H.264 encoding performance on Raspberry Pi 5–series computers

Colophon

Release	1
Build date	05/05/2026
Build version	25aadcd28dac

Legal disclaimer notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME (“RESOURCES”) ARE PROVIDED BY RASPBERRY PI LTD (“RPL”) “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage (“High Risk Activities”). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL’s Standard Terms. RPL’s provision of the RESOURCES does not expand or otherwise modify RPL’s Standard Terms including but not limited to the disclaimers and warranties expressed in them.

Document version history

Release	Date	Description
1	28 Apr 2026	Initial release

Scope of document

This document applies to the following Raspberry Pi products:

Single Board Computers / SBCs

Pi Zero			Pi Zero 2		Pi 1				Pi 2	Pi 3			Pi 4	Pi 5
-	W	H	W	WH	A	B	A+	B+	B	A+	B	B+	B	-
														✓

Compute Modules

CM0	CM1	CM3	CM3+	CM4	CM4S	CM5
						✓

Keyboard Computers

Pi 400	Pi 500	Pi 500+
	✓	✓

Abstract

Raspberry Pi 5 does not feature a hardware H.264 encoder, relying instead on the `libx264` software encoder running on its 2.4GHz quad-core Arm Cortex-A76 processor. This paper evaluates the encoding performance of Raspberry Pi 5 against the hardware H.264 encoder found on Raspberry Pi 4, using rate-distortion analysis with both peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM) quality metrics.

The key finding is that Raspberry Pi 5's software encoder can produce 1080p30 H.264 video in real time, using as little as 60% of a single CPU core in low-latency mode, leaving the remaining cores free for other tasks. High-quality mode delivers 2–3dB better PSNR than Raspberry Pi 4's hardware encoder at practical bitrates (500kb/s to 5Mb/s), while requiring approximately 100–150% of a single core. Extrapolating this, a performance target of 1080p60 using the high-quality preset is achievable at these bitrates, something that would not be possible with the hardware H.264 encoder.

Both software presets provide significantly better bitrate accuracy than the hardware encoder, which exhibits a hard bitrate ceiling of approximately 6–14Mb/s, depending on content complexity. It is also unable to produce bitrates below approximately 240kb/s.

What is H.264?

H.264 (also known as MPEG-4 Part 10 or AVC) is one of the most widely deployed video compression standards. It reduces the data required to process video by exploiting spatial redundancy within frames and temporal redundancy between consecutive frames. The standard is used across streaming, broadcasting, video conferencing, surveillance, and recording.

How it works

H.264 divides each video frame into macroblocks (16 × 16 pixel regions) and then compresses them in three main stages:

Prediction

The encoder predicts the content of each macroblock using already-encoded data. *Intra-prediction* uses neighbouring blocks within the same frame to predict spatial patterns (edges, gradients, flat regions). *Inter-prediction* (motion compensation) searches previous and/or future frames for similar regions and encodes a motion vector plus a small residual difference. The better the prediction, the less residual data needs to be encoded.

Transformation and quantisation

The residual (the difference between the predicted and the actual pixel data) is transformed using an integer discrete cosine transform (DCT), converting spatial-domain pixel differences into frequency-domain coefficients. Quantisation then reduces the precision of these coefficients; this is the primary lossy step. Higher quantisation discards more information, producing smaller outputs with lower quality.

Entropy coding

The quantised coefficients, motion vectors, and other syntax elements are losslessly compressed using either context-adaptive binary arithmetic coding (CABAC) or context-adaptive variable-length coding (CAVLC). This final stage removes statistical redundancy from the data without any further quality loss.

Key definitions

CABAC
: Context-adaptive binary arithmetic coding — an entropy coding method that achieves 10–15% better compression than CAVLC.

CAVLC
: Context-adaptive variable-length coding — a simpler, faster entropy coding method.

I-frame
: Intra-coded frame. Encoded using data from within the frame itself, without referencing other frames. Acts as a random access point (keyframe) in the bitstream.

P-frame
: Predictive frame. References one or more preceding frames to encode motion-compensated differences. Typically much smaller than I-frames.

B-frame
: Bi-predictive frame. References both preceding and succeeding frames for prediction, achieving higher compression than P-frames. Requires the encoder to buffer future frames before encoding, adding latency.

Bitstream
: The compressed output of the encoder; a sequence of encoded frames and associated metadata.

Bitrate
: The number of bits per second used by the encoded video. Lower bitrates mean smaller files, reduced bandwidth, and typically lower quality.

Frame types and the GOP structure

An H.264 bitstream is structured as a sequence of I-, P-, and B-frames. I-frames are inserted at regular intervals (the *keyframe interval*) and when the encoder detects a scene change, as inter-prediction becomes ineffective across scene boundaries. The sequence of frames between consecutive I-frames is known as a group of pictures (GOP).

Between I-frames, the bitstream consists of P-frames and optionally B-frames. Because P- and B-frames only encode differences relative to reference frames, they are substantially smaller than I-frames. Longer GOPs therefore produce lower average bitrates, but at the cost of reduced seek granularity and error resilience.

B-frames provide the best compression but introduce encoding latency: a B-frame that references a future frame cannot be output until that frame has been encoded. For this reason, real-time applications such as live camera streaming typically disable B-frames entirely.

H.264 encoding is inherently lossy — the quantisation step permanently discards information. The degree of loss is controllable: at one extreme, very high bitrates can produce outputs that are visually indistinguishable from the original; at the other, aggressive compression produces visible artefacts (blocking, blurring, colour banding), but at a fraction of the data rate.

Controlling bitrate and quality

An H.264 encoder has two main controls: how aggressively to quantise the frame (which directly trades quality for bitrate), and how much effort to spend finding the most efficient way to represent each frame.

Rate control

Rate control determines how the encoder allocates bits across frames. There are several modes:

Constant rate factor (CRF)

The encoder targets a consistent perceptual quality across the entire video by varying the bitrate as needed. A single parameter controls the quality level — higher CRF values produce lower quality and lower bitrate. This is the preferred mode for offline or file-based encoding, where the output bitrate does not need to be constrained.

Constant quantisation parameter (CQP)

CQP is similar to CRF, but it applies a fixed quantisation parameter to every frame without any rate adaptation. This provides predictable quality on a per-frame basis, but can also produce large bitrate variations between scenes of differing complexity.

Average bitrate (ABR)

The encoder targets a specified average bitrate over the duration of the video. It varies the quantisation per frame to stay near the target, compressing complex scenes more aggressively and allocating more bits to simpler scenes. This is the mode used in the rate-distortion tests in this paper.

Constant bitrate (CBR)

A stricter form of ABR, in which the encoder attempts to maintain a near-constant bitrate at all times. This is typically required for live-streaming or broadcasting applications, where the transport channel has a fixed capacity.

In all bitrate-targeted modes (ABR or CBR), the encoder adjusts quantisation internally to meet the target. Requesting a lower bitrate forces coarser quantisation, which discards more information and reduces quality.

Encoding efficiency parameters

These parameters control how hard the encoder works to find an efficient representation. Enabling more advanced tools improves quality at a given bitrate (or equivalently, reduces the bitrate needed for a given quality), but this increases CPU usage and may add latency.

Entropy coding

CABAC achieves approximately 10–15% better compression than CAVLC for the same quality. CABAC is enabled by default in the H.264 standard's 'Main' and higher profiles, but is not available in the 'Baseline' profile.

B-frames

B-frames reference both past and future frames, allowing for more efficient compression than P-frames alone. However, they require the encoder to buffer future frames before encoding, which adds latency. For real-time applications such as live camera streaming, B-frames are typically disabled.

GOP structure

Increasing the interval between I-frames allows more of the bitstream to consist of the more efficient P- and B-frames. Longer GOPs reduce bitrate but can affect seek performance and error resilience, as the decoder must process frames back to the nearest I-frame to begin playback.

Motion estimation and subpixel refinement

More thorough motion-search algorithms and subpixel refinement produce better motion vectors, improving inter-frame prediction at the cost of increased processing and search time.

Partition and transform modes

Enabling smaller partition sizes (e.g. i4×4) and 8×8 DCT allows the encoder to better adapt to local image detail, improving compression efficiency for textured regions.

H.264 on Raspberry Pi 5

As there are no hardware H.264 encoders on the BCM2712 chip, all encoding is done through the software. Raspberry Pi Ltd uses the `libav` library to provide encoding capability in Raspberry Pi OS. This library is highly optimised for the Arm platform, and makes extensive use of the Arm Neon coprocessor to improve encode speeds. It also offers a considerable number of options to allow users to customise the encoding process. Some of these options are examined below.

The `rpicam-vid` camera application is our standard tool for capturing video on Raspberry Pi devices. On Raspberry Pi 5, it uses the `libx264` software encoder through the `libav` library to encode the camera's processed output into H.264 in real time. It exposes a `-b` (bitrate) parameter that sets the ABR target, and a `--low-latency` flag that selects a reduced-CPU encoder configuration. The specific encoder parameters used by each mode are detailed in the testing sections below.

`rpicam-vid` uses the following code to set up the `libx264/libav` encoder, and should be used as a reference when looking at specific encoder options in the following document: https://github.com/raspberrypi/rpicam-apps/blob/main/encoder/libav_encoder.cpp

Instructions for rebuilding this library are available in the `rpicam_apps` GitHub repository, making it easy to try different default settings.

Testing

The easiest way to see the encoding process in action is to attach a Raspberry Pi Camera Module to a Raspberry Pi 5 and use `rpicam-vid` to encode the resulting images into an H.264 bitstream. However, in order to provide an accurate assessment and comparison of encode quality, the tests included here use a pre-existing set of video frames that are encoded using `ffmpeg` , which uses the same `libav` and `libx264` code path as `rpicam-vid` . By configuring `ffmpeg` with the same `libx264` parameters that `rpicam-vid` uses, the encoding path will be identical and produce results that are directly comparable. Using a fixed set of source frames also ensures that every encoder configuration is tested against exactly the same input, and that the measurements reflect encoder performance alone, without any influence from camera capture or image signal processing (ISP).

The system used in the test comprises a Raspberry Pi 5 with 8GB of RAM, running Raspberry Pi OS Trixie and the Raspberry Pi desktop. To assess encode quality, each encoded video is compared against the original uncompressed frames using two complementary metrics: PSNR and SSIM.

Note

The **peak signal-to-noise ratio (PSNR)** measures the ratio between the maximum possible pixel value and the mean squared error between the original and encoded frames. Expressed in decibels (dB), higher values indicate better fidelity, with typical values for high-quality encodes ranging from 30–50dB. PSNR is straightforward to compute but does not always correlate well with perceived visual quality, as it weights all pixel errors equally regardless of their visibility.

The **structural similarity index measure (SSIM)** evaluates image quality based on perceived changes in structural information, luminance, and contrast. It produces a value between 0 and 1, where 1 indicates a perfect match. SSIM generally correlates better with human perception than PSNR, as it is more sensitive to structural distortions (such as blurring or blocking) and less affected by uniform shifts in brightness or contrast.

Test clip

The source material is a clip from a Sony PXW-FS7M2 professional camera, originally recorded at 3840 × 2160 59.94 frames per second in XAVC-I 4:2:2 10-bit format. This was downscaled and converted to 1920 × 1080 YUV420P 8-bit raw frames (560 frames, I420 planar layout) for use as the uncompressed reference input for the encoders.

Extracts from the test clip can be found in the Appendix.

Encoder configurations

`rpicam-vid` provides two encoder configuration presets: *high quality* and *low latency*. The differences are summarised in the table below:

Parameter	Low latency	High quality
libx264 preset	ultrafast	superfast
Tune	zerolatency	*(none)*
B-frames	0 (disabled)	1 (enabled)
CABAC	Disabled	Enabled
Deblocking	Disabled	Enabled
Partitions	None	i8x8, i4x4
8×8 DCT	Disabled	Enabled
Trellis	0 (off)	0 (off)
Subpixel ME	0 (full-pel only)	0 (full-pel only)
References	1	1
Motion estimation	dia	dia
Weighted prediction	Disabled	Disabled

Parameter	Low latency	High quality
Scene cut detection	Disabled	Disabled
Rate control lookahead	0	0

For testing, these configuration presets are replicated in `ffmpeg` , providing the two software encoder configurations for Raspberry Pi 5. A third configuration uses Raspberry Pi 4's hardware encoder for comparison:

Configuration	Encoder	Description
Raspberry Pi 5 low-latency mode	libx264 (software)	Ultrafast preset with zerolatency tune. Disables B-frames, CABAC, deblocking, and most analysis. Designed for real-time, low-latency applications such as live camera streaming.
Raspberry Pi 5 high-quality mode	libx264 (software)	Superfast preset with B-frames and i8x8/i4x4 partitions enabled. Retains CABAC entropy coding and deblocking. Higher quality at the expense of increased CPU usage and latency.
Raspberry Pi 4 hardware encoder	h264_v4l2m2m (hardware)	Hardware encoder accessed via the V4L2 memory-to-memory interface. Fixed-function encoder with no user-configurable presets.

Command lines

Raspberry Pi 5 low-latency mode

Code

```
ffmpeg -y -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30 \
  -i <input.yuv> -an -c:v libx264 -preset ultrafast \
  -tune zerolatency -refs 1 \
  -x264-params "weightp=0:weightb=0:me=dia:scenecut=0:\
  rc-lookahead=0:mixed-refs=0:merange=16:subme=0" \
  -b:v <bitrate>k <output.mp4>
```

Raspberry Pi 5 high-quality mode

Code

```
ffmpeg -y -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30 \
  -i <input.yuv> -an -c:v libx264 -preset superfast -bf 1 \
  -x264-params "partitions=i8x8,i4x4:weightp=0:weightb=0:\
  me=dia:scenecut=0:rc-lookahead=0:mixed-refs=0:\
  merange=16:subme=0" \
  -b:v <bitrate>k <output.mp4>
```

Raspberry Pi 4 hardware encoder

Code

```
ffmpeg -y -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30 \
  -i <input.yuv> -an -c:v h264_v4l2m2m \
  -b:v <bitrate>k <output.mp4>
```

CPU usage measurement

CPU usage for Raspberry Pi 5's software encoder is measured by recording the total CPU time (user + system, across all threads) consumed by `ffmpeg` for each encode, using `resource.getrusage()` . This is then normalised to express the result as a percentage of a single core required to sustain 30fps encoding:

$$\text{CPU @30 fps} = \left(\frac{\text{CPU time}}{\text{number of frames}}\right) \times 30 \times 100\%$$

A value of 100% means exactly one full core is required; values above 100% indicate that multiple cores are required for real-time encoding. Raspberry Pi 5 has 4 cores, and `libx264` distributes work across all available cores by default. Note that there may be a discrepancy of 10–20% in these measurements, owing to the nature of a live system.

CPU usage is not reported for Raspberry Pi 4's hardware encoder, as the encoding is performed by dedicated hardware with negligible CPU overhead.

Results

The following tables and graphs present encoding results for the test clip (1920 × 1080, 560 frames).

Rate-distortion data

Raspberry Pi 5 low-latency mode

Target (kb/s)	Actual (kb/s)	PSNR (dB)	SSIM	CPU @ 30fps
100	130	25.63	0.839	68.6%
146	201	28.32	0.853	65.5%
213	257	29.40	0.869	62.0%
311	346	32.69	0.899	62.5%
454	481	35.44	0.926	63.4%
663	682	37.40	0.944	63.1%
968	983	39.08	0.956	64.8%
1414	1418	40.54	0.965	65.6%
2064	2053	42.02	0.972	66.9%
3014	2984	43.50	0.978	68.7%
4401	4328	44.81	0.982	70.6%
6426	6307	46.10	0.985	74.4%
9382	9157	47.29	0.988	78.0%
13698	13381	48.40	0.990	83.6%
20000	19672	49.44	0.992	89.5%

Raspberry Pi 5 high-quality mode

Target (kb/s)	Actual (kb/s)	PSNR (dB)	SSIM	CPU @ 30fps
100	140	28.42	0.856	98.2%
146	188	30.28	0.885	99.5%
213	253	33.14	0.915	100.9%
311	336	36.64	0.943	102.2%
454	466	38.74	0.958	103.1%
663	666	40.37	0.967	104.6%
968	962	41.81	0.974	106.6%

Target (kb/s)	Actual (kb/s)	PSNR (dB)	SSIM	CPU @ 30fps
1414	1400	43.10	0.979	108.4%
2064	2034	44.21	0.982	112.5%
3014	2963	45.22	0.985	117.2%
4401	4320	46.16	0.987	122.9%
6426	6309	47.03	0.989	128.4%
9382	9253	47.82	0.990	135.3%
13698	13539	48.56	0.991	143.8%
20000	19836	49.27	0.992	152.7%

Raspberry Pi 4 hardware encoder

Target (kb/s)	Actual (kb/s)	PSNR (dB)	SSIM
100	238	30.24	0.889
146	250	30.66	0.893
213	285	31.39	0.901
311	317	31.84	0.907
454	459	33.63	0.924
663	686	37.11	0.946
968	992	39.49	0.960
1414	1458	41.35	0.969
2064	2109	42.87	0.976
3014	3081	44.18	0.981
4401	4429	46.17	0.986
6426	5892	46.95	0.988
9382	5939	47.01	0.988
13698	5948	47.02	0.988
20000	5953	47.03	0.988

Rate-distortion graphs

Figure 1.
PSNR rate-distortion curves for all three encoder configurations. The x-axis shows actual bitrate on a logarithmic scale. Raspberry Pi 5 high-quality mode consistently achieves the highest PSNR, while Raspberry Pi 4's hardware encoder plateaus above 6Mb/s.

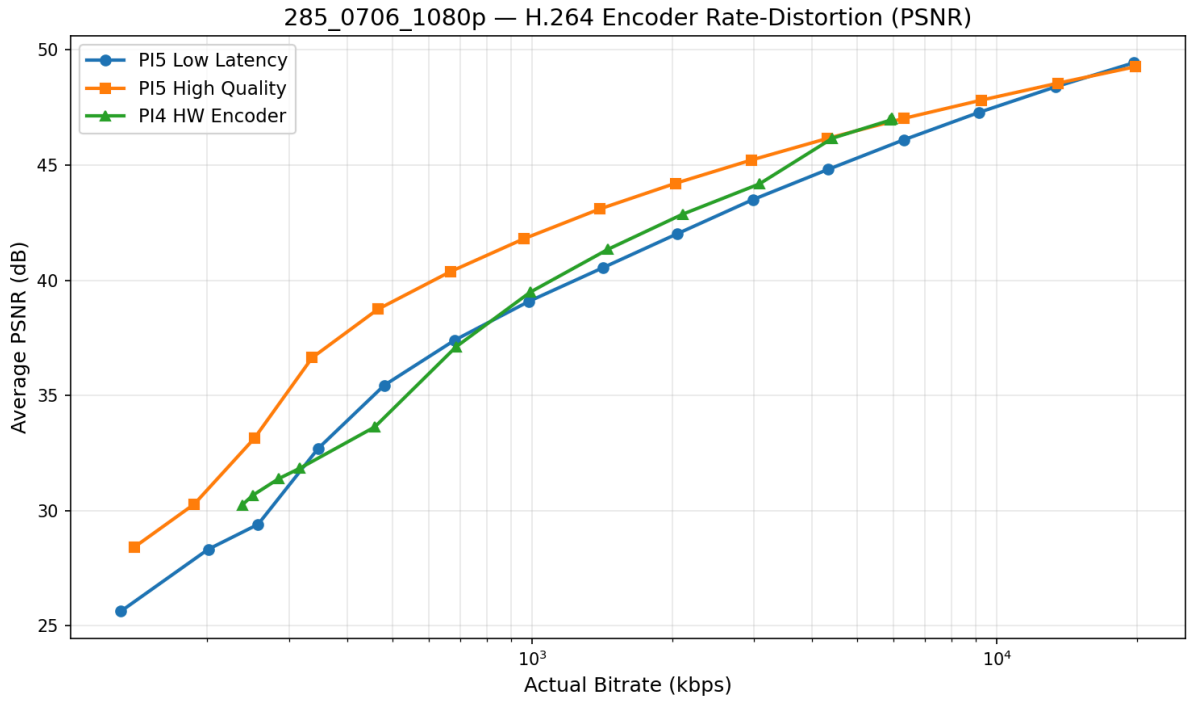


Figure 2.
SSIM rate-distortion curves for all three encoder configurations. SSIM correlates more closely with perceived visual quality than PSNR, with values closer to 1.0 indicating better quality. The distinction between Raspberry Pi 5 high-quality mode and the other encoders is even more pronounced in SSIM at low-to-mid bitrates.

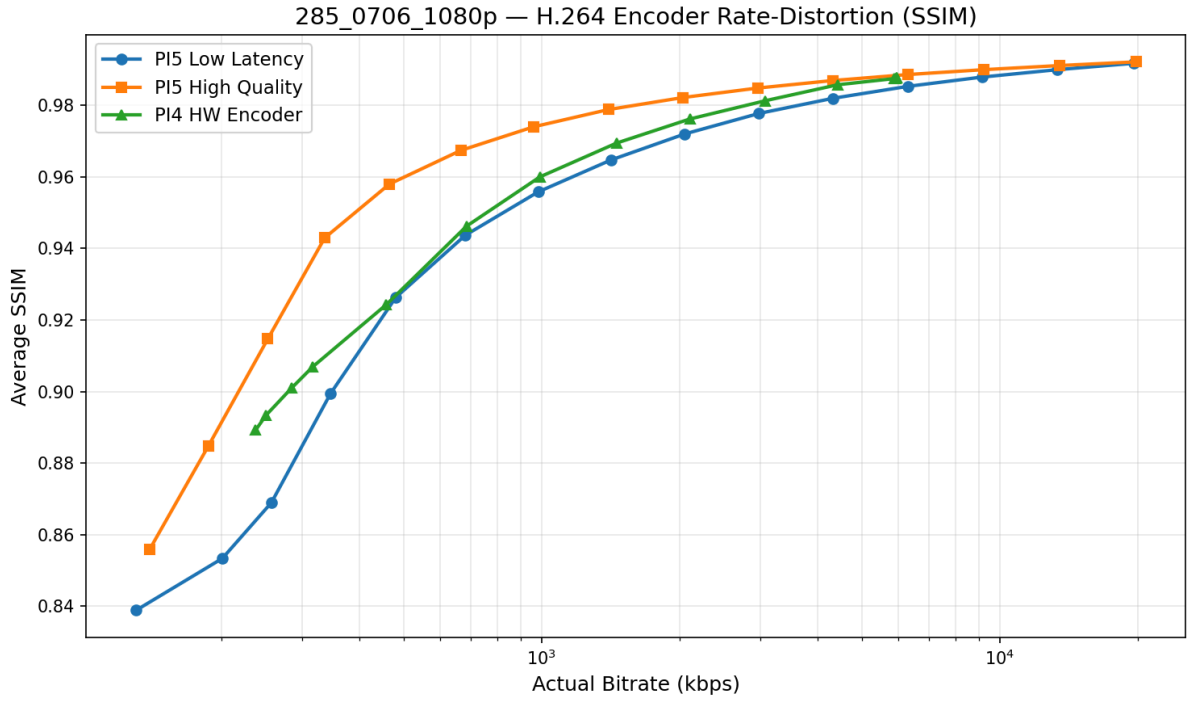


Figure 3.

Target versus actual bitrate accuracy for each encoder. The dashed line represents ideal 1:1 tracking. Both Raspberry Pi 5 presets (high quality and low latency) track closely across the full range, while Raspberry Pi 4's hardware encoder overshoots at low targets and saturates above 6Mb/s.

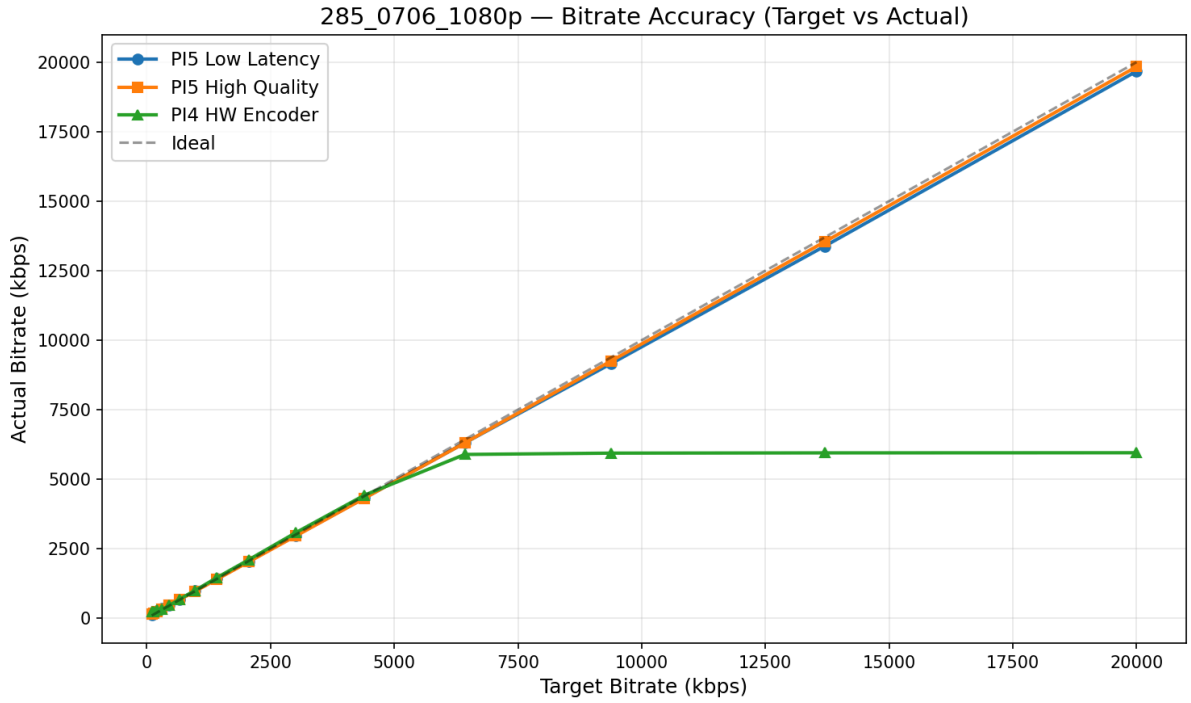
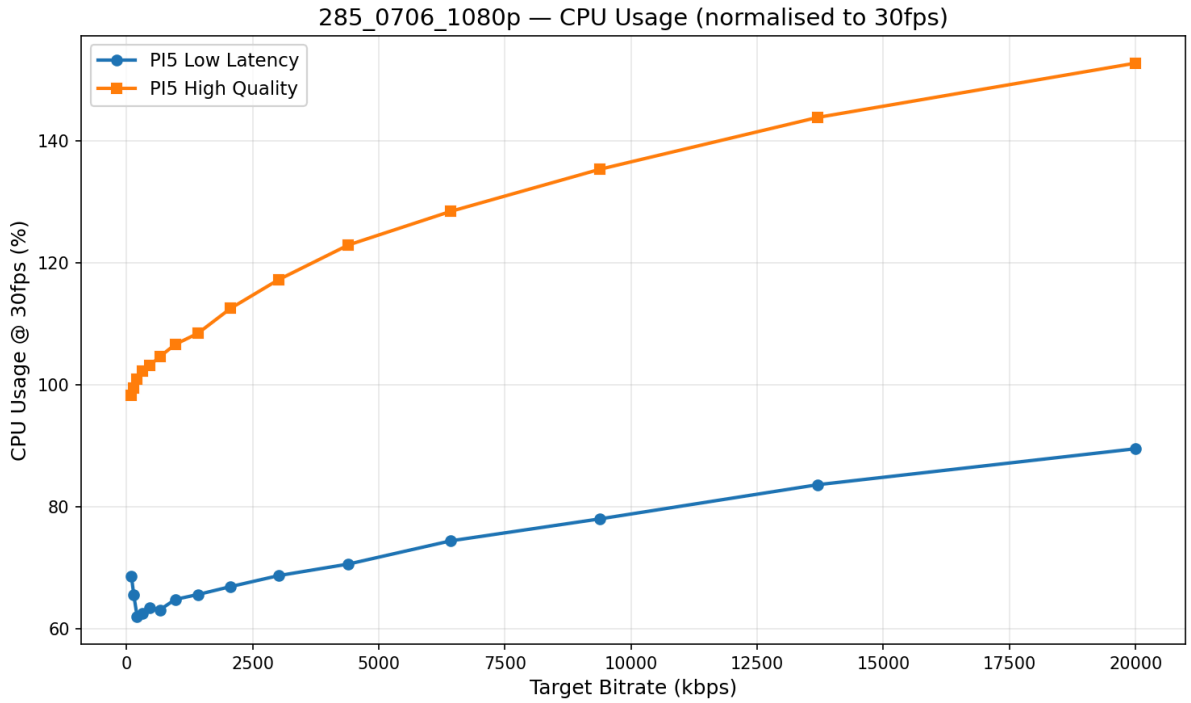


Figure 4.

CPU usage of the two Raspberry Pi 5 software encoder presets, normalised as the percentage of a single core required to sustain 30fps encoding. Low-latency mode stays below 90%, while high-quality mode exceeds 100%, requiring more than one core for real-time operation. CPU usage is not shown for Raspberry Pi 4's hardware encoder, as encoding is offloaded to dedicated hardware.



Per-frame quality

The following graphs show how PSNR and SSIM vary across all 560 frames for each encoder at three representative bitrates. This reveals temporal quality variations that the video-average metrics in the tables above do not capture.

Figure 5.
Per-frame PSNR at ~500kb/s. Raspberry Pi 4's hardware encoder shows large frame-to-frame PSNR swings and several deep dips, while Raspberry Pi 5's high-quality mode maintains the most consistent quality.

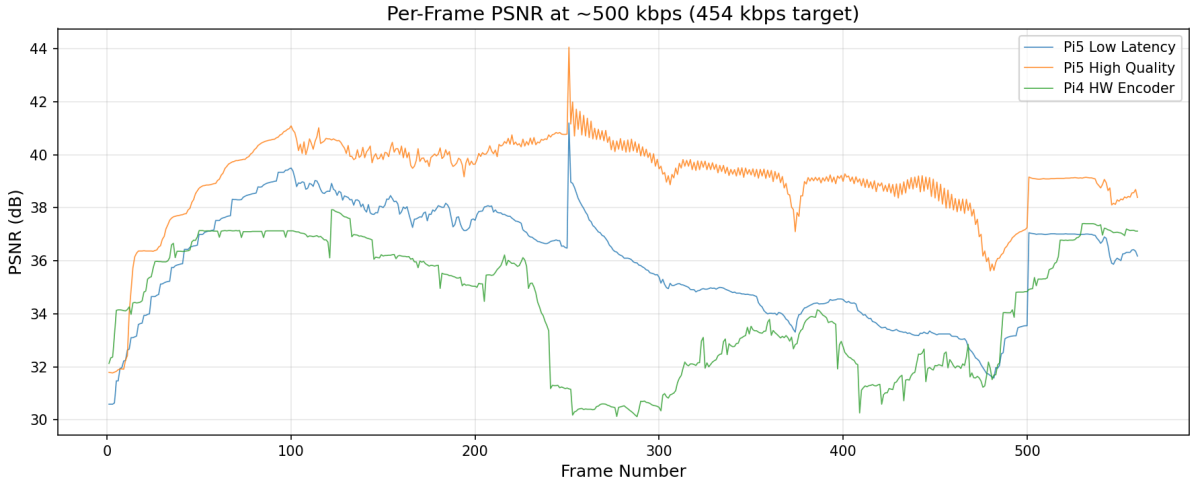


Figure 6.
Per-frame SSIM at ~500kb/s. Raspberry Pi 5's high-quality mode maintains SSIM above 0.93 throughout, while Raspberry Pi 4's hardware encoder drops below 0.85 on several frames.

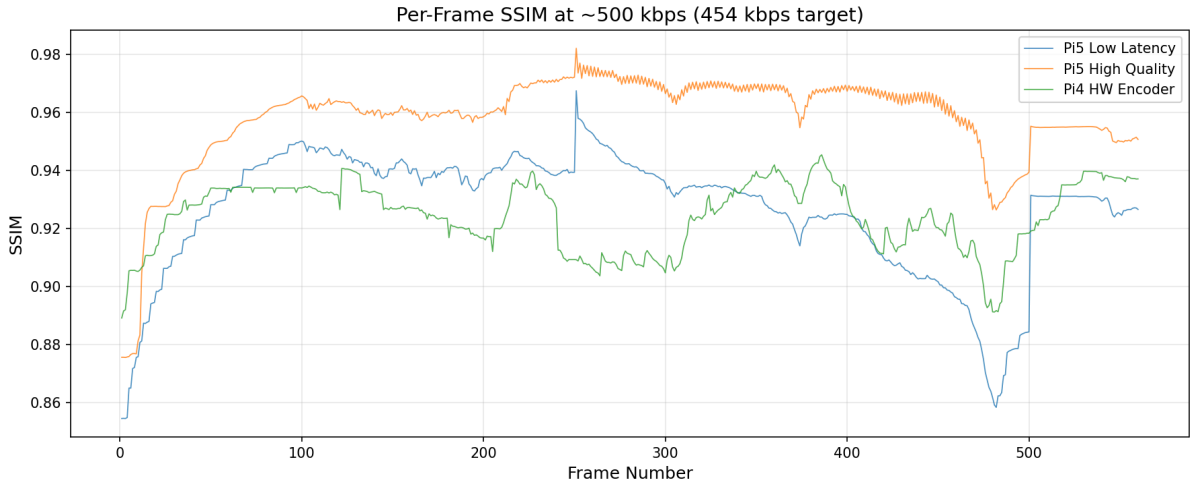


Figure 7.

Per-frame PSNR at ~1Mb/s. All three encoders are more closely matched at this bitrate, though Raspberry Pi 5's high-quality mode still leads consistently.

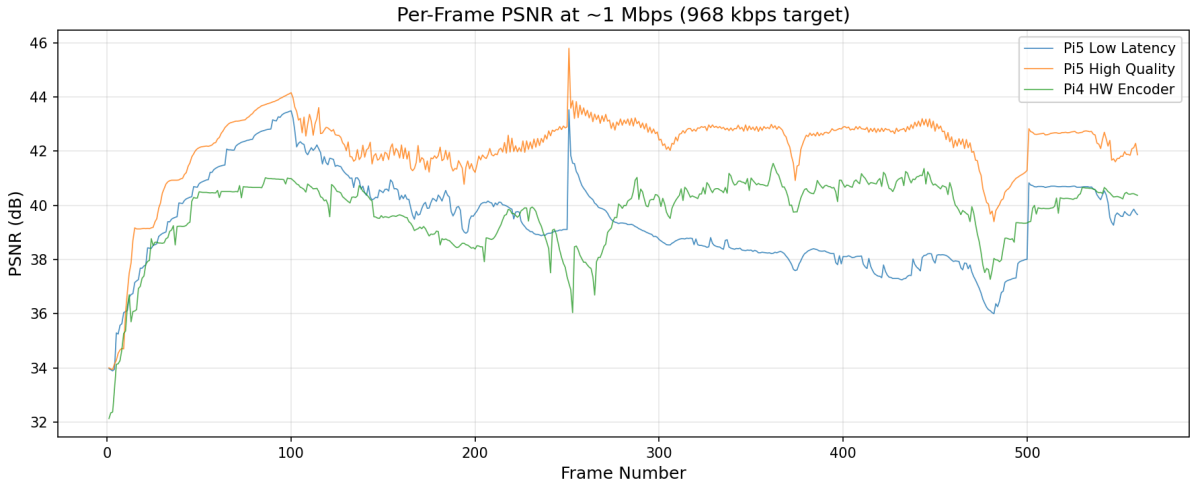


Figure 8.

Per-frame SSIM at ~1Mb/s. The three encoders converge, with Raspberry Pi 4's hardware encoder and Raspberry Pi 5's low-latency mode trading positions across frames, while Raspberry Pi 5 high-quality mode remains on top.

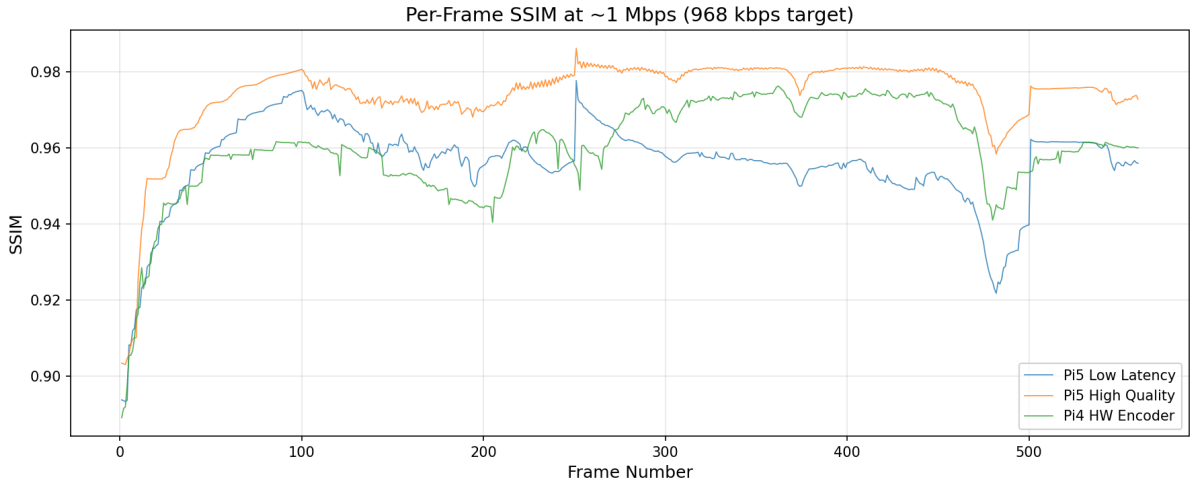


Figure 9.

Per-frame PSNR at ~10Mb/s. All three encoders achieve high PSNR with minimal variation. Note that Raspberry Pi 4's hardware encoder is actually limited to ~6Mb/s at this point.

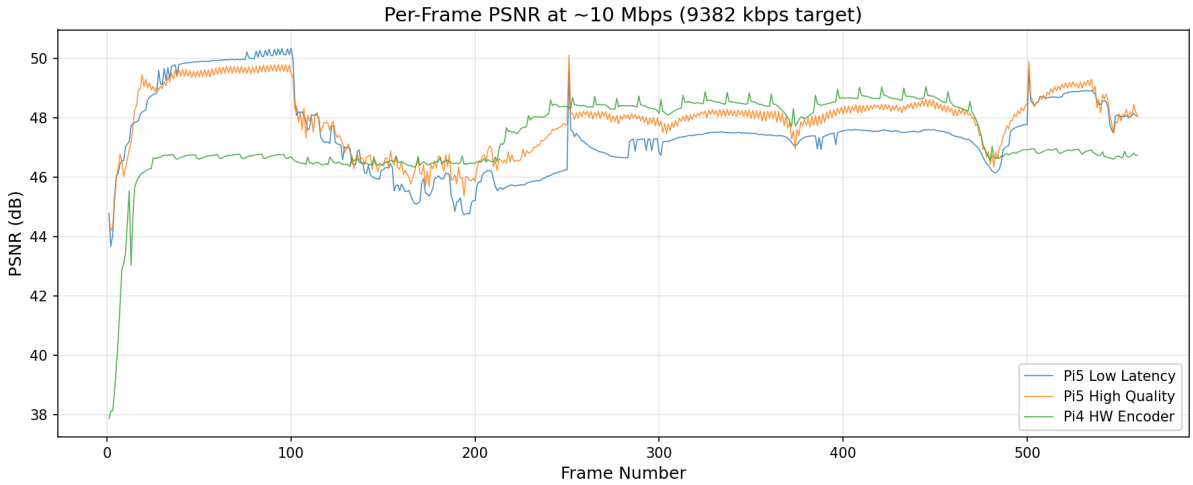
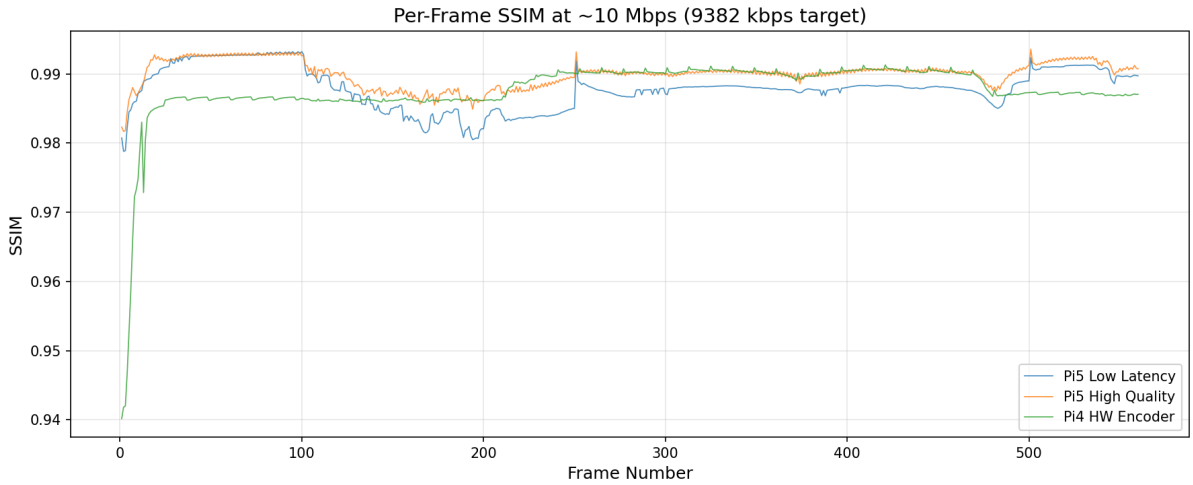


Figure 10.

Per-frame SSIM at ~10Mb/s. Quality is uniformly high across all frames for all three encoders, with SSIM above 0.98 throughout.



Conclusions

As always with encoding, there is a trade-off between encode latency, bitrate, and CPU load. The tests outlined here give some indication of how these trade-offs can be tuned to suit the end user's requirements.

It is clear that the software encoder does a very good job of maintaining the required bitrate. In all cases where a bitrate was specified, the files were all very close in size, indicating that the bitrate overrode any conflicting requirements to ensure a very constant result. In the default case where no bitrate was specified, the low latency options required more storage in order to compensate for less CPU availability.

We have shown that the encoding process can be limited to a single core and still produce good-quality results and file sizes, even at 1080p30; however, allowing the system to use more CPU results in higher quality and lower bitrates. It has also been shown that specifying lower bitrates leads to a loss of quality, although the end results are still very good — even at bitrates as low as 1Mb/s at 1080p30.

One noteworthy statistic is the very high compression ratio of these algorithms. When comparing the uncompressed file to any of these results, we get a compression factor of well over 100 times, with little or no visible degradations in quality. Even the low-quality results are still usable.

Hardware versus software encoders

Prior to Raspberry Pi 5, all Raspberry Pi models included a hardware H.264 encoder in their VideoCore GPU. Raspberry Pi 5 does not include a hardware encoder, relying instead on software encoding. The rate-distortion study above provides a quantitative comparison between Raspberry Pi 5's software encoder and Raspberry Pi 4's hardware encoder.

The key observations from Figure 1 through Figure 10 are summarised below:

CPU usage

A key takeaway from the results is how little CPU the software encoder requires. The low latency preset uses just 60–90% of a single core to encode 1080p30 in real time, leaving three cores entirely free for the application, camera pipeline, or other processing. Even the high-quality preset, which delivers substantially better compression, requires only 100–150% of a single core, which is well within Raspberry Pi 5's 4-core budget.

Extrapolating to 1080p60, the high-quality preset would require approximately 200–300% of a single core, which is still comfortably within the four available cores — something Raspberry Pi 4's hardware encoder is unable to achieve. While Raspberry Pi 4's hardware encoder uses negligible CPU, the CPU footprint of Raspberry Pi 5's software encoder is modest enough that the loss of dedicated hardware is not a practical limitation.

Rate-distortion quality

Raspberry Pi 5 high-quality mode consistently delivers the best PSNR and SSIM across the full bitrate range. At 1Mb/s, it achieves 41.81dB PSNR and an SSIM value of 0.974, compared to 39.08dB and 0.956 for low-latency mode, or 39.49dB and 0.960 for Raspberry Pi 4's hardware encoder. The SSIM metric shows an even clearer separation between the high-quality software encoder and the other two encoders at low-to-mid bitrates.

Bitrate accuracy

Both software encoder presets on Raspberry Pi 5 track the target bitrate closely (within 2–3% across the range). Raspberry Pi 4's hardware encoder is accurate in the mid-range (500–5000kb/s), but fails at both extremes due to floor and ceiling limitations.

Raspberry Pi 4's bitrate floor and ceiling

Raspberry Pi 4's hardware encoder exhibits a hard bitrate ceiling of approximately 6Mb/s for this clip. Above this point, requesting higher target bitrates produces no increase in actual bitrate or quality: PSNR plateaus at 47.03dB and SSIM at 0.988. At the low end, the encoder cannot produce bitrates below 240kb/s, overshooting the 100kb/s target by 2.4 times. The software encoder has no such limitations.

Raspberry Pi 5 low-latency mode versus Raspberry Pi 4 hardware encoder

Raspberry Pi 5's low latency preset and Raspberry Pi 4's hardware encoder achieve similar quality at matched bitrates. At lower bitrates (below 700kb/s), Raspberry Pi 5 low-latency mode has the edge, while above 1Mb/s, Raspberry Pi 4's hardware encoder pulls ahead by up to 0.85dB. This is because the hardware encoder retains coding tools such as CABAC that the ultrafast/zerolatency preset disables. However, Raspberry Pi 5 low-latency mode offers much better bitrate control and is not subject to the hardware bitrate ceiling.

Temporal quality consistency

The per-frame analysis (Figure 5 through Figure 10) reveals that Raspberry Pi 4's hardware encoder exhibits significantly larger frame-to-frame quality swings than Raspberry Pi 5's software encoder, particularly at lower bitrates. At ~500kb/s, the hardware encoder's PSNR drops by over 5dB on certain frames, while Raspberry Pi 5 high-quality mode maintains much more consistent quality throughout. This temporal instability can manifest as visible flickering during playback. At higher bitrates (~10Mb/s), all three encoders converge and produce stable, consistent quality across all frames.

In summary, Raspberry Pi 5's software encoder is a capable replacement for the hardware encoder found on previous models. It delivers superior quality, better bitrate accuracy, and more consistent frame-to-frame quality, while consuming modest CPU resources — as little as 63% of a single core in low-latency mode, or around 100% of one core for the high-quality preset. Raspberry Pi 4's hardware encoder remains competitive in the mid-bitrate range, but is fundamentally limited by its bitrate floor and ceiling.

Appendix

Single frames extracted from encoded video

The following images show frame 300 from the test clip encoded at three representative bitrates (~500kb/s, ~1Mb/s, and ~10Mb/s) for each of the three encoder configurations. The PSNR and SSIM values given are for this individual frame, not the video average. As the bitrate increases, more details are preserved, and compression artefacts (blocking, blurring) are reduced.

~500kb/s

Figure 11.
Raspberry Pi 5 low-latency mode (PSNR: 35.38dB, SSIM: 0.935). Visible blocking and loss of detail in complex regions due to aggressive quantisation with minimal coding tools.

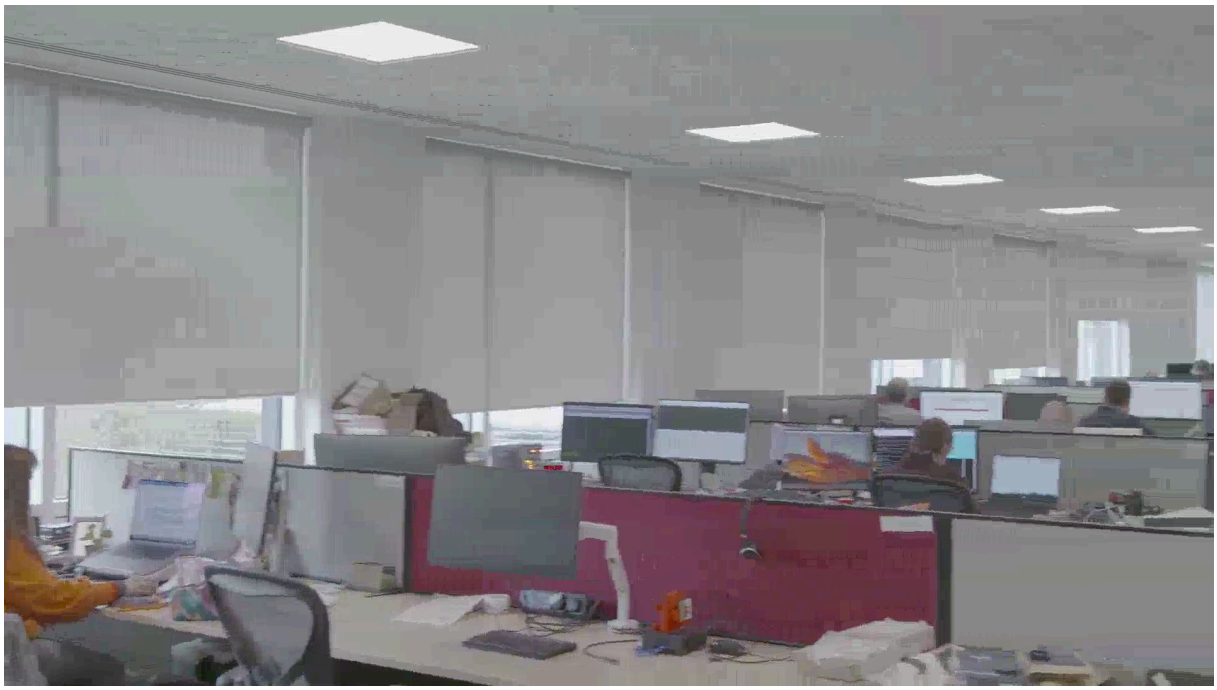


Figure 12.
Raspberry Pi 5 high-quality mode (PSNR: 39.47dB, SSIM: 0.966). Noticeably better structure retention than low-latency mode at the same bitrate, thanks to CABAC and B-frames.

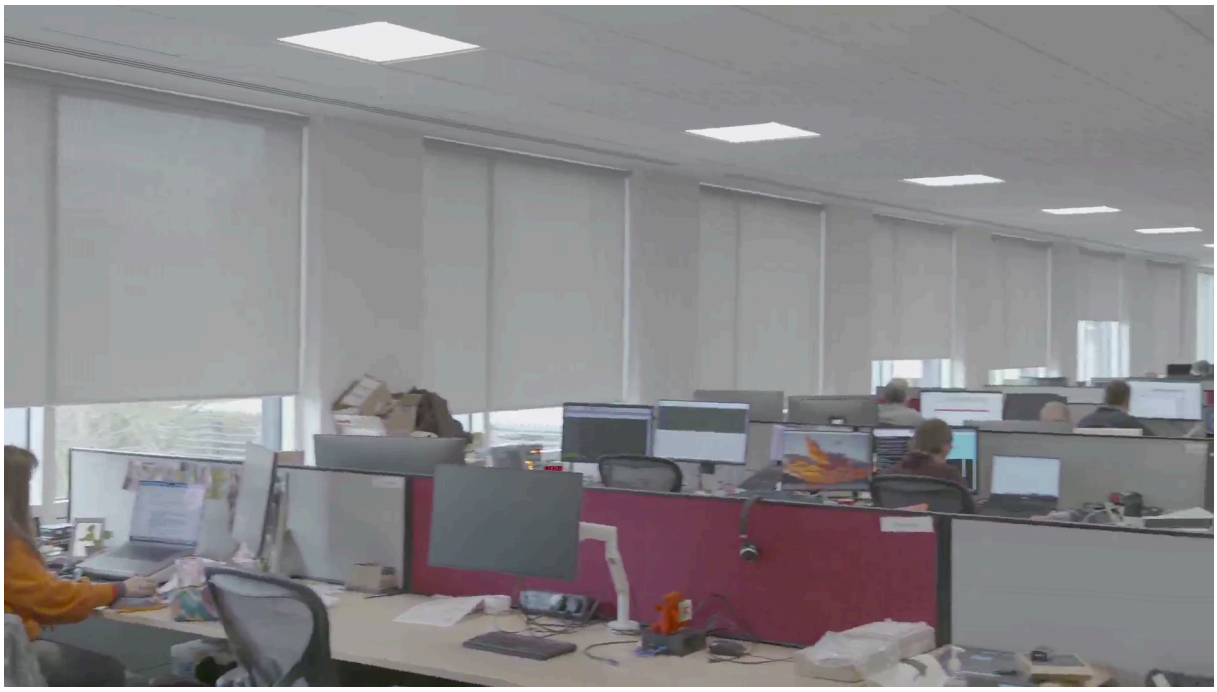
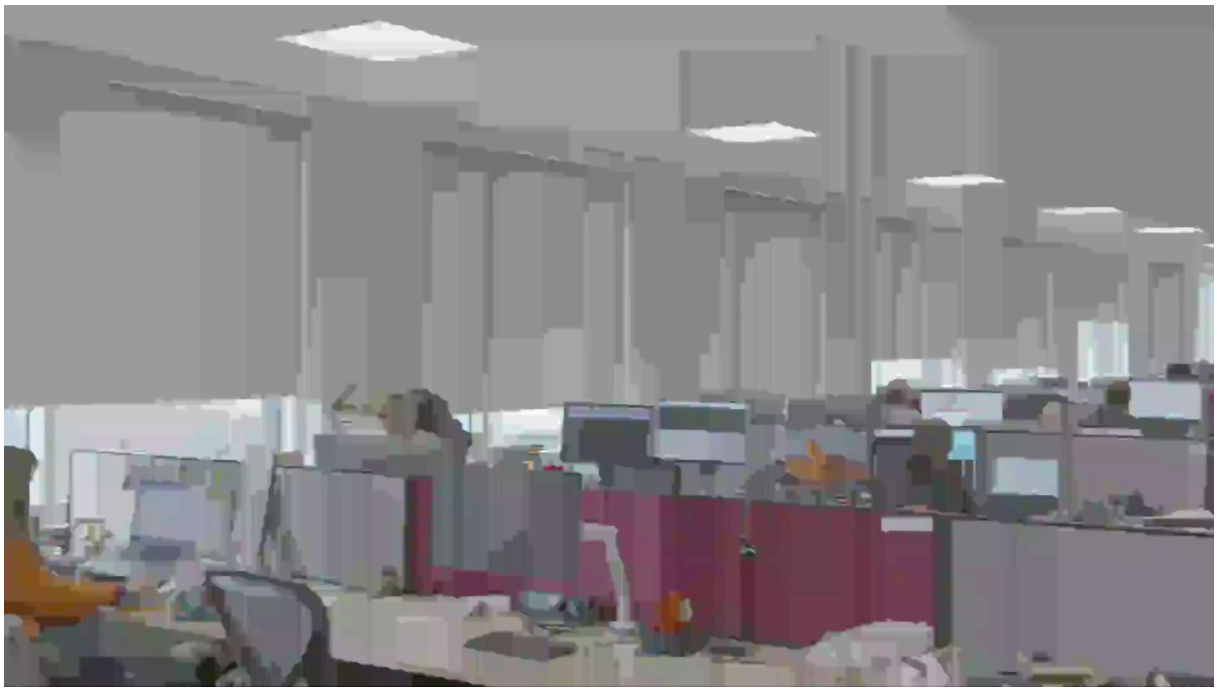


Figure 13.
Raspberry Pi 4 hardware encoder (PSNR: 30.47dB, SSIM: 0.905). Heavy blocking artefacts, particularly in the upper half of the frame. The lowest quality of the three at this bitrate.



~1Mb/s

Figure 14.
Raspberry Pi 5 low-latency mode (PSNR: 38.74dB, SSIM: 0.958). Reasonable quality, with some visible blocking in detailed regions.

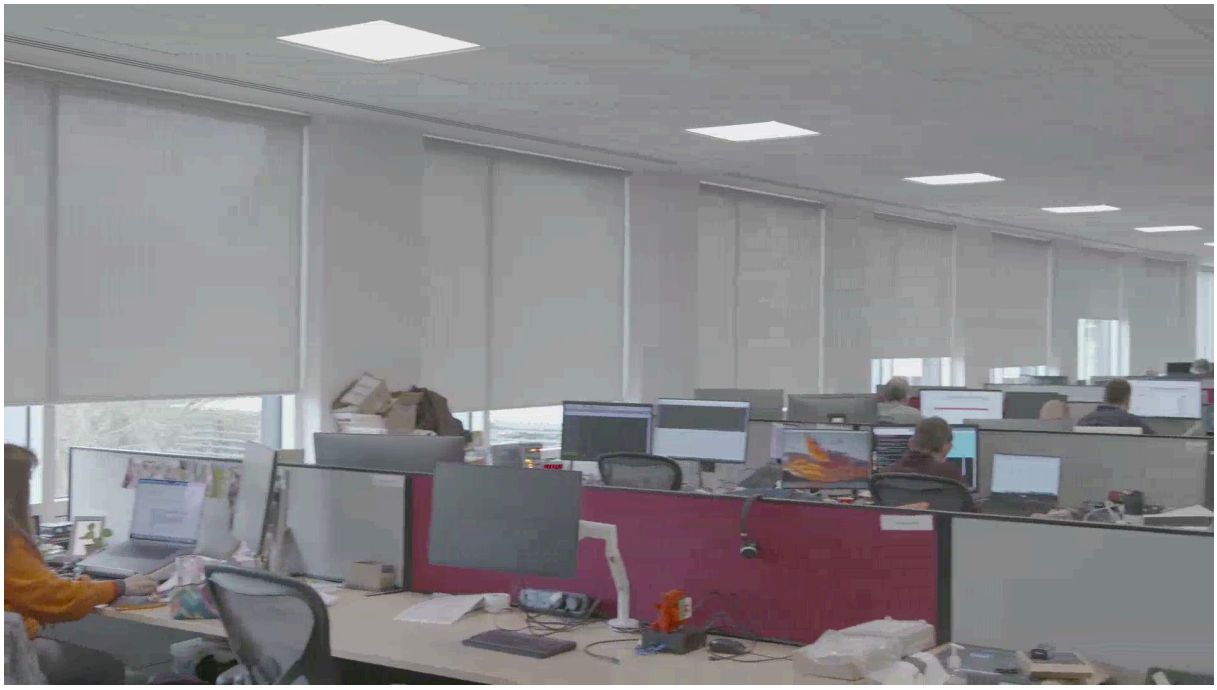


Figure 15.
Raspberry Pi 5 high-quality mode (PSNR: 42.41dB, SSIM: 0.979). Cleaner edges and fewer artefacts than low-latency mode.

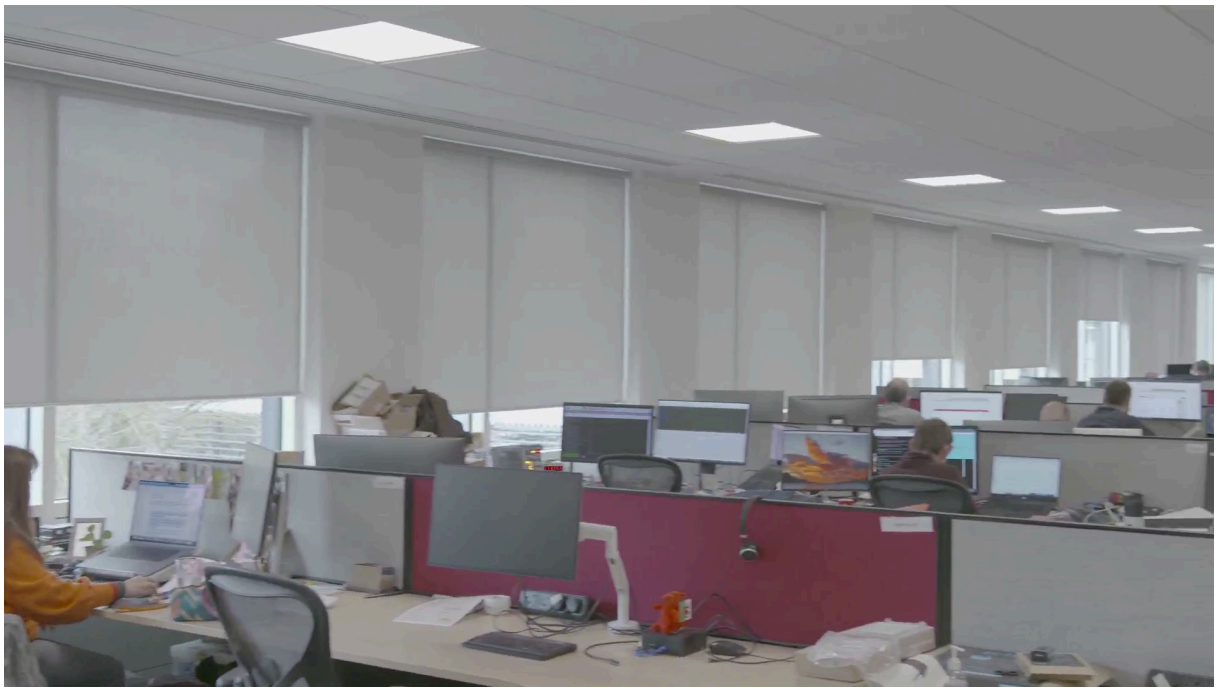
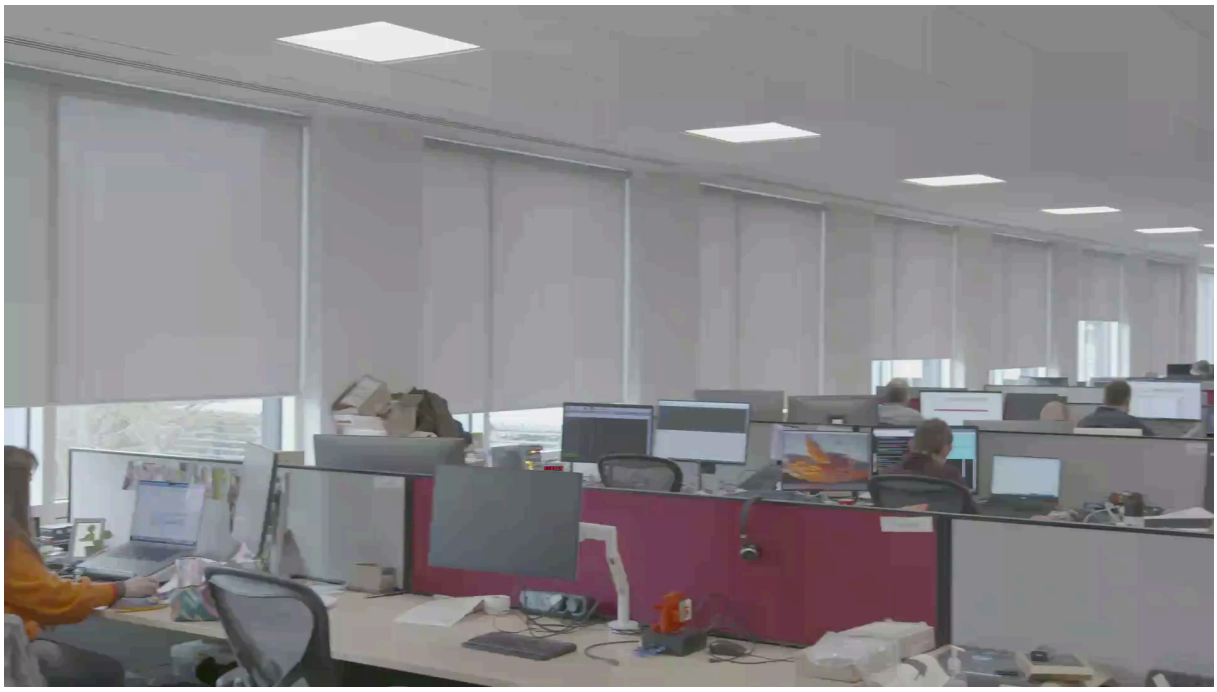


Figure 16.
Raspberry Pi 4 hardware encoder (PSNR: 40.29dB, SSIM: 0.970). Good quality at this mid-range bitrate, between the two Raspberry Pi 5 presets.



~10Mb/s

Figure 17.
Raspberry Pi 5 low-latency mode (PSNR: 46.84dB, SSIM: 0.987). High quality with minimal visible artefacts.

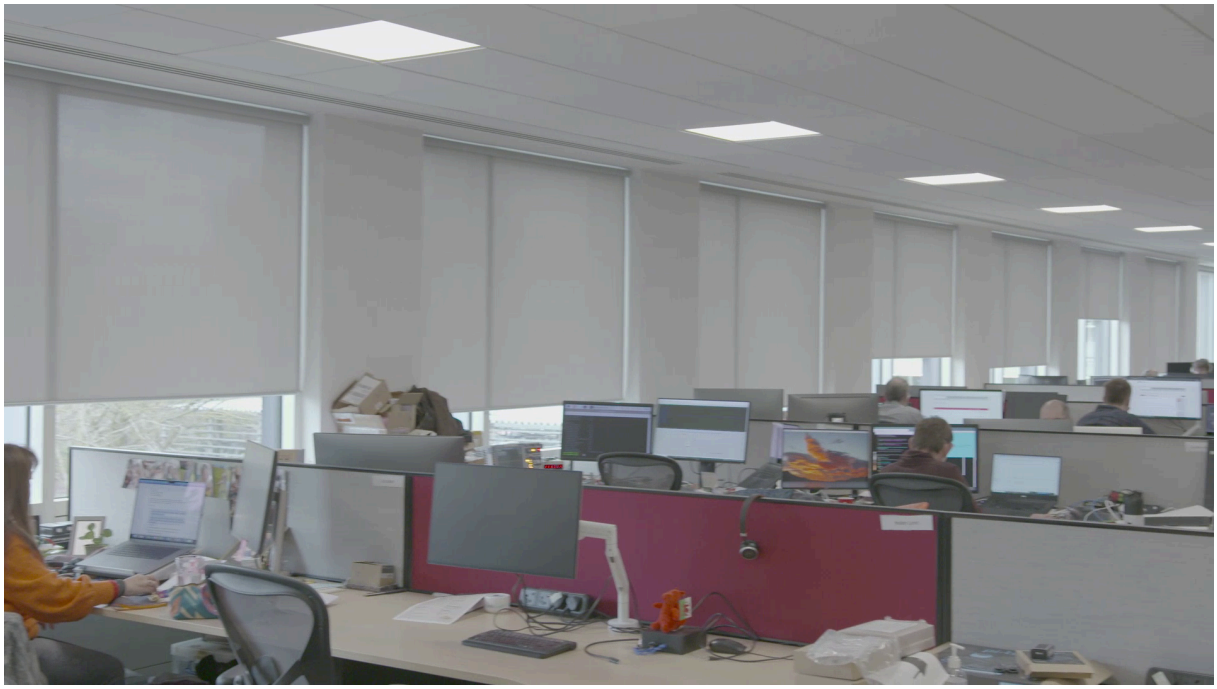


Figure 18.
Raspberry Pi 5 high-quality mode (PSNR: 47.72dB, SSIM: 0.990). Excellent quality, with near-transparent compression.

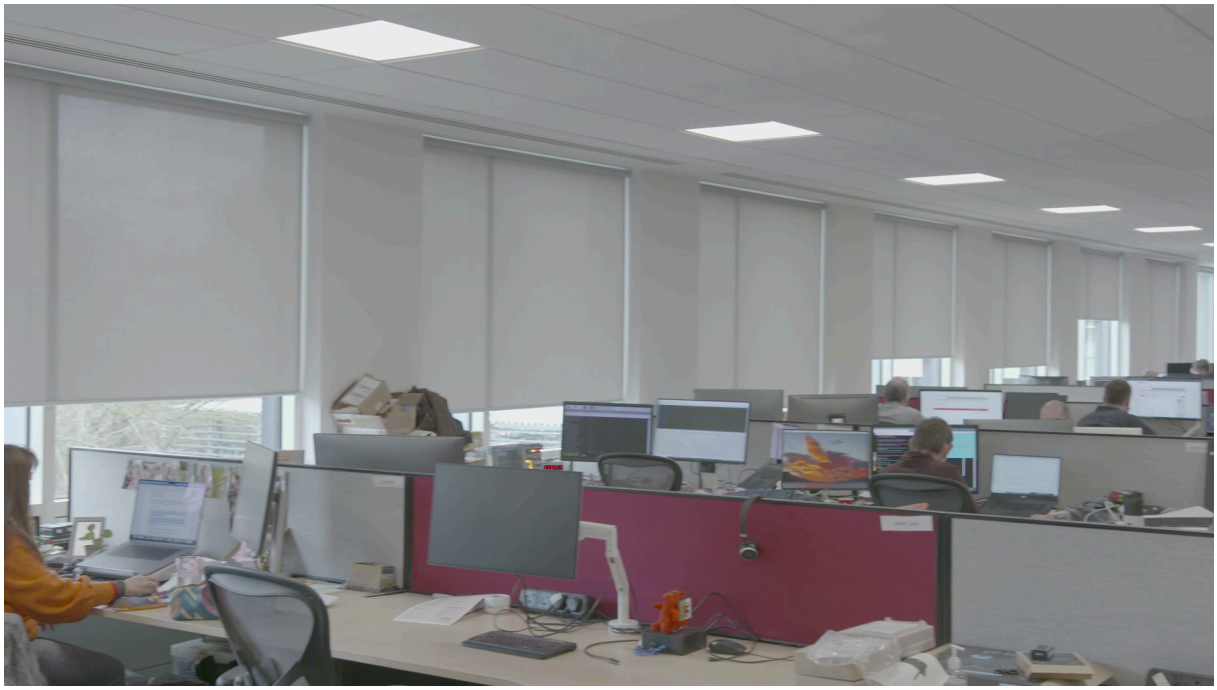
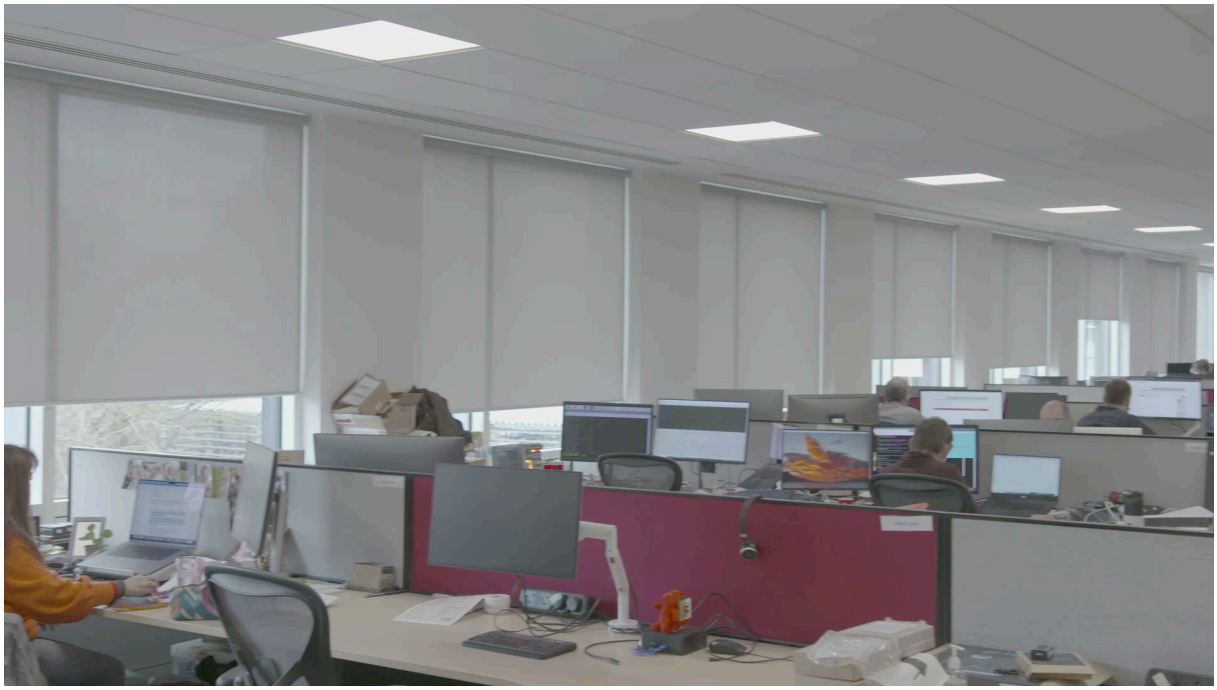


Figure 19.
Raspberry Pi 4 hardware encoder (PSNR: 48.15dB, SSIM: 0.990). Despite requesting ~10Mb/s, the hardware encoder saturates at ~6Mb/s. Quality is nevertheless excellent on this frame.



Contact us for more information

Please contact applications@raspberrypi.com if you have any queries about this whitepaper.

Web: www.raspberrypi.com

Raspberry Pi

Raspberry Pi Ltd