

AV quality enhancement — Q&A session notes

From AV_QUALITY_QA_SESSION.md

Field	Value
Author	Anton Afanasyeu
Revision	R1
Creation date	2026-05-21
Last modification date	2026-05-21
Co-authored	
Severity	medium
State	pending review
Document type	internal

Table of contents

- 0. Receiver A/V presets (product, May 2026)
- 1. User requirements (preliminary conditions)
- 2. Mode overview
- 3. Video — method comparison
- 4. Audio — method comparison
- 5. Single vs 2-frame vs stream — decision matrix
- 6. PTS/DTS reassembly (2.3)
- 7. Pipeline placement: sender (2.1) vs receiver (2.2)
 - User question
 - Assessment
 - Preference (consensus from session)
- 8. Named algorithm pipelines (best practices)
 - A. Receiver — live video cast (default)
 - B. Receiver — movie cast (+1 frame delay)
 - C. Sender — optional compression prep only
 - D. Avoid for realtime in this project
- 9. Ranked preferences (quality benefit vs implementation in android cast)
- 10. Multithreading pipeline (target architecture)
- 11. Short answers to numbered conditions
- 12. Bottom line
- 13. Other work in same project session (context only)
- 14. Open decisions when implementation starts

Source: AV_QUALITY_QA_SESSION.md — section links work in PDF viewers that support internal anchors.

Research and recommendations for **movie cast** vs **video cast** quality enhancement in the Android Cast project. Combines user requirements, comparison tables, algorithm choices, placement (sender vs receiver), and fit with the current codebase.

Status: Receiver experimental presets in **Developer settings** — audio PCM chain + video GLES post-decode (immersive / HDR shaders).

Related code: app/.../receiver/av/*, ReceiverVideoGLESRenderer, AudioDecoder, VideoDecoder / LibvpxVideoDecoder, ReceiverPlaybackActivity, DeveloperSettingsActivity.

0. Receiver A/V presets (product, May 2026)

Developer-only combo in **Developer settings** (not the main cast drawer):

Video	Audio	Pipeline today
Default — decode→Surface passthrough	Default — decode→AudioTrack passthrough	Hook only; no DSP / no GLES
Immersive — GLES edge-aware denoise + mild sharpen	Bass boost — shelf below ~150 Hz, gentle treble cut	PCM + GLES; intensity 0–10
HDR — GLES Reinhard-lite + saturation	Immersive — clarity + noise gate + limiter (no ML v1)	RNNoise-class deferred; see §8A
	Dolby 5.1 / 7.1 — crossfeed + short delays for stereo headsets	Spatialization stub; intensity 0–10

Immersive audio (this pass): clearer, wider perceived loudness without extra hiss — high-pass (~90 Hz), envelope noise gate, mild presence lift, soft ceiling. Not “cinema reverb”; aim for intelligibility on cast compression artifacts. Scale all stages by **intensity/10**.

Intensity: 0 = bypass effect (passthrough samples); 10 = maximum for that preset.

Integration: ReceiverAvRuntime → AudioDecoder after AAC decode; video decoder → intermediate OES SurfaceTexture → GLES shader → TextureView (ReceiverVideoGLESRenderer) when preset is immersive/HDR; default preset uses direct decode→TextureView Surface.

1. User requirements (preliminary conditions)

- Content is either **movie cast** or **video cast** (live screen/camera); **both** must be supported with different tuning.
 - Both** video and audio need quality enhancement.
 - Two processing shapes:
 - 2.1 Temporal (N=2 frames/windows):** small gap between sequential video frames → predict/enhance using deltas between frames; same idea for audio (overlapping buffers).
 - 2.2 Single frame/chunk:** static image from sink/source; one video frame; one audio PCM chunk.
 - 2.3 Reassembly:** output stream must keep **original PTS/DTS** (or fixed delay with uniform compensation); avoid changing frame/sample count on live paths unless explicitly designed.
- Comparison dimensions requested:** implementation details, difficulty, realtime on modern ARM (GPU Android/Linux where applicable), input vs output quality, single vs 2-frame vs stream (with delay), multithreading.

2. Mode overview

Mode	Video input	Audio input	Typical delay	Best when
2.2 Single	One buffer / frame	10–40 ms PCM chunk	0–1 frame (~0–33 ms @ 30 fps)	Live video cast , UI, games
2.1 Two-frame (N=2)	Pair with small temporal gap	Overlapping windows	~1 frame (+ overlap)	Motion; mild temporal help
Stream (3–8)	Short FIFO	Ring + STFT state	100–400 ms+	Movie cast , VOD-like
Heavy stream ML	Long context	Streaming neural	200 ms–2 s+	Offline / non-live only

Quality (typical): stream/temporal > 2-frame > single frame

Latency (best first): single frame > 2-frame > stream

Policy split:

Cast type	Video	Audio
Video cast (live)	2.2 default; optional light 2.1 (+1 frame max)	2.2 (RNNoise + limiter)
Movie cast	2.1 or 3–4 frame TNR / MCTF-lite	2-window spectral or DeepFilterNet2 streaming

3. Video — method comparison

Method	Implementati on	Difficulty	ARM @ 720p30	ARM @ 1080p30	GPU (Androi d/Linux)	Quality vs input	PTS/DTS	Multithread
Single: bilateral / guided filter denoise	GLES/Vulkan on YUV; NEON fallback	Low	Yes	Yes (tuned)	High	Small–mediu m	Trivial	Yes (tiles)

Single: tone / CLAHE-lite	LUT + luma histogram tiles	Low–med	Yes	Yes	Medium	Medium in flat scenes	Trivial	Yes
Single: light SR (ESPCN-class tiny CNN)	TFLite/NCNN → NNAPI/GPU	Medium	Maybe	Tight	High	Medium (edges/text)	Trivial	Yes (async infer)
Single: heavy SR (Real-ESRGAN, SwinIR)	Large CNN	High	No	No	BW-bound	High but slow	Trivial	Limited
2-frame: DIS / Farneback flow + warp blend	Flow on downscaled luma → warp prev	Medium	Yes	Maybe	Med–high	Medium	+1 frame	Yes
2-frame: residual / delta enhance	enhanced = $f(\text{curr}) + \alpha \cdot (\text{curr} - \text{warped_prev})$	Medium	Yes	Maybe	Medium	Med–high on repetitive motion	+1 frame	Yes
2-frame: RIFE-lite / FILM-small	Interpolate between frames	Med–high	Borderline 720p	Unlikely 1080p30	High if custom	High for large gaps	Timing care	Yes
Stream: IIR TNR / motion-mask accumulate	3–4 frame buffer	Medium	Yes	Maybe	High	High static; trail risk	+2–3 frames	Yes
Stream: ML temporal (VRT/VRT-classes)	Stateful temporal CNN	Very high	No live	No	Medium	Very high	Buffer delay	Some

4. Audio — method comparison

Method	Implementation	Difficulty	ARM realtime 48 kHz	GPU	Quality vs input	PTS/DTS	Multithread
Single: DC block + EQ + limiter	Biquads per chunk	Low	Yes	Low	Small	Preserve samples	Yes
Single: RNNoise / DeepFilterNet (10 ms)	TFLite/ONNX streaming	Medium	Yes mono; maybe stereo	NNAPI	High for noise	+10–20 ms	Yes (infer thread)
2-window: Wiener / min-stats STFT	20 ms, 50% hop	Medium	Yes	Low	Med–high	+1 hop (~10 ms)	Yes
2-window: PLC	Predict from last 2 frames	Low–med	Yes	Low	High only on loss	Keep clock	Yes
Stream: multiband + AGC	Ring buffer envelopes	Low–med	Yes	Low	Medium intelligibility	5–30 ms	Yes
Stream: DeepFilterNet2	Stateful STFT+GRU	Med–high	Yes modern ARM	NNAPI	High	20–40 ms	Yes

5. Single vs 2-frame vs stream — decision matrix

Criterion	Single (2.2)	2-frame (2.1)	Stream (3–8)
Implementation complexity	Lowest	Medium	Highest
Encoder integration	Easiest	+1 frame pipeline	Delay queue
ARM realtime 720p30	Best	Good	Good (TNR); poor heavy ML
ARM realtime 1080p30	Good	Marginal	Marginal
GPU fit	Best	Good (warp)	Good (TNR)
Quality gain (video)	Low–medium	Medium + motion	High static / medium action
Quality gain (audio)	Medium	Med–high	High steady noise
Artifact risk	Banding, oversharpen	Ghosting, warp error	Trails, smear
PTS/DTS	Trivial	Fixed +1 frame delay	Uniform delay all outputs
Multithread	Capture ■ GPU ■ encode	Flow ■ enhance	Buffer thread + pool
Movie cast	OK quick polish	Recommended	Recommended if delay OK

Video cast	Recommended	Optional 1-frame	Not recommended live
------------	-------------	------------------	----------------------

6. PTS/DTS reassembly (2.3)

Approach	Description	Modes
In-place payload replace	Same PTS/DTS, duration; swap enhanced payload	Single
Delay compensation	Queue enhanced frames; release with original PTS	2-frame, stream
Composition timestamp	presentationTimeUs from source → enhanced buffer (MediaCodec)	All on Android
Audio	Keep sample count constant → PTS unchanged	Prefer for live

Rule: Do not change frame/sample count on live cast unless explicit insert/drop policy exists.

7. Pipeline placement: sender (2.1) vs receiver (2.2)

User question

- **2.1 Sender:** enhancement between capture (camera/screen) and encoder.
- **2.2 Receiver:** enhancement between decoded stream and renderer.
- **User intuition:** 2.1 has limited value because encoder lossy output destroys pre-enhance detail; increases sender load.

Assessment

Claim	Verdict
Encoder “kills” sharpen/SR/local contrast pushed pre-encode	True
2.1 is pointless	False for denoise-before-encode (helps bitrate/artifacts at same rate)
2.1 burns sender CPU for little gain	Often true on thermally limited phones

Preference (consensus from session)

Placement	Role
Receiver (2.2)	Primary perceived quality — deblock, denoise, mild sharpen; optional 2-frame for movie
Sender (2.1)	Optional compression prep only (bilateral/NLMeans denoise), not main beauty pipeline
Both	Receiver polishes; sender light or off on live video cast

Current app hooks:

Pipeline	Integration point	Difficulty (1=easy, 5=hard)
2.2 Video receiver	After VideoDecoder / LibvpxCapableVideoDecoder, before TextureView	3–4 (new GLES: decode → FBO → display)
2.2 Audio receiver	AudioDecoder PCM before AudioTrack	2
2.1 Video sender	Between capture and VideoEncoder.prepare() Surface	4–5 (intermediate EGL Surface, rotation)
2.1 Audio sender	Before AudioEncoder	2–3
2-frame video	Ring buffer +1 frame delay	+1 on above

Recommended implementation fork: decoded-buffer → **GLES** → Surface (consistent for MediaCodec and libvpx VP9), not only TextureView overlay.

8. Named algorithm pipelines (best practices)

A. Receiver — live video cast (default)

Video (2.2, GPU-first):

1. Post-decode **deblocking filter** (lite H.264-style / mild dering; VP9 loop filter already in decode)
2. **Guided filter** or **bilateral filter** (edge-preserving denoise) on luma
3. Mild **unsharp mask** (limit on blocky regions)
4. Optional: **gamma** / **BT.1886** + **CLAHE-lite**

Audio (2.2):

1. **RNNoise** (10 ms frames)
2. **Limiter** + high-pass (DC removal)
3. Optional: **SpeexDSP** preprocess (AGC, mild denoise)

B. Receiver — movie cast (+1 frame delay)

Video (2.1):

1. **DIS** or **Farneback** optical flow (downscaled)
2. **MCTF-lite** or **IIR temporal denoise** on stable regions
3. Spatial chain from A (guided/bilateral + mild unsharp)

Audio (2.1):

1. **STFT overlap-add** + **Wiener filter**, or **DeepFilterNet2** (streaming)
2. **Multiband compressor**

C. Sender — optional compression prep only

Video: **Bilateral** or fast **NLMeans** only when bitrate-starved — **no** Real-ESRGAN / strong sharpen pre-encode.

Audio: **SpeexDSP preprocess** or light **RNNoise** before AAC.

D. Avoid for realtime in this project

Algorithm	Reason
Real-ESRGAN, SwinIR, BasicVSR++	Too heavy ARM @ 720p–1080p30 live
RIFE / FILM @ 1080p30 full rate	Borderline; movie-only, lower rate
BM3D, VRT/RVRT	Offline-tier
Heavy 2.1 sender SR before VP9	Encoder erases; thermals

9. Ranked preferences (quality benefit vs implementation in android cast)

Descending order — build in this sequence:

Rank	Choice	Placement	Core algorithms	Quality benefit	Impl difficulty (this app)
1	Receiver video 2.2 (GPU)	2.2	Guided/bilateral + deblock + mild unsharp	High vs blocky stream	3–4
2	Receiver audio 2.2	2.2	RNNoise + limiter (+ SpeexDSP)	High for noise	2
3	Receiver video 2.1 (movie)	2.2 + 1f delay	DIS/Farneback + MCTF-lite / IIR TNR + spatial	Higher static/grain	4
4	Receiver audio 2.1 (movie)	2.2	DeepFilterNet2 or Wiener STFT	Medium–high	3
5	Sender audio light prep	2.1	SpeexDSP / RNNoise → AAC	Low–medium	2–3
6	Sender video denoise only	2.1	Bilateral / fast NLMeans (not SR)	Low–medium (bitrate)	4–5
7	Sender video 2.1 beauty	2.1	Flow + enhance pre-encode	Low (washed by codec)	5
8	Heavy SR either side	either	Real-ESRGAN class	High offline only	5, not realtime

10. Multithreading pipeline (target architecture)

[Capture / Network decode] → queue → [Enhance worker(s)] → queue → [Encode / Render]

↑
N=2: previous frame
Stream: ring buffer

Stage	Threads	Notes
Capture / decode callback	1	Non-blocking
Enhance (GPU)	1–2	Double-buffered FBO / AHardwareBuffer
CPU fallback (NEON)	2–4	Tile-based
ML audio	1	Async from video
Encode / display	1	MediaCodec async; TextureView main

Golden rule: max **one frame** enhancement queue for **video cast**; **2–4 frames** for **movie cast**.

Receiver already has `videoDecodeThread` in `ReceiverCastService` — natural host for GL enhance stage.

11. Short answers to numbered conditions

#	Requirement	Preference
1	Movie or video cast	Different latency policies; same codebase, different presets

2	Process both A+V	Shared audio chain; video policy switches by cast mode
2.1.1	2-frame video deltas	DIS/Farneback + residual — not full RIFE on ARM live
2.1.2	2-frame audio	Overlap-add STFT or streaming DeepFilterNet
2.2.1	Single image video	GPU bilateral/guided + tone ; tiny SR @ 720p only if ever
2.2.2	Single chunk audio	RNNoise + limiter
2.3	PTS/DTS reassembly	In-place or fixed-delay queue; no timeline shift on live

12. Bottom line

Goal	Best option
Lowest latency + ARM-safe	Single frame receiver (2.2) + GPU shaders
Best quality per ms delay	2-frame receiver (2.1) + 2-window/stream audio
Best absolute quality (movie)	Small stream buffer (3–4) video TNR + streaming audio ML
Avoid realtime cast	Heavy per-frame SR, RIFE @ 1080p30, deep temporal video CNN
Placement	Receiver-first ; sender denoise-only at most

User's encoder observation is correct for “beauty” enhancement pre-encode; denoise-before-encode remains the only strong sender-side exception.

13. Other work in same project session (context only)

Unrelated to AV enhancement but done in parallel on this branch:

- OTA optional .otabundle.zip, crash reporter (:crashwatcher), HEADER.txt automation, PHP crash backend — see docs/OTA.md, docs/CRASH_REPORTER.md, examples/crash_reporter/.

14. Open decisions when implementation starts

- Enhance on **display path** (TextureView) vs **decode buffer** → **GL ES** → **Surface** (recommended: latter).
- User-toggle vs automatic preset for movie vs video cast.
- Whether sender **denoise-only** preset is worth thermals on low-end devices.
- GPU API: **OpenGL ES 3.x compute** vs **Vulkan** (ES3 often faster to integrate in Android media apps).

Document generated from AV quality Q&A session. Update as prototypes prove FPS/latency on target devices.