and CPU load at higher concurrency, while selective worker offload improves the 24-client case but does not eliminate the N = 32 degradation observed in the present configuration. These results motivate future work on backpressure, adaptive modality throttling, and improved failover behavior for larger deployments.

7 Discussion, limitations, and ethical considerations

Our results suggest that a WebSocket-orchestrated proctoring workflow can shift instructor attention from continuous first-person observation to triage. By externalizing learner viewpoint, attention targets, and step progress into a browser-based dashboard, a proctor can monitor many learners in parallel. From a systems perspective, the experiments show that the core streaming substrate remains responsive through 32 concurrent clients for gaze and training-log traffic, while Quest-like video adds only modest overhead. This supports the claim that multimodal VR data streaming can provide practical proctor awareness at cohort scale.

Audio is the primary scaling risk among the modalities examined here. Continuous 10 Hz audio chunks represent a worst-case stress condition and can dominate the server event loop at high concurrency. In practice, voice-activity gating reduces the probability that all learners stream simultaneously. Nevertheless, large-cohort deployments benefit from architectural mitigations such as (a) offloading selected high-overhead tasks to worker devices, (b) batching audio samples to reduce per-message overhead, and (c) explicit backpressure or admission control when compute resources are saturated. The worker failover experiment further shows that loss of a worker does not halt operation but can cause a transient multi-second recovery window, motivating more robust degradation and recovery policies.

This paper focuses on the streaming substrate and performance characterization; it does not evaluate pedagogical effectiveness or learner outcomes. Future user studies can examine how instructors interpret gaze and progress signals, how different procedures change the relative value of the streamed modalities, and what dashboard designs best support intervention at cohort scale.

Privacy and ethics are central because egocentric video, audio, and eye-tracking streams can be sensitive. The architecture supports local, on-premises processing, but deployments should define informed consent, retention policies, and role-based access controls for who can view and export telemetry. Technically, the current implementation assumes a reliable local network and uses base64 encoding, which increases payload size. Future versions could use binary WebSocket frames or alternate transport mechanisms for large payloads while retaining the simplicity

of the current control plane.

8 Conclusions and future work

VR training can reduce cost and risk in engineering education, but scaling instruction requires infrastructure that lets a small number of proctors supervise many concurrent learners without entering the headset experience. This paper presented a WebSocket-orchestrated VR streaming framework that delivers egocentric video, gaze events, and step-level training-log events from multiple headsets to a browser-based proctor dashboard. Using synthetic clients, we showed that the orchestration layer maintains low-latency operation through 32 concurrent learners under baseline gaze and training-log traffic, while realistic video streaming adds only modest overhead.

The results provide practical design guidance. Moderate-resolution dashboard video can be delivered without degrading responsiveness, whereas continuous audio streaming can saturate the server loop at high concurrency and should be voice-activity gated, batched, and selectively offloaded. Future work should validate the proctor workflow with human users, improve robustness under worker failure, and further reduce overhead for higher-bandwidth modalities.

Acknowledgments
The project is supported by the National Science Foundation under Grant No. 2417371.

References

[1] F. Wang, X. Xu, S. Li, W. Feng, and M. Almasri, “Learning cleanroom microfabrication operations in virtual reality - An immersive and guided learning experience,” Computers & Education: X Reality, vol. 5, Article 100073, Dec. 2024, doi: 10.1016/j.cexr.2024.100073.

[2] A. Collins, J. S. Brown, and A. Holum, “Cognitive apprenticeship: Making thinking visible,” American Educator, Winter 1991.

[3] X. Ochoa and M. Worsley, “Augmenting learning analytics with multimodal sensory data,” Journal of Learning Analytics, vol. 3, no. 2, pp. 213-219, 2016, doi: 10.18608/jla.2016.32.10.

[4] I. B. Adhanom, P. MacNeilage, and E. Folmer “Eye tracking in virtual reality: A broad review of applications and challenges,” Virtual Reality, vol. 27, pp. 1481-1505, 2023, doi: 10.1007/s10055-022-00738-z.

[5] B. Shiferaw, L. Downey, and D. Crewther, “A review of gaze entropy as a measure of visual scanning efficiency,” Neuroscience & Biobehavioral Reviews, vol. 96, pp. 353-366, Jan. 2019, doi: 10.1016/j.neubiorev.2018.12.007.

[6] K. Verbert, S. Govaerts, E. Duval, J. L. Santos, F. Van Assche, G. Parra, and J. Klerkx, “Learning dashboards: An overview and future research opportunities,” Personal and Ubiquitous Computing, 2014, doi: 10.1007/s00779-013-0751-2.

[7] D. M. Broussard, Y. Rahman, A. K. Kulshreshth, and C. W. Borst, “Visual indicators for monitoring students in a VR class,” in IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2021, pp. 502-503, doi: 10.1109/VRW52623.2021.00133

[8] S. Thanyadit, M. Heintz, and E. L-C Law, “Tutor In-sight: Guiding and visualizing students’ attention in a virtual reality classroom using the teacher’s gaze,” in CHI Conference on Human Factors in Computing Systems (CHI ’23), 2023, doi: 10.1145/3544548.3581069.