Rasterift: Application-Layer Glyph and Pixel Video Streaming and Representation-Aware Security

A prototype study of representation transcoding, application payloads, inspection boundaries, and policy continuity

Robin Johns | 13 June 2026 | robinjohns@therobinjohns.com

1. Abstract

Security controls often attach decisions to a particular observation point: file upload, MIME type, media container, decoded frame, network response, application message, rendered browser output, audio stream, metadata field, or retained security artifact. Rasterift investigates the risk that a decision made for one computational representation may not remain valid after the governed information has been transformed into another representation. The prototype decodes conventional media server-side, maps frames into either a glyph grid or reduced pixel field, compresses visual application payloads, transmits them over WebSocket, and reconstructs them in a browser canvas with audio handled separately.

The evaluation uses one independent 6.0 s, 640 × 360 source clip and two looped derivatives of the same content at 30.01 s and 300.01 s. Rasterift Glyph used a 200 × 56 grid; Rasterift Pixel used a 450 × 253 grid. Relative to full-resolution raw RGB transport, Glyph reduced visual application payload by 98.37% and Pixel by 80.25% for the six-second source. Both transformed visual payloads were larger than the compressed MPEG-4 Part 2 Simple Profile source representation in the MP4 container, and the looped derivatives support only repeated-content duration-scaling observations.

The paper's main contribution is a representation-aware policy-equivalence model that separates semantic preservation, classification consistency, and policy consistency. Two representations are policy-equivalent for a specified policy and content property only when the same policy-relevant fact is preserved and the control produces the same justified action under equivalent context. The study does not demonstrate superiority over modern video codecs, commercial DLP or moderation bypass, practical covert-channel capacity, latency improvement, browser-computation reduction, production scalability, or general external validity.

2. Introduction and Research Questions

Conventional video delivery is highly optimized. Modern codecs exploit spatial transforms, prediction, quantization, entropy coding, and container-level delivery to reduce visual byte rates while preserving acceptable perceptual quality [1]-[4]. Adaptive delivery systems then select encoded representations appropriate to bandwidth and playback conditions [5], [6]. Browser media stacks and emerging interfaces such as WebCodecs expose parts of this codec pipeline to web applications while retaining the concept of encoded media chunks, decoded frames, and platform codec resources [7]. Rasterift deliberately occupies a different design space. It decodes a conventional media source on the server, maps frames into lower-dimensional display representations, transports those representations as application messages, and reconstructs visible output in a browser surface.

The prototype is useful precisely because it is not a replacement codec. Rasterift does not define a DCT, wavelet, residual, or coefficient-domain video syntax. It performs representation transcoding: visual information moves from a media container into glyph fields, reduced pixel fields, zlib-compressed payloads, WebSocket messages, JavaScript decoder state, browser display queues, canvas output, audio responses, metadata, and retained security artifacts. That movement gives systems researchers a concrete environment for measuring the cost of transformed representations, and it gives security researchers a concrete environment for asking whether policy decisions remain valid after information changes computational form.

The central thesis is therefore representation-continuity rather than codec efficiency. Security decisions attached to one computational representation cannot automatically be assumed to remain valid after the information has been transformed into another representation. Rasterift makes the boundaries explicit enough to compare source media, decoded frames, transformed glyph and pixel representations, compressed payloads, browser reconstruction, audio, metadata, logs, and samples under the same policy question.

The measurements reported here are intentionally narrow. Rasterift Glyph and Rasterift Pixel reduce visual application payload relative to full-resolution raw RGB transport, but both are larger than the compressed source-container rate in the current experiment. The 30-second and 300-second clips repeat the same six-second source content. They test duration scaling, payload accumulation, and throughput stability under repeated content; they do not support claims about sports footage, screen recordings, surveillance video, animation, high-motion scenes, or other media classes. The

results also do not show superiority over H.264, H.265, AV1, WebCodecs, reduced-resolution conventional video, or native browser playback.

The security question is similarly bounded. Rasterift is a representation-transcoding system that may create inspection or policy-decision mismatch when security controls observe only one representation or trust boundary. This paper therefore avoids treating transformed media as inherently malicious. It asks when a policy-relevant fact survives a representation change, when a detector continues to recognize it, and when the resulting policy action remains justified. It does not claim that Rasterift has bypassed any commercial product. The objective is to make the representation boundary measurable enough that such claims could be tested rigorously.

Table 1: Research questions and where they are answered.

Question	Scope	Answer location
RQ1	What visual application-payload cost do Rasterift Glyph and Rasterift Pixel incur relative to raw RGB, the source video stream, and the complete source container in the measured prototype?	Sections 8-10 and 14.
RQ2	How stable are transformed-frame payload sizes and offline processing throughput when identical source content is repeated over longer durations?	Sections 8-10 and 14; limited to repeated-content duration scaling.
RQ3	Which security observation points and trust boundaries change when visual information moves from a media container into glyph fields, reduced pixel fields, compressed WebSocket payloads, and browser-canvas reconstruction?	Sections 5, 7, 11, and 12.
RQ4	Under what conditions can semantic preservation, classification consistency, and policy consistency be established across source, transformed, and reconstructed media representations?	Sections 6, 7, 11-14.

3. Associated Studies and Adjacent Systems

Rasterift is closest to work on media representation, browser delivery, non-photorealistic rendering, and information-flow security, but it is not a direct extension of any single line of work. Conventional video standards such as H.264/AVC, H.265/HEVC, AV1, and MPEG-4 Part 2 define compressed media syntax and decoding behavior [1]-[4], while HLS and MPEG-DASH define adaptive delivery over HTTP [5], [6]. WebCodecs exposes browser codec work queues and decoded frames to web authors [7], and WebSocket defines an application messaging channel rather than a media transport [8]. Rasterift differs from these systems by decoding media server-side and sending representation-transcoded visual payloads as application-layer messages.

Adjacent engineering patterns include JPEG, WebP, PNG frame streams, remote framebuffers, and browser-canvas reconstruction [9]-[12]. These systems show that visual information can be moved as still-image sequences, framebuffer updates, or canvas buffers rather than as conventional media segments. Rasterift uses that design space as an experimental substrate: the point is not to improve codec efficiency, but to expose measurable boundaries between source media, decoded frames, transformed payloads, WebSocket messages, and browser reconstruction.

Raster-to-symbol rendering is also established. AA-lib demonstrated practical ASCII-art rendering for images and video [13], and Markuš et al. studied fast rendering of image mosaics and ASCII art using grid partitioning and codebook selection [14]. Spatial reduction and quantization are likewise standard signal-processing tools [15]. Rasterift's novelty is therefore not that images can be reduced to symbols or pixels. Its contribution is that those reduced representations are linked to source artifacts, protocol state, measured payload cost, and security observation points.

The security argument draws on data leakage prevention, confinement, covert-channel, steganography, perceptual hashing, and image-quality literature. DLP work distinguishes content, channel, deployment point, and data state [16]. Lampson's confinement problem and Saltzer and Schroeder's protection principles show why outputs and derived artifacts matter to policy enforcement [17], [18]. Covert-channel and steganography studies motivate caution about permitted carriers [19]-[21], while perceptual hashes and structural quality metrics show both the promise and limits of transformed-content comparison [22]-[25]. Rasterift uses these adjacent studies to frame policy equivalence as an empirical question rather than an assumption.

Vision-language models remain relevant only as future work. Language models and vision-language systems demonstrate that visual evidence can be associated with textual descriptions and multimodal reasoning tasks [26]-[28], but Rasterift does not implement or evaluate that path. In the present paper, those studies motivate a future representation-derived summary experiment; they are not part of the measured contribution.

4. Terminology and Contributions

Representation transcoding is the primary technical term used in this paper. It denotes the process of decoding an existing media representation and mapping the decoded visual frames into a different display representation. Rasterift differs from conventional transform coding because it does not introduce a DCT-, wavelet-, residual-, or coefficient-domain compressed-video syntax. Its transformed outputs are application-layer display representations.

Rasterift Original denotes conventional source playback through a browser video path. In the application, Original mode serves a browser-compatible source resource and relies on the browser media stack for playback. It is the fidelity baseline for the user, but this paper does not measure complete browser traffic for Original mode.

Rasterift Glyph denotes raster-to-symbol streaming. Server-side decoding produces frames that are downsampled to a character grid; luminance is quantized into a glyph alphabet; and the resulting text-like visual payload is transmitted to the browser. In the measured configuration, Glyph used a 200 × 56 grid for the 640 × 360 source. Per-cell Glyph color was not enabled in the measured Glyph result.

Rasterift Pixel denotes reduced raster-cell streaming. Server-side decoding produces BGR frames that are downsampled to a color-cell field and serialized as compact pixel buffers. In the measured configuration, Pixel used a 450 × 253 BGR grid, with the browser converting BGR cells into RGBA ImageData for canvas reconstruction.

Rasterift Stream Protocol denotes the prototype's application-layer message structure. Initial text messages communicate frame rate, render mode, grid dimensions, and representation flags. Binary visual frames carry a 32-bit big-endian frame index, an 8-bit codec tag, and a payload. Payloads may be raw frame buffers, zlib-compressed full frames, or zlib-compressed deltas.

Representation laundering is terminology proposed cautiously by this paper. It denotes the movement of governed information into a representation that may not receive an equivalent security decision. The term describes a policy risk, not a proven bypass outcome.

The contributions are fourfold. First, the paper documents the Rasterift prototype and its upload, transformation, transport, reconstruction, audio, and deletion paths. Second, it specifies the representation and protocol model sufficiently to distinguish source media, decoded frames, transformed frames, raw messages, zlib messages, delta messages, browser decoder state, and derived artifacts. Third, it reports preliminary visual application-payload measurements for a six-second source and two repeated-content duration-scaling derivatives. Fourth, it provides a policy-equivalence model and defensive threat model for representation-aware inspection and governance.

5. Prototype Architecture

Rasterift is implemented as a local Python/FastAPI application serving static assets, upload, listing, source retrieval, selection, deletion, audio extraction, and WebSocket routes. The tested deployment uses local HTTP at `http://localhost:8000` with TLS disabled. The browser client exposes a video library, upload control, delete controls for uploaded files, playback controls, and three modes: Rasterift Original, Rasterift Glyph, and Rasterift Pixel.

Uploads are submitted as multipart form data, assigned sanitized local filenames, stored under the upload directory, and probed with OpenCV. Original mode either serves a compatible source file directly or creates a browser-compatible H.264/AAC MP4 cache using FFmpeg when required. Transformed modes use OpenCV to decode frames and resize them with bilinear interpolation. Audio is extracted separately by FFmpeg as an MP3 stream for transformed playback; transformed visual-payload measurements in this paper exclude audio bytes.

For Glyph, OpenCV produces a reduced BGR frame, the prototype converts it to grayscale, and luminance values select characters from a fixed glyph alphabet. The measured Glyph path serializes a text frame as `frame_index + "\n" + rows`. For Pixel, OpenCV produces a reduced BGR cell field; the server serializes the cell bytes; and the browser converts BGR to RGBA for ImageData before calling putImageData. The analytical model uses RGB channel order, but the implementation uses BGR internally because OpenCV returns BGR arrays. The browser reconstruction path uses RGBA buffers because that is the canvas ImageData format [12].

The adaptive binary visual frame has the form `index32_be || tag8 || payload`. The first four bytes are an unsigned 32-bit frame index in network byte order. The fifth byte is a codec tag. Tag 0 denotes a raw framebuffer; tag 1 denotes a zlib-compressed full framebuffer; tag 2 denotes a zlib-compressed delta. Delta payloads concatenate little-endian 32-bit changed-cell indices and changed cell values before zlib compression. zlib level 3 is used. The current implementation forces a full-frame candidate every 48 frames as a bounded recovery point inside a continuous session. That interval does not by itself provide complete late-joiner or reconnection support;

a production protocol would also need decoder reset messages, state acknowledgement, keyframe requests, sequence validation, session admission at a full-frame boundary, or explicit reference-state identifiers.

The current implementation selects the smallest candidate payload among the encodings it builds. In the measured Pixel clips, every binary frame selected zlib full-frame encoding; no raw or delta payloads were selected. For the evaluated content and configuration, spatially smooth reduced frames compressed well as full zlib payloads, while the delta format had to carry explicit changed indices and changed values. With lossless equality, many reduced cells were treated as changed, so the indexed delta representation was not smaller. This result is content-specific. Static slides, low-motion interfaces, surveillance footage, animation with large unchanged regions, lossy thresholds, block-based deltas, run-length coding, or motion-compensated approaches may behave differently.

The browser initializes decoder state after receiving the `INIT` text message. Incoming binary messages are decoded sequentially, and the decoded frame becomes the current reconstructed decoder state. Only after decoding is the frame queued for display. The display loop compares frame time against the audio clock and may remove late decoded frames from the display queue. This distinction matters: the measured path may drop a decoded frame from drawing without corrupting decoder state. A future design that discarded undecoded messages before state update would have different semantics and would require explicit state recovery.

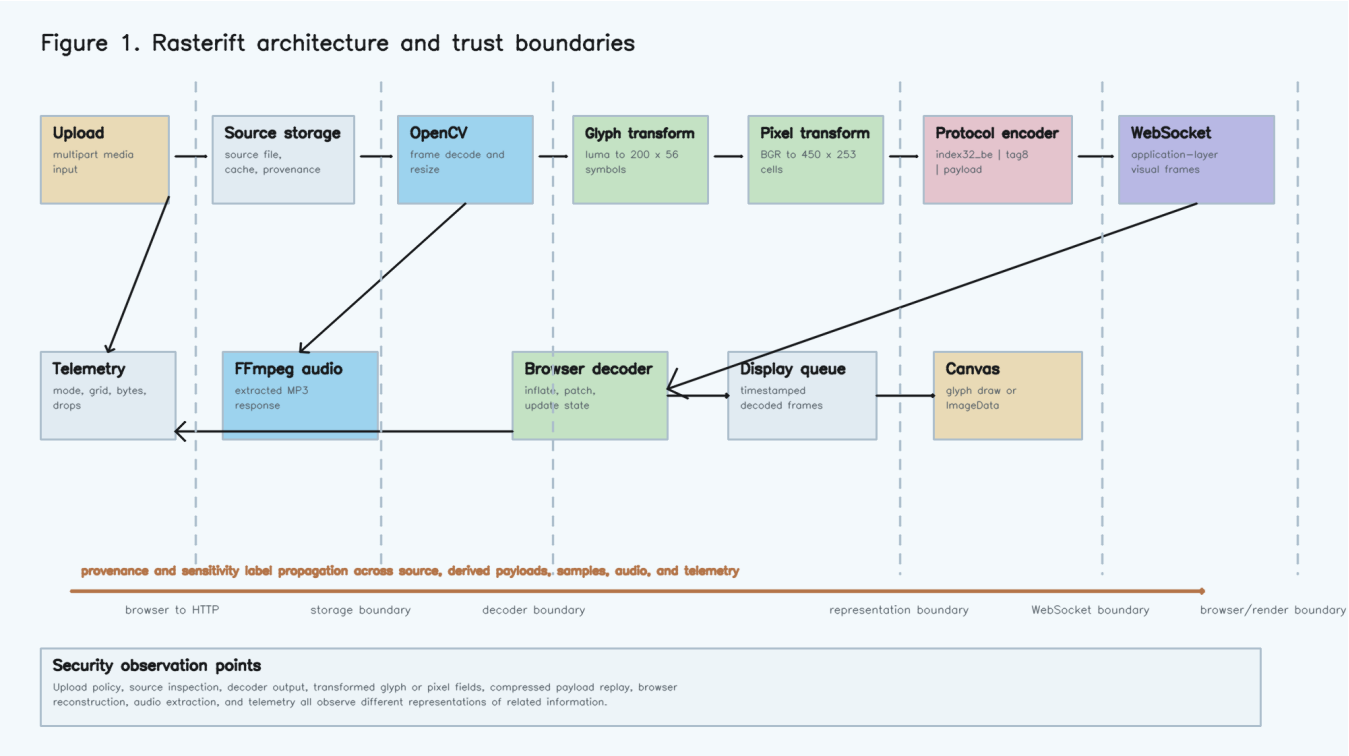

Figure 1: *Rasterift architecture. The prototype moves uploaded media through local storage, OpenCV/FFmpeg decoding, Glyph or Pixel transformation, protocol encoding, WebSocket transmission, browser decoding, canvas reconstruction, and separate audio extraction. Dashed lines mark major trust boundaries that should be instrumented in shared or internet-facing deployments.*

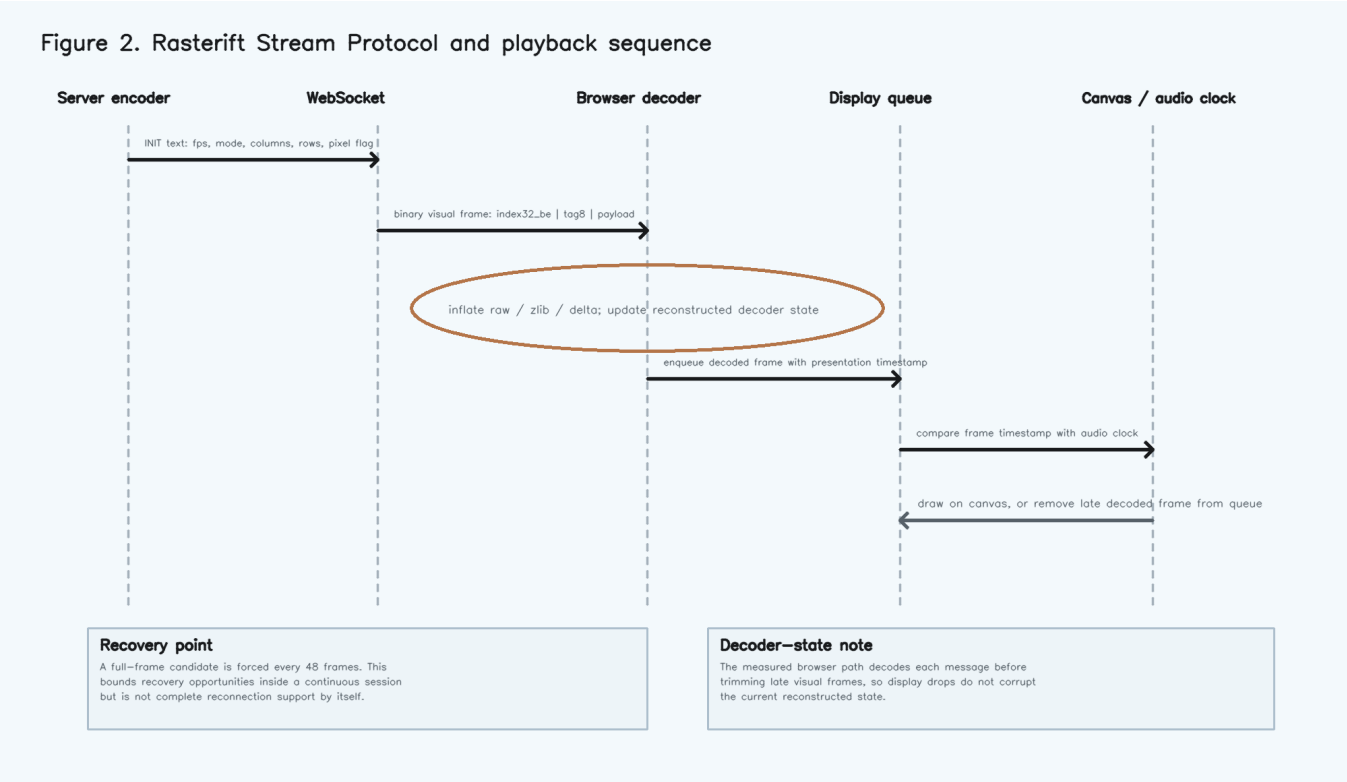

Figure 2: *Protocol and playback sequence. The* `INIT` *text message precedes binary frame transmission. The browser decodes incoming messages and updates decoder state before display-queue trimming, preserving decoder state even when late visual frames are not drawn.*

6. Formal Representation Model

The formal model is intended to make accounting boundaries explicit. It describes the source frame sequence, the downsampled representation, the serialized payload, and the protocol message. It also separates analytical RGB notation from BGR implementation arrays and RGBA browser reconstruction buffers.

Table 2: Notation used in the representation model.

Symbol	Meaning
V	Decoded source video as a finite frame sequence.
F_t	RGB source frame at time index t, height H, width W, and three 8-bit channels.
C	Requested transformed column count.
R_g, R_p	Derived Glyph and Pixel row counts.
ρ	Character-cell aspect ratio, defined as cell height divided by cell width. The measured Glyph path uses an approximate value of 2.
$D_{C,R}$	Spatial downsampling operator from the source frame to a C by R display grid.
Y_t	Luminance plane after downsampling.
A	Ordered glyph alphabet of length K.
S_t	Glyph symbol plane.
Q_t	Optional per-cell color plane for colored Glyph variants; disabled in measured Glyph results.
P_t	Reduced Pixel color field.
B_t	Serialized full-frame payload bytes for a transformed frame.
M_t	Rasterift Stream Protocol message for frame t.
x, $T(x)$	A source artifact and a transformed representation of that artifact.
g	A specified policy-relevant property, such as confidential text, a face, a prohibited symbol, sensitive spoken audio, an internal diagram, or an identifier.
D, P, c	A detector or reviewer, a policy-decision function, and representation context.

Let the decoded source video be a sequence of RGB frames:

$$V = (F_0, ..., F_{T-1}), \text{ with } F_t \in \{0, ..., 255\}^{H \times W \times 3}. \quad (1)$$

For a requested column count C, Glyph rows preserve the source aspect ratio while correcting for character-cell geometry; Pixel rows treat reduced cells as square display samples:

$$R_g = round((H/W)C/\rho), \text{ and } R_p = round((H/W)C). \quad (2)$$

For the 640 × 360 source, C=200 and ρ=2 yield R_g=56; C=450 yields R_p=253. The spatial downsampling operator produces either a Glyph sampling field or a Pixel color field:

$$Z_t = D_{C,R_g}(F_t), \text{ and } P_t = D_{C,R_p}(F_t). \quad (3)$$

Glyph selection is computed from luminance. For downsampled RGB value (R, G, B) at cell (r, c), the model uses the conventional luma approximation:

$$Y_t(r,c) = 0.299R + 0.587G + 0.114B; S_t(r,c) = A[\blacksquare Y_t(r,c)(K-1)/255\blacksquare]. \quad (4)$$

The measured Glyph representation serializes S_t and omits Q_t. Pixel serializes P_t directly. The protocol message forms are:

$$M_t^{raw} = I_t \,||\, 0x00 \,||\, B_t; M_t^{zlib} = I_t \,||\, 0x01 \,||\, zlib_3(B_t); M_t^{\Delta} = I_t \,||\, 0x02 \,||\, zlib_3(\Delta_t). \quad (5)$$

Here I_t is the 32-bit big-endian frame index. Δ_t contains little-endian changed-cell indices and changed cell values relative to the current reconstructed decoder state, which in this implementation is the previous decoded frame. This differs from the most recent full-frame message, the previous transmitted frame, and the previous displayed frame. A zlib or raw full-frame message replaces the decoder state; a delta message patches a copy of the current reconstructed state. The forced 48-frame full-frame interval bounds recovery opportunities inside a continuous session, but complete reconnection or late-joiner support would require explicit reference-state identifiers, decoder reset semantics, acknowledgement state, keyframe requests, sequence validation, and admission at a full-frame boundary.

The implemented encoder currently chooses the smallest candidate payload. A more complete encoder objective would use a weighted cost function:

$$e_t^* = \arg\min_{e \in E_t} [\, \lambda_b |M_t^e| + \lambda_{enc} T_{enc}(e) + \lambda_{dec} T_{dec}(e) + \lambda_{lat} L(e) + \lambda_{rec} R(e) \,]. \quad (6)$$

The weights define the intended optimization model: payload bytes, encode time, decode time, latency, and recovery risk. The current implementation corresponds to a size-only setting in which λ_b=1 and the other terms are not measured. Future protocol work should add acknowledgement state, recovery requests, explicit decoder reset semantics, and a policy for reconciling delta-dependent frames with late-frame dropping.

Policy equivalence is defined only with respect to a specified policy-relevant property. Let x be a source artifact, $T(x)$ a transformed artifact, g a governed property, D a detector or reviewer, P a policy-decision function, and c the representation context. Semantic preservation asks whether the governed property survives the transformation:

$$S_g(x, T) = present(g, x) \wedge present(g, T(x)). \quad (7)$$

Classification consistency then asks whether the same preserved property is identified consistently across compatible representations:

$$C_g(x, T, D) = S_g(x, T) \wedge [D(x, g, c) = D(T(x), g, c_T)]. \quad (8)$$

Policy consistency asks whether equivalent evidence and context lead to the same justified action, such as allow, block, redact, retain, quarantine, or alert:

$$E_g(x, T, P) = S_g(x, T) \wedge [P(x, g, c) = P(T(x), g, c_T)]. \quad (9)$$

Two representations are policy-equivalent for a specified policy and content property when the same policy-relevant fact is preserved and the control produces the same justified action under equivalent context. Representations are not expected to be policy-equivalent when the transformation genuinely removes the governed information. Glyph may remove small text that remains visible in Original or Pixel; audio may contain sensitive speech absent from visual frames; metadata may contain identifiers absent from decoded media; and cropping, quantization, or downsampling may remove a relevant feature. Hash equality is neither necessary nor sufficient, and perceptual similarity alone does not prove policy equivalence.

7. Representation and Inspection Threat Model

The threat model distinguishes demonstrated prototype behavior from risks that arise only under particular deployment assumptions. Localhost research use primarily exposes local files and measurement artifacts to the local operator. Authenticated shared deployment adds user identity, authorization, retention, and provenance requirements. Internet-facing multi-user deployment adds unauthenticated upload pressure, malicious media, origin risks, denial-of-service exposure, and network-visible transformed streams.

The protected assets are not limited to the uploaded media file. A single source may produce extracted audio, transformed glyph rows, reduced pixel fields, zlib-compressed protocol payloads, reconstructed browser output, provenance metadata, user identifiers, logs, sampled frames, and cached derivatives. Processing, network, storage, and browser-rendering capacity are also assets because representation transcoding can convert a media object into sustained compute and memory pressure. The relevant actors therefore include the local researcher,

ordinary authenticated users, authenticated users who misuse their access, unauthenticated external attackers in an exposed deployment, malicious file uploaders, modified or compromised browser clients, insiders attempting unauthorized transfer, application operators, and network or endpoint defenders.

The most important trust boundaries are the browser-to-HTTP upload path, the browser-to-WebSocket stream, the server-to-storage boundary, the server-to-OpenCV/FFmpeg decoding boundary, the protocol decoder-to-browser renderer boundary, and the application-to-telemetry boundary. These boundaries matter because policy may be enforced at one point while the policy-relevant fact becomes intelligible at another. A gateway may see a media upload; a server may see decoded frames; a WebSocket monitor may see compressed application messages; a browser may see reconstructed canvas output; and a logging system may retain derived samples or hashes. Representation-continuous policy enforcement requires provenance and sensitivity labels to follow artifacts when the governed fact is semantically preserved.

The security objectives are confidentiality, integrity, availability, provenance, access control, lawful monitoring, retention, deletion, and policy continuity across representations. The measured prototype deliberately uses local HTTP without TLS, so it should not be treated as a secure shared deployment. A shared or internet-facing version would require TLS, authentication, authorization, origin validation, upload isolation, decoder sandboxing, message-size limits, endpoint or decrypted-traffic visibility for inspection, server trust, client-integrity assumptions or endpoint instrumentation, and a defined logging and retention policy. Because a modified client can alter presentation and local telemetry, final security decisions cannot rely solely on browser-reported state; they must be enforced or corroborated at trusted boundaries.

Table 3: Threat-model matrix for representation-transcoded visual streams. The entries separate architectural risk, measured behavior, proposed validation, and operational control.

Threat scenario	Preconditions	Preserved property	Representation boundary	Detector or control	Possible inconsistency	Required telemetry	Preventive control	Detective control	Validation method
Inspection mismatch	Policy is enforced at upload or aggregate network level.	Visible text, diagram, logo, face, or other visual property survives transformation.	Source container to Glyph, Pixel, zlib payload, WebSocket, canvas.	DLP, classifier, OCR, reviewer, or rule.	Semantic preservation without classification or policy consistency.	Source hash, mode, grid, frame window, payload bytes, sampled reconstruction.	Representation-aware inspection and source-linked labels.	Reference decoder replay and sampled-frame review.	Compare semantic preservation, detector result, and policy action across representations.
Decoder exposure	Untrusted media reaches FFmpeg or OpenCV.	Not required; this is parser and availability risk.	HTTP upload to decoder worker.	Upload validation, sandbox, parser hardening.	Malformed input reaches privileged or shared processing.	Decoder stderr, crashes, process exits, MIME/type result, source hash.	Isolation, patching, type allowlists, size limits.	Malformed-media test corpus and crash telemetry.	Run hostile and malformed files in an isolated harness.
Resource exhaustion	High grid size, high FPS, repeated Pixel streams, or slow clients.	Not required; availability is the protected property.	Protocol encoder, WebSocket, browser queue, canvas.	Admission control, quotas, backpressure.	Accepted session consumes disproportionate CPU, memory, or queue capacity.	FPS, grid, bytes/frame, queue depth, dropped frames, CPU, memory, disconnects.	Message-size limits, FPS caps, grid caps, session quotas.	Load testing and queue-depth alarms.	Constrained-client and multi-client stress tests.
Object-access failure	Shared deployment exposes source or derivative routes.	Sensitive source, audio, sample, cache, or transformed payload remains present.	Storage, source route, audio route, cache, sample store.	Object authorization and route policy.	Source access differs from derivative access.	User id, object id, source hash, derivative id, access decision.	Object-scoped authorization for every derivative.	Impossible-access and cross-user access alerts.	Route-level authorization tests over source and derivatives.
Lineage and retention drift	Derived artifacts persist after source labels or deletion duties are lost.	Governed information remains in payloads, samples, hashes, logs, audio, or metadata.	Application telemetry, logs, retention stores.	Data inventory, retention, deletion workflow.	Derivative loses provenance or declassification evidence.	Source hash, policy label, transform mode, timestamp, deletion state.	Provenance inheritance and deletion propagation.	Retention audits and derived-object inventory.	Deletion tests covering source, cache, audio, samples, logs, and retained payloads.
Untrusted client rendering	Browser or client script is modified.	Rendered visual evidence or local telemetry may differ from expected output.	Protocol decoder to browser renderer.	Reference decoder, endpoint instrumentation, server policy.	Client-reported display state differs from trusted replay.	Decoder errors, frame index, queue state, sampled frames, client version.	Enforce policy at trusted boundaries rather than browser claims alone.	Reference replay and endpoint attestation where available.	Compare server-side replay against client-rendered output.

8. Experimental Method

The evaluation is split into two experiments. Experiment 1 measures representation cost for the independent six-second source. Experiment 2 measures duration scaling over two looped derivatives of the same content. The measurement script processes all frames in each clip for Rasterift Glyph and Rasterift Pixel. It records application visual payload bytes only. Unless explicitly stated otherwise, audio bytes, WebSocket frame overhead, TCP/IP headers, TLS, retransmission, initialization messages, browser cache effects, and total captured network bytes are excluded.

The source video is MPEG-4 Part 2 Simple Profile video in an MP4 container with AAC audio. It should not be described generically as modern compressed video. The source MP4 had the lowest nominal byte-rate baseline among the evaluated stored and transformed representations. H.264, H.265, AV1, WebCodecs, reduced-resolution conventional video, JPEG/WebP frame streaming, MJPEG, PNG frame sequences, and raw captured network traffic are proposed baselines for future work; they were not measured here.

The source MP4 includes both video and audio container data. Rasterift visual payload measurements include only transformed visual application payload. Complete source-container comparisons are therefore indicative and intentionally conservative in favor of the transformed modes only when clearly labelled; the comparison is not a complete browser-traffic comparison for Original playback. The transformed audio endpoint is configured to emit MP3 audio at 128 kbit/s when audio is enabled, but transformed audio bytes were not captured in this measurement pass.

Experiment 1: Representation-cost measurement. The independent six-second source was processed once for Glyph and once for Pixel. The script records frame count, grid dimensions, per-frame payload statistics, total visual payload bytes, processing runtime, raw RGB baseline, source video-stream rate, and complete-container rate.

Experiment 2: Duration-scaling evaluation. The 30.01 s and 300.01 s derivatives were generated by looping the same source content. These clips test repeated-content processing, approximate linear payload accumulation, per-frame payload stability, and processing throughput over longer durations. They are not independent visual samples.

A perceptual-utility evaluation was not performed. Future work should measure human scene recognition, object recognition, OCR accuracy, UI-text readability, diagram interpretation, classifier accuracy, SSIM, LPIPS, and multiple Glyph and Pixel grid sizes. Figure 3 provides visual evidence of representation differences, but it is not a substitute for controlled perceptual or classifier evaluation.

Table 4: Local environment used for the final measurement run. Unique hardware identifiers were intentionally omitted.

Item	Observed value
Host	Apple MacBook Pro, model identifier Mac15,6.
Processor	Apple M3 Pro, 11 CPU cores: 5 performance and 6 efficiency.
GPU/display	Apple M3 Pro integrated GPU, 14 GPU cores; built-in 3024 × 1964 Liquid Retina XDR display.
Memory	36 GB unified memory; `sysctl hw.memsize` reports 38,654,705,664 bytes.
Operating system	macOS 26.5.1, build 25F80; Darwin 25.5.0, arm64.
Browser reference	Google Chrome 149.0.7827.103 installed locally. Visual verification used a Chromium-family application webview.
Python/runtime	Python 3.14.3 in `.venv`; NumPy 2.4.6; OpenCV 4.13.0; FastAPI 0.136.3; Uvicorn 0.49.0.
FFmpeg	FFmpeg/ffprobe 8.1.1 built with Apple clang 21.0.0; configuration includes libx264, libx265, libsvtav1, libvpx, libmp3lame, libdav1d, VideoToolbox, AudioToolbox, OpenSSL, and NEON.
Repository state	Base commit `f0a411e349554056fa7fc66fa6d4263acac6fba2`. The working tree contained manuscript, UI, measurement-script, generated-figure, and PDF changes during measurement; no clean release tag existed at measurement time.
Network/TLS state	Local HTTP server at `http://localhost:8000`; TLS disabled. Measurements report visual application payload bytes and exclude TCP/IP, WebSocket network framing, retransmission, initialization messages, TLS, and audio bytes.

Table 5: Benchmark clips and source characteristics.

Clip	Construction	Video properties	Container and audio	SHA-256
6.0 s source	Independent project demonstration clip.	640 × 360; MPEG-4 Part 2 Simple Profile; yuv420p; 24 FPS; 144 frames; video stream rate 1.316 Mbit/s.	MP4; AAC LC audio at 69.4 kbit/s; complete container 1,043,908 B, 1.392 Mbit/s.	`c375e4f8e05f980e7160cf27a9fdc64bb35575095224bcc54628732628fa4ed3`

Clip	Construction	Video properties	Container and audio	SHA-256
30.01 s looped derivative	Repeated-content duration-scaling clip derived from the source.	640 × 360; MPEG-4 Part 2 Simple Profile; 718 frames; average 23.93 FPS; video stream rate 1.311 Mbit/s.	MP4; AAC LC audio at 69.1 kbit/s; complete container 5,195,908 B, 1.385 Mbit/s.	`3866252fc809b5603d372fc7e2b7d385808afe64f2809df74519a4494ee64146`
300.01 s looped derivative	Repeated-content duration-scaling clip derived from the source.	640 × 360; MPEG-4 Part 2 Simple Profile; 7,173 frames; average 23.91 FPS; video stream rate 1.308 Mbit/s.	MP4; AAC LC audio at 69.1 kbit/s; complete container 51,821,421 B, 1.382 Mbit/s.	`5809c6f7261ad9819805a328a6f0881a9b824a47578ba593fd191e313d35c8c4`

9. Results

Experiment 1 answers RQ1 for the measured prototype. A full-resolution raw RGB frame at 640 × 360 contains 691,200 B before protocol overhead. At 24 FPS that visual baseline is 16.59 MB/s, or 132.71 Mbit/s. Rasterift Glyph averaged 11,258 B/frame and Rasterift Pixel averaged 136,483 B/frame. Glyph therefore reduced application visual payload by 98.37% relative to raw RGB, while Pixel reduced it by 80.25%. The result is a raw-transport comparison, not a codec-efficiency comparison.

Against the complete source-container average, the transformed visual payloads are larger. Glyph is approximately 1.55× the complete source-container average per frame, and Pixel is approximately 18.83×. This is the most important boundary condition in the performance result. Rasterift's transformed modes reduce raw application visual transport, but the compressed source MP4 has the lowest nominal byte-rate baseline in this study.

Glyph frame size is nearly constant because the serialized frame is dominated by a fixed 200 × 56 character grid, line breaks, and frame index. Pixel frame size varies because zlib-compressed BGR cell fields depend on image content and repeated visual structure. The observed cyclic or sawtooth pattern in Figure 4 follows repeated changes in the source animation and the resulting compressibility of full-frame Pixel buffers. It should not be interpreted as a general property of zlib full-frame encoding for arbitrary video.

Experiment 2 answers RQ2 only for repeated-content duration scaling. The 30.01 s and 300.01 s derivatives show stable per-frame payload sizes and approximately linear aggregate payload accumulation. Glyph averaged 11,259 B/frame across the looped derivatives; Pixel averaged approximately 136.8 kB/frame. Offline transformation throughput exceeded the 24 FPS source rate on the tested host: the six-second run processed Glyph at 835.5 frames/s and Pixel at 179.7 frames/s, corresponding to 1.20 ms/frame and 5.57 ms/frame respectively. This does not establish end-to-end playback latency, multi-client capacity, network delivery performance, browser rendering performance, audio-synchronization quality, or production scalability.

Figure 3: *Representation comparison for natural-scene and screen/document examples. Each row shows Rasterift Original, Rasterift Glyph, and Rasterift Pixel. The examples demonstrate representation change, spatial reduction, and information loss; they are not evidence of validated perceptual equivalence.*

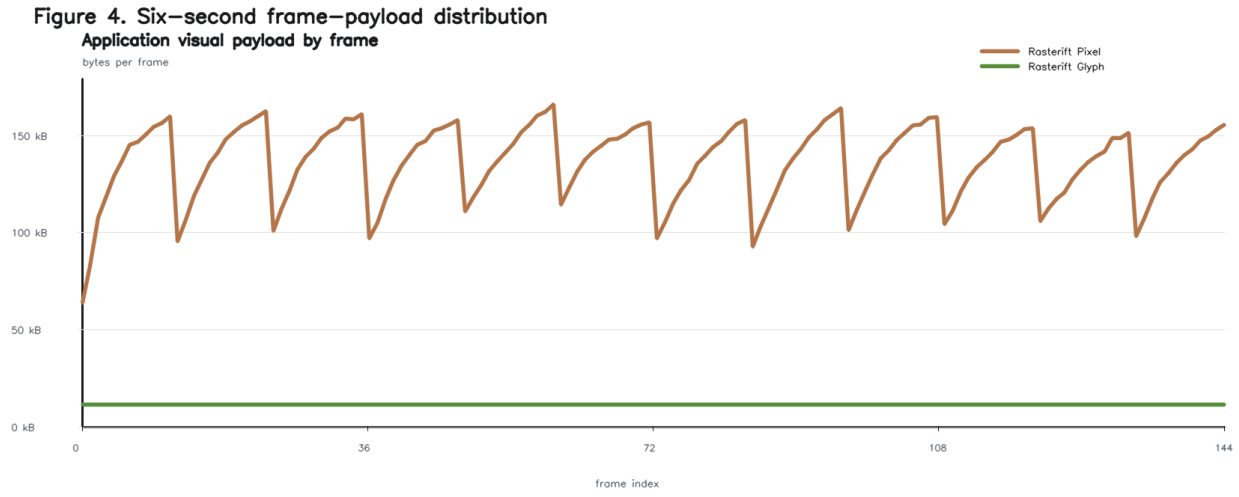

Figure 4: *Frame-payload distribution for the six-second source. Glyph is nearly constant because the text grid dominates payload size. Pixel varies with zlib compressibility of the reduced color field. Audio and network overhead are excluded.*

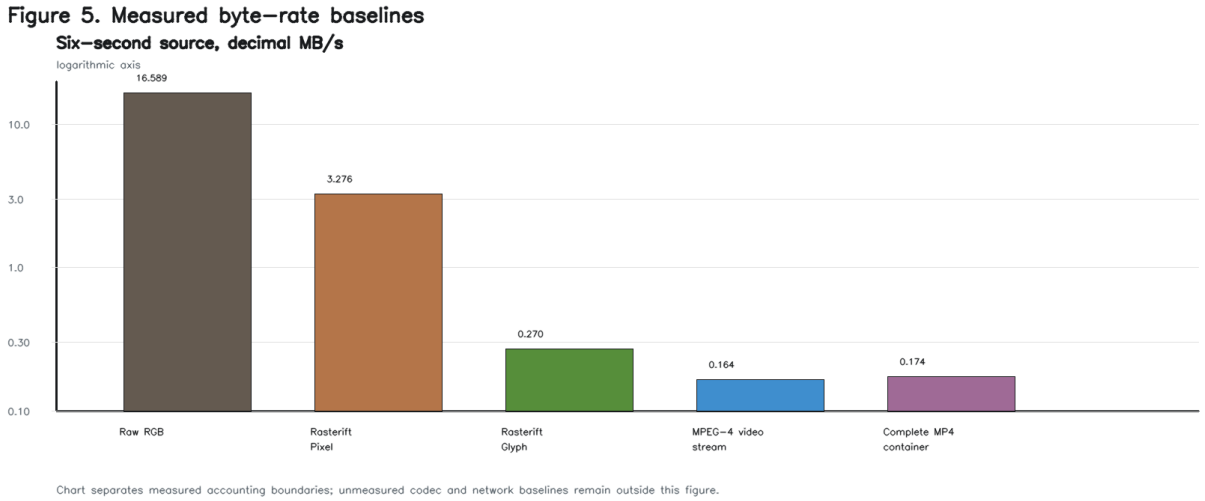

Figure 5: *Measured byte-rate baselines for the six-second source. The chart separates raw RGB, transformed visual application payload, source video-stream rate, and complete source-container rate. Additional codec and frame-streaming baselines are explicitly marked as not measured in this study.*

Table 6: Experiment 1 accounting boundaries for the six-second source. B means bytes; kB and MB are decimal units.

Representation or boundary	Measured quantity	Per-frame visual payload	Byte rate	Bitrate	Total over 6.0 s	Scope
Raw RGB baseline	Calculated from 640 × 360 × 3 at 24 FPS.	691,200 B	16.59 MB/s	132.71 Mbit/s	99.53 MB	Visual bytes only; no protocol overhead.
Source video stream	ffprobe video-stream bitrate.	Not frame-comparable	164.44 kB/s	1.316 Mbit/s	Approx. 986.64 kB	Video stream only; excludes audio and container overhead.
Complete MP4 container	File size divided by duration.	Indicative average 7,249 B/frame	173.98 kB/s	1.392 Mbit/s	1.044 MB	Container includes video, audio, and metadata.
Rasterift Glyph	Measured visual application payload.	11,258 B mean	270.20 kB/s	2.162 Mbit/s	1.621 MB	Visual payload only; text frame bytes; audio/network excluded.
Rasterift Pixel	Measured visual application payload.	136,483 B mean	3.276 MB/s	26.205 Mbit/s	19.654 MB	Visual payload only; zlib full-frame messages; audio/network excluded.
Transformed audio	Configured endpoint behavior.	Not applicable	Not captured	Configured MP3 target 128 kbit/s	Not captured	Requires direct audio-response measurement.
Total captured network bytes	Not captured in this study.	Not applicable	Not captured	Not captured	Not captured	Requires packet or browser-network instrumentation.

Table 7: Transformed per-frame visual payload statistics. Network overhead, TLS, retransmission, initialization messages, and audio are excluded.

Clip	Mode	Frames	Grid	Mean B/frame	Median B	p95 B	Min / max B	Runtime	Throughput	ms/frame	Codec tags
6 s	Glyph	144	200 × 56	11,258.24	11,258	11,259	11,257 / 11,259	0.17 s	835.5 frames/s	1.20	Text frames
6 s	Pixel	144	450 × 253	136,483.49	141,370	159,661	63,997 / 166,033	0.80 s	179.7 frames/s	5.57	144 zlib, 0 raw, 0 delta
30 s loop	Glyph	718	200 × 56	11,258.85	11,259	11,259	11,257 / 11,259	0.80 s	892.1 frames/s	1.12	Text frames
30 s loop	Pixel	718	450 × 253	136,776.48	141,986	156,763	67,001 / 159,080	3.77 s	190.5 frames/s	5.25	718 zlib, 0 raw, 0 delta

Clip	Mode	Frames	Grid	Mean B/frame	Median B	p95 B	Min / max B	Runtime	Throughput	ms/frame	Codec tags
300 s loop	Glyph	7,173	200 × 56	11,259.85	11,260	11,260	11,257 / 11,260	8.29 s	864.8 frames/s	1.16	Text frames
300 s loop	Pixel	7,173	450 × 253	136,814.39	141,986	156,763	67,001 / 159,080	38.81 s	184.8 frames/s	5.41	7,173 zlib, 0 raw, 0 delta

Table 8: Experiment 2 aggregate visual payload and repeated-content scaling.

Clip	Mode	Total visual payload	Visual payload rate	Reduction vs raw RGB	Transformed visual payload divided by complete source-container average - indicative comparison
6 s	Glyph	1.621 MB	270.20 kB/s	98.37%	1.55×
6 s	Pixel	19.654 MB	3.276 MB/s	80.25%	18.83×
30 s loop	Glyph	8.084 MB	269.38 kB/s	98.37%	1.56×
30 s loop	Pixel	98.206 MB	3.272 MB/s	80.21%	18.90×
300 s loop	Glyph	80.767 MB	269.21 kB/s	98.37%	1.56×
300 s loop	Pixel	981.370 MB	3.271 MB/s	80.21%	18.94×

10. Discussion

The measured value of Glyph is not that it beats a compressed source container. It does not. Its value is that it produces a low, stable, text-like visual application payload relative to raw RGB transport. That stability can simplify capacity planning and instrumentation, but it comes with severe information loss. Small text, color distinctions, faces, fine diagrams, and subtle motion may be lost. Without a perceptual evaluation, the paper cannot claim that Glyph preserves enough semantic content for any particular analytic or user task.

Pixel retains more color samples and spatial samples than Glyph at the evaluated configurations. It is therefore the more relevant representation for studying recognizable browser-reconstructed media. The cost is substantial: Pixel is roughly 12.1× the Glyph visual byte rate for the six-second source and roughly 18.8× the complete source-container average. Pixel's contribution is not codec efficiency; it is a controllable reduced-raster representation with explicit application-layer payloads and browser-canvas reconstruction.

No delta frames were selected in the measured Pixel clips. In a lossless configuration, a delta must carry changed-cell indices and changed values. The tested source contains moving gradients and text whose reduced pixel fields change broadly enough that zlib-compressed full frames were smaller than explicit deltas. Different content, lower motion, lower grid sizes, or lossy temporal tolerance could change that result. Equation (6) identifies the broader decision problem, but the current implementation still optimizes payload size rather than measured encode time, decode time, latency, or recovery risk.

The duration-scaling result is consistent but narrow. The looped derivatives show that aggregate transformed visual payload accumulates approximately linearly when identical content is repeated. They do not show content-class robustness. A publication-strength study would require clips across motion classes, resolutions, frame rates, colors, text density, source codecs, grid sizes, and repeated runs. It would also require direct browser and network instrumentation for time-to-first-render, p50 and p95 frame render time, memory, CPU, dropped frames, A/V skew, total network bytes, and behavior under constrained bandwidth and latency.

Accounting boundaries drive interpretation. Comparing transformed visual payload to raw RGB asks whether representation transcoding reduces an uncompressed visual application stream. Comparing transformed visual payload to the source MP4 asks whether it competes with a compressed media file. Comparing total application-layer bytes would require transformed visual plus transformed audio plus initialization and control messages. Comparing total network bytes would additionally require WebSocket framing, HTTP headers, TCP/IP, TLS if enabled, retransmission, caching, and browser behavior. Only the first two boundaries are measured here.

Future work: representation-derived summaries

One future direction is to convert glyph rows, reduced pixel fields, frame statistics, and temporal descriptors into compact textual summaries for search, accessibility, triage, compliance review, security classification, and representation-aware AI inspection. That path is not implemented or evaluated in the current study. It would require separate comparisons against direct image or video models, human annotations, OCR baselines, and policy-decision ground truth. The relevant question would

be the same one developed in this paper: which policy-relevant facts survive the transformation, which are recognized consistently, and which lead to the same justified action under equivalent context?

11. Representation-Aware Security Analysis

Conditions for Inspection or Policy Mismatch

Potential inspection mismatch arises when a control makes a decision about one representation while a policy-relevant fact is preserved in another. A gateway may classify an upload as an MP4, a server may hold decoded frames, a WebSocket monitor may see zlib payloads, a browser may reconstruct a canvas image, and an endpoint may sample the final display. If these observation points are governed independently, the same surviving fact can receive different decisions. This is the representation-laundering risk proposed in this paper: the governed information may remain present, but the artifact to which a control is attached has changed.

DLP implications should be stated carefully. Rasterift does not demonstrate DLP bypass. It does show an architectural inspection gap worth testing: a DLP policy that identifies sensitive screenshots in stored MP4 files or browser video responses may not inspect a zlib-compressed sequence of reduced pixel cells or a glyph grid over WebSocket. Conversely, a policy that inspects only network byte volume may flag raw RGB but miss lower-volume transformed streams or misclassify Pixel because its byte rate resembles application data rather than media. The correct research question is whether the control reconstructs enough of the governed representation to make the same decision.

Content moderation and transformed-content classification face a related challenge. Glyph changes scale, texture, typography, contrast, and luminance distribution. Pixel changes scale, channel order, compression structure, and rendering path. These transformations may weaken classifiers or perceptual hashes that were tuned for ordinary image or video inputs, but that remains a hypothesis until tested against specific models, thresholds, and content classes. The perceptual-hashing literature provides both motivation and caution: robust hashes are designed to tolerate transformations, yet learned and hand-crafted hashing systems have documented collision, robustness, and adversarial limitations [22]-[24]. Policy equivalence therefore requires evidence that the relevant fact survived, that the detector recognized it, and that the policy action remained justified.

Potential Abuse Scenarios

An insider transfer scenario would involve policy-relevant visual information entering a transformed stream whose reconstructed output remains meaningful to a human or downstream classifier, while intermediate controls treat the payload as non-media application traffic. A moderation-mismatch scenario would involve visual content that is transformed before the inspection point and reconstructed after it. A provenance-loss scenario would involve derived glyph or pixel payloads being detached from the original source identifier and policy labels. A covert-channel scenario is more speculative: adversarially chosen glyphs, color-cell perturbations, frame timing, or payload-size modulation could carry secondary information. This paper does not estimate such capacity and does not provide an encoder; it treats the issue as a hypothesis requiring steganalysis, capacity measurement, and defender-visible telemetry.

If an adversary intentionally conceals information within transformed visual payloads, the behavior may correspond to ATT&CK T1027.003, Obfuscated Files or Information: Steganography, or T1001.002, Data Obfuscation: Steganography, depending on whether the behavior is primarily defense evasion or command-and-control obfuscation [29], [30]. If an adversary uses a persistent application channel to move stolen data, the surrounding behavior may correspond to T1041, Exfiltration Over C2 Channel, or T1567, Exfiltration Over Web Service, depending on the infrastructure [31], [32]. These mappings apply to adversary behavior, not to Rasterift itself.

Comparative Security Characteristics

Rasterift Original remains closest to ordinary browser media delivery. Its risks are familiar: upload handling, media parsing, source-file retention, source access control, provenance, audio extraction, and ordinary video-based misuse. It is also the representation most likely to be understood by existing media-aware controls. That does not make it safe by default; it makes its security surface less surprising.

Rasterift Glyph creates the largest representational departure from conventional media objects among the evaluated modes. It removes much color and spatial detail and turns the visual stream into text-like payloads. That makes it useful for testing whether security policy follows a moving image after it is no longer a conventional media object, while also making fidelity an empirical limitation. Fine text, detailed screenshots, many faces, and subtle object categories may become unreadable or ambiguous. Any policy-equivalence claim for Glyph must therefore begin by testing semantic preservation for the specific property at issue.

Rasterift Pixel retains more color samples and spatial samples than Rasterift Glyph at the evaluated configuration, while remaining smaller than raw RGB. Pixel is therefore the stronger candidate for future policy-equivalence testing involving screenshots, interfaces, diagrams, and other visually detailed content. Compared with the compressed

source MP4, Pixel is byte-inefficient here; compared with raw RGB, it changes the observable signature from an obvious raw-frame stream into zlib-compressed application messages and browser-canvas reconstruction. A control stack that understands MP4 files but cannot reconstruct Pixel frames may make a policy-decision mismatch if a governed visual fact is preserved across that boundary.

Resource Exhaustion and Observability

Pixel mode is intentionally heavier than Glyph. At higher column counts, it can generate large WebSocket messages, increased CPU load, browser memory pressure, and GPU/canvas work. Slow clients can build queues; fast producers can waste CPU generating frames that will be dropped. Backpressure, message-size limits, FPS caps, maximum grid dimensions, and session limits are therefore security controls, not only performance optimizations.

Useful telemetry includes selected mode, source hash, dimensions, FPS, grid size, payload bytes per frame, codec tag distribution, dropped frames, decode errors, client disconnects, upload provenance, cache hits, and transformed-stream duration. Aggregate network bytes alone are not enough because the same byte volume can represent very different content sensitivity.

Table 9: Conditional framework traceability matrix. Framework identifiers describe adversary behavior or defensive controls, not Rasterift as a technology label.

Scenario	Conditional framework reference	Telemetry	Preventive or detective control	Validation test
Hidden information carried in transformed visual payloads.	If intentional concealment is present: ATT&CK T1027.003 or T1001.002 [29], [30].	Payload entropy, reconstructed-frame samples, perceptual hashes, payload-size or timing anomalies.	D3FEND File Analysis D3-FA, Content Filtering D3-CF, and Network Traffic Policy Mapping D3-NTPM [33]-[35].	Controlled steganalysis corpus and policy-equivalence decisions across source and reconstructed frames.
Unauthorized transfer over an application channel.	If stolen data is moved through a C2 or web service channel: ATT&CK T1041 or T1567 [31], [32].	Source access followed by WebSocket transfer, unusual destinations, large transformed-stream duration.	Outbound traffic filtering, admission control, authentication, authorization, and DLP policy mapping.	Simulated benign and policy-relevant transfers with endpoint and network logging.
Loss of source provenance after transformation.	Governance problem rather than an ATT&CK technique.	Source hash, user id, transform mode, derived-artifact lineage.	D3FEND Data Inventory D3-DI and Access Mediation D3-AMED [36], [37].	Deletion and retention tests covering source, cache, audio, samples, and logs.
WebSocket or upload misuse.	Application-security control domain.	Origin, authentication state, message size, upload type, parser errors.	OWASP ASVS for requirements and WSTG for upload and WebSocket testing [40]-[42].	Origin validation, file-upload abuse tests, message-size tests, and authorization tests.
Enterprise risk management for transformed media.	NIST CSF 2.0 Govern, Identify, Protect, Detect, Respond, Recover; selected SP 800-53 controls [38], [39].	Asset inventory, data classification, audit events, incident handling records.	Representation-continuity policy, AU logging, AC access control, SI input validation, SC boundary controls.	Control assessment mapped to artifacts and trust boundaries rather than source files alone.

12. Governance and Mitigation

The guiding governance principle is information continuity: derived artifacts should inherit the source's sensitivity and provenance when they retain the relevant governed information. That inheritance is not automatic when a transformation genuinely destroys the governed fact, and it should not replace documented declassification. It does, however, prevent a source file, decoded frame, transformed payload, reconstructed canvas sample, audio extraction, perceptual hash, or telemetry record from silently becoming ungoverned merely because its representation changed.

Preventive controls. Uploads should be authenticated, authorized, size-limited, type-validated, stored outside executable paths, scanned, and decoded in isolated processes. Parser hardening should include FFmpeg/OpenCV patch management, sandboxing, malformed-media tests, and failure isolation. WebSocket admission should require origin validation, authentication, authorization, message-size limits, per-session quotas, grid and FPS caps, backpressure, and keyframe recovery. Derived artifacts should inherit source labels and access-control decisions.

Detective controls. Defenders should log source hash, user identity, upload provenance, mode, dimensions, FPS, grid size, per-frame payload bytes, codec tag distribution, decoder errors, dropped frames, client disconnects, cache hits, transformed-stream duration, audio extraction, and derived-artifact identifiers. Where lawful and proportionate, reference decoders can reconstruct sampled frames for perceptual hashing or classifier inspection. Those samples must themselves be treated as sensitive.

Responsive controls. Incident response should support suspension of upload and stream sessions, revocation of derived-object access, quarantine of source and cache artifacts, deletion across source, transformed, audio, sample, and log stores, and preservation where legally required. Alerts should identify the representation boundary at which the policy mismatch was detected.

Governance controls. Organizations should define lawful basis, proportionality, user notice, retention, deletion, access control, and audit requirements for both source

media and derived representations. Security inspection can itself create sensitive artifacts: reconstructed-frame samples, perceptual hashes, retained transformed payloads, source identifiers, user identifiers, extracted audio, and security telemetry may all reveal governed information. These artifacts require access control, encryption where appropriate, retention limits, deletion propagation, auditability, and incident-response procedures. OWASP ASVS and WSTG provide useful application-security requirements and tests for authentication, access control, input validation, file handling, WebSocket behavior, and logging [40]-[42]. NIST CSF 2.0 and SP 800-53 are useful when mapped to specific assets, telemetry, and trust boundaries rather than cited as broad checklists [38], [39].

The essential defensive lesson is that security policy must follow information through computation. A source file may be deleted while derived samples remain; a WebSocket may be permitted while reconstructed frames display governed material; a perceptual hash may be computed on the wrong representation; and an audio extraction path may persist after visual policy is enforced. Rasterift gives these problems a concrete systems substrate.

13. Limitations, Ethics and Reproducibility

The evidence in this study is deliberately narrow. Rasterift was evaluated with one independent six-second source clip and two longer clips obtained by repeating that same content. The repeated clips are useful for observing duration scaling, payload accumulation, and processing stability under a controlled visual sequence, but they do not provide evidence about other media classes. Different conclusions may follow for screen recordings, surveillance footage, animation, sports video, high-motion scenes, dense user-interface text, low-light material, or sources encoded at different resolutions, frame rates, and codec settings. The paper should therefore be read as a prototype case study and measurement report, not as a claim about the general behavior of representation-transcoded media streams.

The measured construct is also limited. This paper measures visual application-payload bytes and offline processing runtime; it does not measure perceived visual quality, recognizability, OCR retention, classifier accuracy, moderation decisions, DLP outcomes, covert-channel capacity, browser computation, or end-to-end latency. Figure 3 illustrates representational change, but it is not a perceptual study. A stronger evaluation would need human-subject or task-oriented recognition experiments, OCR and classifier benchmarks, reconstructed-frame quality metrics, browser render-time instrumentation, and controlled comparisons against security decisions made at several observation points.

The baseline comparisons should be interpreted with the same caution. The measured baselines are raw RGB, source video-stream rate, complete MP4 container rate, and transformed visual application payload. The study does not measure H.264, H.265, AV1, WebCodecs, reduced-resolution conventional video, MJPEG, JPEG/WebP image sequences, PNG frame streams, or total captured network traffic. It also excludes WebSocket framing, HTTP headers, TLS, retransmission, browser JavaScript decode time, canvas/GPU work, memory pressure, and transformed-audio response bytes. For that reason, the reported processing runtime is not a latency result, and the paper cannot claim codec competitiveness, native-playback superiority, or complete transport efficiency.

The security analysis is likewise bounded by what was actually tested. No named DLP, moderation, gateway, endpoint, browser-security, cloud-security, vision-language, or language-model product was evaluated, and the paper intentionally describes potential inspection mismatch rather than demonstrated bypass. Any claim about the behavior of a commercial control or model would require an authorized test plan, explicit product configuration, a labeled corpus, a ground-truth policy definition, captured telemetry, and reproducible evidence. The contribution here is to identify where representation changes alter observation points and where a future evaluation would need to compare semantic preservation, classification consistency, and policy consistency across source media, transformed payloads, reconstructed frames, audio, metadata, and retained samples.

The next experimental programme should separate media-performance questions from policy-equivalence questions. The media corpus should include talking-head video, high-motion footage, low-motion footage, animation, screen recordings, slide presentations, document capture, low-light footage, detailed natural scenes, and synthetic test patterns, with multiple Glyph and Pixel grid sizes and repeated runs. Baselines should include reduced-resolution H.264, H.265, AV1, WebCodecs, JPEG or WebP frame streaming, MJPEG, PNG frame streaming, and raw RGB. A separate policy-equivalence experiment should use controlled content properties such as visible confidential text, labels, QR codes, faces, logos, screenshots, diagrams, personal identifiers, sensitive audio, and metadata. For each property, the experiment should test whether the fact survives in Original, Glyph, and Pixel; run compatible OCR, classifier, perceptual-hash, or reviewer controls; record true positives, false negatives, false positives, confidence changes, and OCR accuracy; and map the result to a defined policy action.

The ethical posture of the work is defensive. Rasterift is analyzed because representation changes may affect how governed visual information is inspected, retained, and audited. The paper does not provide operational bypass procedures or covert-channel encoders. Future evaluations against organizational controls should be conducted only in

authorized environments, using synthetic, licensed, or otherwise permissible media. Reconstructed-frame samples, perceptual hashes, telemetry, source identifiers, and user identifiers are themselves sensitive artifacts when the source content is sensitive. A deployed system would therefore need user notice, a lawful monitoring basis, object-scoped access controls, retention limits, deletion semantics, and restrictions on secondary use for training or moderation.

14. Conclusion

Rasterift demonstrates a complete working path for representation-transcoded visual streaming: a conventional media source is decoded server-side, transformed into either a glyph grid or reduced pixel field, serialized into application payloads, compressed when beneficial, carried over WebSocket, decoded in the browser, and reconstructed on a canvas with audio delivered separately. That path is the main systems contribution. It is not a new video codec, and it is not evidence that application-layer media streaming outperforms mature codec pipelines. Its value is experimental control over the representations that appear between source media and browser-visible output.

RQ1 is answered by the six-second measurement. Rasterift Glyph and Rasterift Pixel reduce visual application payload relative to full-resolution raw RGB transport: Glyph by 98.37% and Pixel by 80.25% in the measured configuration. The same data also bound the claim: both transformed visual payloads are larger than the compressed MPEG-4 Part 2 source representation in its MP4 container, and the source MP4 has the lowest nominal byte-rate baseline among the evaluated stored and transformed representations.

RQ2 is answered only for repeated-content duration scaling. The 30.01 s and 300.01 s looped derivatives show stable per-frame payload sizes, approximately linear aggregate payload accumulation, and offline transformation throughput above the source frame rate for identical repeated content. They do not establish behavior across independent media classes. RQ3 is answered architecturally: security observation points change at upload, source storage, decoder output, glyph or pixel fields, zlib payloads, WebSocket messages, browser decoder state, display queues, canvas reconstruction, audio extraction, telemetry, and retained samples. RQ4 is answered conceptually rather than experimentally: policy equivalence can be established only when a specified policy-relevant fact is semantically preserved, recognized consistently by the applicable detector or reviewer, and mapped to the same justified policy action under equivalent context.

The boundaries of the conclusion are material. The study does not establish codec efficiency, visual utility, OCR retention, classifier performance, latency improvement, browser-computation reduction, commercial security-control behavior, content-moderation outcomes, covert-channel capacity, or external validity beyond the measured clips. The next empirical steps are therefore clear: broaden the corpus; add repeated trials; measure H.264, H.265, AV1, WebCodecs, reduced-resolution video, and frame-streaming baselines; capture total network and browser resource behavior; perform perceptual and classifier evaluations; run a controlled policy-equivalence experiment over visible text, confidential labels, QR codes, faces, logos, screenshots, diagrams, identifiers, audio, and metadata; and archive a clean reproducibility artifact. Rasterift is most credible when treated as a testbed for representation continuity: a way to study whether security policy follows visual information after computation changes what the information looks like to the system.

15. References

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, Jul. 2003, doi: 10.1109/TCSVT.2003.815165.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649-1668, Dec. 2012, doi: 10.1109/TCSVT.2012.2221191.

[3] Alliance for Open Media, "AV1 Bitstream and Decoding Process Specification," 2018. [Online]. Available: https://aomedia.org/specifications/av1/

[4] ISO/IEC, "ISO/IEC 14496-2:2004, Information technology - Coding of audio-visual objects - Part 2: Visual," 2004. [Online]. Available: https://www.iso.org/standard/39259.html

[5] R. Pantos and W. May, "HTTP Live Streaming," RFC 8216, IETF, Aug. 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8216

[6] ISO/IEC, "ISO/IEC 23009-1:2022, Dynamic adaptive streaming over HTTP (DASH) - Part 1," 2022. [Online]. Available: https://www.mpeg.org/standards/MPEG-DASH/

[7] W3C Media Working Group, "WebCodecs," W3C Working Draft, 8 Jun. 2026. [Online]. Available: https://www.w3.org/TR/webcodecs/

[8] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, IETF, Dec. 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455

[9] ITU-T and ISO/IEC, "Digital compression and coding of continuous-tone still images: Requirements and guidelines," ITU-T Recommendation T.81 / ISO/IEC 10918-1, 1992. [Online]. Available: https://www.w3.org/Graphics/JPEG/itu-t81.pdf

[10] Google, "Compression Techniques: WebP," Google for Developers. [Online]. Available: https://developers.google.com/speed/webp/docs/compression

[11] T. Richardson and J. Levine, "The Remote Framebuffer Protocol," RFC 6143, IETF, Mar. 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6143

[12] WHATWG, "The canvas element," *HTML Standard*. [Online]. Available: https://html.spec.whatwg.org/multipage/canvas.html

[13] J. Hubicka, "AA-lib: An ASCII-art library," AA-project. [Online]. Available: https://aa-project.sourceforge.net/aalib/

[14] N. Markuš, M. Fratarcangeli, D. Pandži■, and J. Ahlberg, "Fast Rendering of Image Mosaics and ASCII Art," *Computer Graphics Forum*, vol. 34, no. 6, pp. 251-261, 2015, doi: 10.1111/cgf.12597.

[15] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA, USA: Kluwer Academic Publishers, 1992.

[16] A. Shabtai, Y. Elovici, and L. Rokach, *A Survey of Data Leakage Detection and Prevention Solutions*. New York, NY, USA: Springer, 2012, doi: 10.1007/978-1-4614-2053-8.

[17] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613-615, Oct. 1973, doi: 10.1145/362375.362389.

[18] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278-1308, Sep. 1975, doi: 10.1109/PROC.1975.9939.

[19] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 3, pp. 44-57, 2007, doi: 10.1109/COMST.2007.4317620.

[20] J. Fridrich, *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge, U.K.: Cambridge University Press, 2009. [Online]. Available: https://www.cambridge.org/core/books/steganography-in-digital-media/2D5989F2462923428BE01F2042A532AB

[21] R. Anderson, R. Needham, and A. Shamir, "The steganographic file system," in *Information Hiding*, Lecture Notes in Computer Science, vol. 1525, pp. 73-82, 1998, doi: 10.1007/3-540-49380-8_6.

[22] V. Monga and B. L. Evans, "Robust perceptual image hashing using feature points," in *Proc. IEEE International Conference on Image Processing*, 2004, pp. 677-680. [Online]. Available: https://users.ece.utexas.edu/~bevans/papers/2004/imageHashing/ICIP2004ImageHashingPaper.pdf

[23] C. Zauner, "Implementation and benchmarking of perceptual image hash functions," M.S. thesis, Upper Austria University of Applied Sciences, 2010. [Online]. Available: https://www.phash.org/docs/pubs/thesis_zauner.pdf

[24] L. Struppek, D. Hintersdorf, D. Neider, and K. Kersting, "Learning to break deep perceptual hashing: The use case NeuralHash," in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 58-69, doi: 10.1145/3531146.3533073.

[25] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, Apr. 2004. [Online]. Available: https://www.cns.nyu.edu/pub/lcv/wang03-reprint.pdf

[26] T. B. Brown *et al.*, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877-1901. [Online]. Available: https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[27] A. Radford *et al.*, "Learning Transferable Visual Models From Natural Language Supervision," in *Proc. International Conference on Machine Learning*, PMLR, vol. 139, 2021, pp. 8748-8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[28] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual Instruction Tuning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 34892-34916. [Online]. Available: https://papers.nips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html

[29] MITRE ATT&CK, "Obfuscated Files or Information: Steganography, T1027.003." [Online]. Available: https://attack.mitre.org/techniques/T1027/003/

[30] MITRE ATT&CK, "Data Obfuscation: Steganography, T1001.002." [Online]. Available: https://attack.mitre.org/techniques/T1001/002/

[31] MITRE ATT&CK, "Exfiltration Over C2 Channel, T1041." [Online]. Available: https://attack.mitre.org/techniques/T1041/

[32] MITRE ATT&CK, "Exfiltration Over Web Service, T1567." [Online]. Available: https://attack.mitre.org/techniques/T1567/

[33] MITRE D3FEND, "Network Traffic Policy Mapping, D3-NTPM." [Online]. Available: https://d3fend.mitre.org/technique/d3f:NetworkTrafficPolicyMapping/

[34] MITRE D3FEND, "File Analysis, D3-FA." [Online]. Available: https://d3fend.mitre.org/technique/d3f:FileAnalysis/

[35] MITRE D3FEND, "Content Filtering, D3-CF." [Online]. Available: https://d3fend.mitre.org/technique/d3f:ContentFiltering/

[36] MITRE D3FEND, "Data Inventory, D3-DI." [Online]. Available: https://d3fend.mitre.org/technique/d3f:DataInventory/

[37] MITRE D3FEND, "Access Mediation, D3-AMED." [Online]. Available: https://d3fend.mitre.org/technique/d3f:AccessMediation/

[38] National Institute of Standards and Technology, *The NIST Cybersecurity Framework (CSF) 2.0*, Feb. 2024. [Online]. Available: https://www.nist.gov/cyberframework

[39] National Institute of Standards and Technology, *Security and Privacy Controls for Information Systems and Organizations*, NIST SP 800-53 Rev. 5, Sep. 2020. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final

[40] OWASP Foundation, "Application Security Verification Standard (ASVS)." [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/

[41] OWASP Foundation, "Web Security Testing Guide (WSTG)." [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/

[42] OWASP Foundation, "Testing WebSockets," *Web Security Testing Guide*, v4.2. [Online]. Available: OWASP WSTG WebSocket testing chapter