What Images Remember: Exact Colour Differences and Reproducible Floating Point in a Context-Mixing Lossless Image Codec

Saksham

The Truepix project

November 2026

Abstract. A general-purpose compressor, paq8px, is 33.6% smaller than JPEG XL on the Kodak benchmark, far beyond what its modelling advantage achieves elsewhere. We show why. About nine tenths of the gap between paq8px and our predictive codec lies in colour rather than brightness, and the cause is historical: the Kodak images are Photo CD scans whose RGB values are a stretched integer lattice of PhotoYCC samples. A fixed-point inverse places 87.3% of pixels on a lattice covering 12.8% of colours, with gains matching Kodak's published constants. Exploiting it cuts our Kodak total by 31.2%, to 5.723 bits per pixel. Because that gain says nothing about other photographs, we then generalise: hashed contexts over exact colour differences learn such structure with no knowledge of its origin. The resulting codec, Truepix, combines online least squares, parametric models and logistic mixing. On 24 unseen DIV2K images, JPEG XL at effort 10 is 3.62% larger in total, and Truepix is smaller on all 24 (Wilcoxon $p = 1.2 \times 10^{-7}$; geometric-mean lead 3.66%, 95% CI [2.91, 4.44]). Two earlier hold-out sets gave leads of 4.28% and 4.29%. Truepix uses floating point yet is reproducible: seven builds across compilers, 32- and 64-bit x86 and WebAssembly wrote byte-identical files, and all 245 cross-build decodes were exact. A later byte-identical rewrite cut decode time by 32%, but Truepix remains far slower than JPEG XL on pixels, and paq8px still compresses more. A separate JPEG path rebuilds JPEG files exactly and saved 21.84% against 16.40% for JPEG XL's recompression.

1 Introduction

Lossless image compression has two very different leaders. Standardised codecs such as JPEG XL [1] decode in milliseconds. Context-mixing compressors such as paq8px [2, 4] are smaller and take minutes. We set out to build a codec between them, and hit an anomaly at once. On the 24-image Kodak suite [5], our codec trailed paq8px by 47.66%, yet it trailed by only about 8% on other images. An unexplained gap that large suggests that something is hidden in the data rather than missing from the model.

The explanation turned out to be useful in three ways. It exposes a colour lattice in the most widely used image compression benchmark. It leads to a simple general mechanism that needs no knowledge of how a lattice arose. And it forced a discipline we recommend: once a set has been studied it is spent, so each claim below is tied to a set that was unseen when the codec was frozen.

Contributions.

- A diagnosis of the paq8px gap on Kodak, tracing it to the Photo CD PhotoYCC lattice, verified against Kodak's published conversion (§3).
- Exact-difference contexts, which learn colour lattices online without any parametric fit (§4.3).
- A floating-point context-mixing codec with reproducible output across builds, and the rules that make it so (§5).
- Three hold-out evaluations, including the negative results (§7).

2 Related work

Predictive coding runs from JPEG-LS [6] and CALIC [7] to the far stronger adaptive schemes built on local least squares [8], such as TMW [9], MRP [10] and Vanilc [11]. FLIF [12] and JPEG XL [1] offer practical progressive and fast coding. PAQ [2] introduced logistic mixing of many context models, and paq8px [4] carries specialised image models. Learned codecs such as L3C [13] replace hand-built models with neural networks.

Recovering a processing history from pixels is well established. Fan and de Queiroz detect JPEG history and estimate quantisers from bitmaps [14], Neelamani et al. estimate the colour transform, subsampling and quantisers of decoded colour images and show that this aids recompression [15], and Lewis and Kuhn study exact JPEG recompression [16]. The JPEG side of our lattice work is therefore not new. We found no report of the Photo CD lattice in the Kodak suite, though our search was limited and we do not claim priority. Reproducible floating point is a known hazard [17, 18].

3 Why paq8px wins on Kodak

3.1 Where the bits go

We first split the gap. Each codec compressed the green plane alone, and the remainder of its whole-image size was attributed to colour. Figure 1 shows the result for four images: green accounts for only about a tenth of the gap. At that stage Truepix spent about 2.2 and 2.1 bits per pixel on red and blue of `kodim01` once green was known, while paq8px spent about 1.5 on both together.

3.2 A lattice in the colour differences

The colour differences $(R - G, B - G)$ are strikingly sparse. Image `kodim01` uses only 1,453 distinct pairs

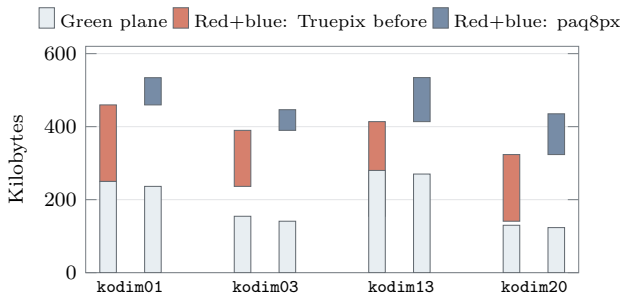


Figure 1: Size split for the diagnosis, using the codec as it stood before this work. In each pair the left bar is Truepix and the right bar is paq8px. Green is nearly level; colour is not.

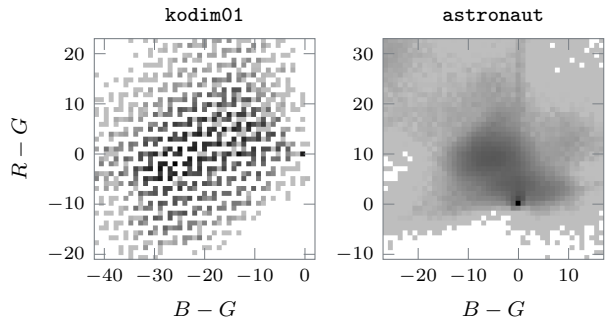


Figure 2: Occupancy of colour-difference pairs in a window around each image's median, with darker meaning more pixels. Kodak falls on a lattice; the reference image shows no such grid in this window.

in 393,216 pixels, while the 262,144-pixel reference photograph `astronaut` uses 11,756. Plotting occupancy reveals why (Figure 2): Kodak's pairs sit on a skewed grid.

A two-dimensional Fourier transform of this occupancy gives the lattice with no prior knowledge. Over four images its strongest frequencies were $(0.1348, -0.3994)$ and $(-0.3848, 0.0625)$ cycles per level. Inverting them gives the basis vectors in Table 1, which match the steps implied by Kodak's published PhotoYCC decoding [19]:

$$\begin{aligned} L &= 1.3584\,Y, \\ C_1 &= 2.2179\,(c_1 - 156), \quad C_2 = 1.8215\,(c_2 - 137), \\ R &= L + C_2, \quad B = L + C_1, \\ G &= L - 0.194\,C_1 - 0.509\,C_2. \end{aligned} \tag{1}$$

One step of c_2 moves $R - G$ by $1.8215 \times 1.509 = 2.749$, the ≈3-level jump visible in raw pixels. The same document explains the image size, since Photo CD Base images are 512×768, with chroma subsampled by two in each direction.

3.3 An exact fixed-point form

Rounding decides whether a lattice can be used losslessly, so we fitted an integer form. Starting from gains rounded from Eq. 1, a coordinate search over all Kodak colours

Table 1: The recovered lattice against Kodak's constants, as steps in $(R - G,\ B - G)$. Blind values come only from pixels; published values from Eq. 1; fitted values from Eq. 2.

Step	Blind	Published	Fitted/1024
c_1+1	(0.430, 2.648)	(0.430, 2.648)	(0.431, 2.648)
c_2+1	(2.749, 0.928)	(2.749, 0.927)	(2.748, 0.927)

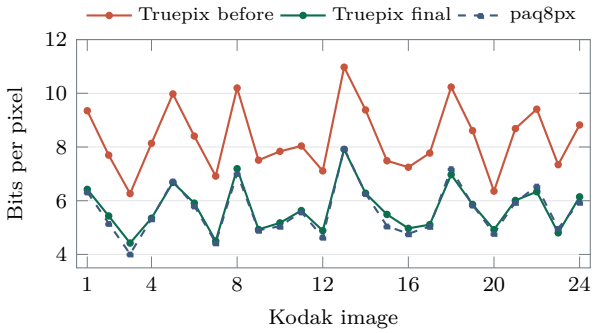


Figure 3: Kodak per image. This set was used to design mode 2, so these numbers are not a fair score.

gave

$$\begin{aligned} R &= \lfloor (1391\,Y + 1865\,c_2 - 255504)/1024 \rfloor, \\ G &= \lfloor (1391\,Y - 441\,c_1 - 949\,c_2 + 199153)/1024 \rfloor, \\ B &= \lfloor (1391\,Y + 2271\,c_1 - 354277)/1024 \rfloor, \end{aligned} \tag{2}$$

clamped to $[0, 255]$. The search moved only the offsets: nearby gains were no better. This form reaches 2,142,823 colours, or 12.8% of the cube, yet covers 87.3% of Kodak's pixels on average (84.5–93.9% by image). The first attempts missed badly, because out-of-range colours were discarded instead of clamped. Most remaining misses are one step away. We do not know their cause, and we code them as corrections.

Mode 2 of Truepix codes (Y, c_1, c_2) directly, then the corrections. All 24 Kodak images select it. Figure 3 shows the effect: the total falls from 8.324 to 5.723 bits per pixel, and paq8px's lead (5.637) falls from 47.66% to 1.52%, with Truepix smaller on 5 of 24 images.

A warning about the benchmark. Kodak is used everywhere, but its colour is unusually compressible for a reason unrelated to modelling natural images. A codec that happens to learn exact colour relationships gains far more there than it does on other photographs, so rankings on Kodak alone can mislead.

4 The Truepix codec

Figure 4 shows the pipeline. Samples are coded in raster order, plane by plane within each pixel, with G first. Encoder and decoder run the same model, so the two take similar time.

4.1 Prediction

Four recursive least-squares predictors run per plane. Each keeps an exponentially weighted covariance and solves by Cholesky factorisation every few samples:

$$\mathbf{R}_t = \lambda \mathbf{R}_{t-1} + (1-\lambda)\mathbf{x}_t\mathbf{x}_t^{\top}, \quad \mathbf{w}_t = (\mathbf{R}_t + \nu\mathbf{I})^{-1}\mathbf{r}_t. \tag{3}$$

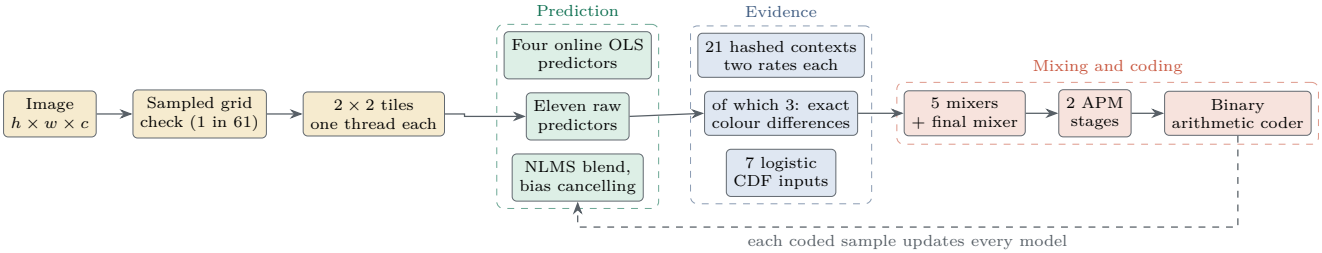


Figure 4: Truepix. Each tile is coded independently and each sample is broken into binary decisions, as in Figure 5.

Table 2: Least-squares predictors. "Own" counts neighbours in the same plane, and "cross-plane" counts neighbours per earlier plane, before removing any that are not yet decoded.

Predictor	Own	Cross-plane	λ	Solve every
Standard	18	7	0.9985	16
Medium	10	4	0.995	8
Fast	5	2	0.97	4
Wide	30	7	0.9995	32

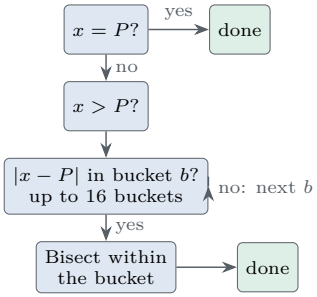


Figure 5: Decisions for one sample. Range limits skip decisions that are already certain.

Table 2 gives their settings. Regressors include causal neighbours in earlier planes, including the current pixel. Later planes keep a separate solver for each 2×2 position, because subsampled chroma behaves differently by phase.

Three of these join simple predictors (median edge, neighbours, gradient and inter-plane forms) and a combination weighted by each predictor's recent local error, giving eleven raw predictions. A normalised LMS blend, chosen by local activity, merges them into $\hat{x}$, followed by context-based bias cancellation.

4.2 Binary decomposition

The rounded prediction P splits each sample into a short series of decisions (Figure 5). Keeping P as the origin concentrates statistics, and each decision can be priced directly by a parametric distribution as well as learned by counters.

4.3 Exact-difference contexts

A decision's meaning depends on P, so a context that omits P cannot learn that specific sample values are impossible. The fix is to include the difference between P and the already coded sample G, exactly and unquantised. Writing $d = P - G$, a counter at residual node n then refers to one definite value of $x - G$, and can learn which differences occur. For plane index i, with k the known difference $R - G$ when coding B:

$$\begin{aligned} c_1 &= h(i, d, k), \\ c_2 &= h(i, d, k,\ W - W_G), \\ c_3 &= h(i, d, k,\ W - W_G,\ N - N_G), \end{aligned} \tag{4}$$

where W, N are this plane's neighbours and W_G, N_G their green samples. No lattice is detected or fitted. On `kodim01` coded without mode 2, this and the smaller changes of Table 6 together took blue from 2.07 to 1.14 bits per pixel and red from 2.22 to 1.89. For Kodak that is less than mode 2 achieves (Table 3), but it also works where no formula is known.

That matters because such structure is common. In 16 unlocked DIV2K images the same recovered bases recurred across unrelated photographs, for example $(4.49, -3.0)$ and $(-1.0, -6.15)$ in eight images, and $(0.45, 2.74)$ and $(4.83, -0.96)$ in three. We have not established where they come from, and they do not all match standard transforms. Fitting formulas image by image looked fragile, and learning sidesteps the question.

Table 3: Cost of `kodim01`'s planes, in bits per pixel. Mode 2 also pays for corrections, omitted here, which bring its total to 6.42.

Version	First plane	Second	Third
Before this work	5.06	2.22	2.07
Final model, plain RGB	5.03	1.89	1.14
Mode 2, on (Y, c_1, c_2)	4.56	0.57	0.53

4.4 Mixing

The 21 contexts each feed fast and slow adaptive counters, stored in tables of 2^{19} entries. With seven parametric logistic inputs and a bias, each decision has 50 inputs in the logit domain. Five mixers, selected by decision node combined with activity, gradient, position and local texture, feed a final mixer. Two adaptive probability maps refine the output, which drives a 32-bit binary arithmetic coder with 16-bit probabilities.

4.5 Grid modes

Mode 1 inverts the libjpeg YCbCr conversion, and mode 2 inverts Eq. 2. The encoder tests one pixel in 61 and skips the expensive inversion unless at least half fit. If at least 60% of all pixels fit, it encodes that mode as well and keeps the smaller result. Corrections are coded with a small adaptive model after the planes.

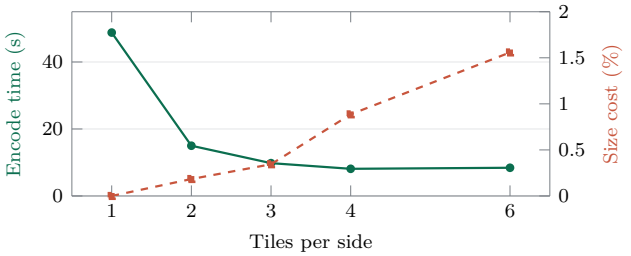


Figure 6: Tile count against mean encode time (solid) and total size (dashed) on six unlocked images.

TPX1	w	h	c	side	CRC	sizes	tile 0	⋯	tile n

Figure 7: `TPX1` layout. Each tile repeats its own size and mode, and is checked before decoding.

4.6 Tiles

Images of one megapixel or more are split into 2×2 independently coded tiles on separate threads. Tiles are the only source of parallelism, and they lose context across boundaries. Figure 6 shows the trade-off: two tiles per side cost 0.18% and run 3.3 times faster, while beyond four per side nothing more is gained on this eight-core machine.

5 Reproducible floating point

A symmetric codec fails silently if encoder and decoder disagree in one low-order bit. Fixed point is the usual remedy, but IEEE 754 requires addition, subtraction, multiplication, division and square root to be correctly rounded [18]. Reproducibility then needs only that the program's operations are the same everywhere [17]. Three things had to change:

1. **No libm.** `log` and `exp` are replaced by polynomial versions built only from the operations above.
2. **Fixed summation order.** Mixer dot products accumulate in eight lanes with a fixed final tree, so vector width cannot alter the sum, and the compiler can still vectorise it.
3. **No contraction.** `-ffp-contract=off` stops multiply-add fusion. Compile-time checks refuse `-ffast-math` and excess-precision targets. The contraction flag itself cannot be checked in the source, so it remains a build requirement.

The file (Figure 7) stores a CRC-32 of the pixels, so a disagreement is an error rather than a wrong image. Every tile header is validated against the file header before any model is allocated, which closes a memory-safety hole for untrusted files.

Evidence. We built seven ways: GCC 14 with AVX2, with FMA enabled, with baseline SSE2, and unoptimised; Clang (via Zig) for 64-bit and 32-bit x86; and WebAssembly under Node. Five test images covered plain RGB, Photo CD, JPEG history, greyscale and RGBA. Every build wrote byte-identical files, and every build decoded every other build's file exactly: all $7 \times 7 \times 5 = 245$ decodes (Figure 8). In a separate check,

	GCC AVX2	GCC FMA	GCC SSE2	GCC -O0	Clang x64	Clang x86	WebAssembly
GCC AVX2	5/5	5/5	5/5	5/5	5/5	5/5	5/5
GCC FMA	5/5	5/5	5/5	5/5	5/5	5/5	5/5
GCC SSE2	5/5	5/5	5/5	5/5	5/5	5/5	5/5
GCC -O0	5/5	5/5	5/5	5/5	5/5	5/5	5/5
Clang x64	5/5	5/5	5/5	5/5	5/5	5/5	5/5
Clang x86	5/5	5/5	5/5	5/5	5/5	5/5	5/5
WebAssembly	5/5	5/5	5/5	5/5	5/5	5/5	5/5

Figure 8: Row build decoding column build's files. Each cell counts exact decodes out of five images.

a full-size tiled image also matched across GCC, Clang, 32-bit x86 and WebAssembly. All 12 random bit flips in a coded stream were rejected, and no wrong image was returned. ARM64 compiles but has never been run, so that case remains untested.

6 Method

Data. Three kinds of set were kept apart. *Design:* 17 PNGs used to build the codec, and Kodak once it was studied. *Tuning:* crops of DIV2K validation images 0825–0840 [20]. *Hold-out:* three blocks of 24 consecutive DIV2K images, chosen by filename rule before being opened, each checksummed and scored once by a frozen codec: Set A (0801–0824), Set B (0841–0864) and Set C (0865–0888). Only Set C is scored by the final build, and it is our primary result. Set B was scored by the tiled build before the portability changes. Set A was first scored without tiles, at a lead of 4.55% over JPEG XL effort 10 and 24 of 24 wins, then rerun with tiles for Table 4.

Verification. All Truepix results were decoded and compared byte for byte, and rival results were decoded and compared where they are reported. Rivals used their strongest lossless settings: JPEG XL efforts 9 and 10, FLIF `-E100`, WebP method 6.

Statistics. Paired by image, we report total size, a percentile bootstrap (20,000 resamples of images) for the geometric-mean ratio, and an exact Wilcoxon signed-rank test.

Timing. A Ryzen 7 5800HS (8 cores, 16 threads, 15 GB) ran one job at a time, with each library's default threading. The machine was not idle throughout: an unrelated CPU job shared it for part of the work, so timings are indicative. Paq8px was too slow to run on the hold-out sets.

7 Results

7.1 Compression

Table 4 gives the main results. On Set C, Truepix is smaller than every rival on all 24 images. Against JPEG XL at effort 10 the total lead is 3.62% and the

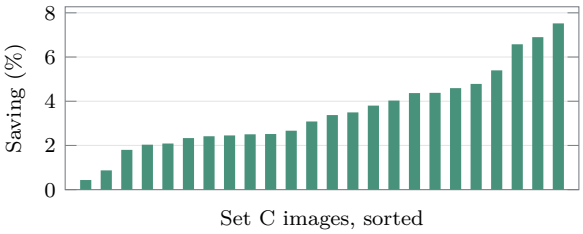


Figure 9: Truepix's size saving against JPEG XL at effort 10, per image.

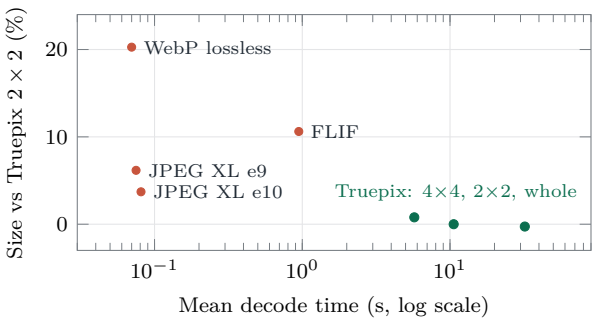


Figure 10: Size against decode time. Lower left is better. Truepix's lead is bought with roughly two orders of magnitude in decode time.

geometric-mean lead 3.66% (95% CI [2.91, 4.44]), ranging from 0.44% to 8.13% by image (Figure 9). The signed-rank test gives $p = 1.2 \times 10^{-7}$, the smallest value possible with 24 pairs. The leads on Sets A and B were 4.28% and 4.29%, so a fair summary is 3.6–4.3%. Tiling cost one win on Set A: image 0804 moved from a narrow win to a 0.16% loss.

On the design PNGs, Truepix totals 2,251,773 bytes, 9.4% below JPEG XL e10 (smaller on 16 of 17). On Kodak it totals 6,750,776 bytes. Table 5 lists Kodak rivals, but these figures show Photo CD history, not general strength, and they should not be read as a fair score.

7.2 Against paq8px

We could only compare paq8px on the 16 tuning crops, where the final Truepix is still behind: green 2,697,523 against 2,566,223 bytes (paq8px 4.9% smaller), colour 5,638,301 against 5,267,444 (6.6% smaller). Truepix is therefore not the strongest compressor here, and cmix, GraLIC, EMMA and learned codecs [13] were not tested at all.

7.3 What helped, and what did not

Table 6 records each change in turn. Gains were modest except for exact-difference contexts, and several reasonable ideas failed. Trying several spreads per sample did nothing. Pre-training was estimated by stacking one image above another: the second image saved about 0.8%, too little to justify. Learning rates were already near their best. A green gap of about 5% to paq8px remains and has not been explained.

7.4 Speed

Table 7 and Figure 10 give timing for the scored build on five Set C images. Encoding with tiles takes about 17.2 s, between JPEG XL at effort 9 (1.8 s) and effort 10 (42.0 s). Decoding is Truepix's real weakness: 10.59 s against 0.08 s, about 141 times slower, because a symmetric model does all its work again. Truepix suits archiving, not web delivery. By calculation, the tables of a tiled image take about 265 MB.

7.5 Faster decoding without changing a byte

After the hold-out tests we looked for speed that leaves every file unchanged, so that all results above still apply. Cycle counters put about half of decode time in three places: the parametric inputs, the least-squares update and the least-squares solve. The compiler had not vectorised the update, because possible pointer aliasing blocked it.

The constraint is that each value must receive the same operations in the same order. Removing the aliasing vectorises the update directly. The row form of Cholesky is an order-sensitive sum that cannot run in lanes, but the column form gives each entry the same subtractions in the same order, so it vectorises with identical bits. The seven parametric inputs and four of the five mixer squashes run in AVX2 lanes that repeat the scalar operations, with no fused multiplies. Decisions in one sample also share interval edges, so their logistic values are cached. Smaller changes replace library rounding and a linear quantiser search with exact equivalents, and prefetch the counters.

On the same five images, in alternating runs with tiles on all cores, decoding fell from 10.86 to 7.37 s and encoding from 16.65 to 12.01 s, saving 32% and 28% (Table 8 shows the steps). Compared with the earlier JPEG XL timing, decoding is now about 98 times slower instead of 141. Plain builds without AVX2 keep only the compiler-level gains, which we did not measure. Profile-guided builds and CPU-specific tuning gave no gain.

All five files were identical to the old build's, and all 20 decodes were exact. We reran the seven-build check on the new source, with Clang and the unoptimised build also taking the AVX2 path: all 35 files matched the old build byte for byte, and all 245 cross-build decodes were exact. The new source has not been run on ARM.

7.6 JPEG recompression

Pixel coding is the wrong tool for a JPEG: it must preserve every artefact, so the file grows. Truepix therefore has a separate path for sequential and progressive Huffman JPEGs. It reads the quantised DCT coefficients, codes them with their own mixing model, and rebuilds the original file byte for byte. Headers are stored, Huffman coding is repeated, and each file is rebuilt in full before it is accepted, so an unusual encoder is refused and never corrupted. This path uses a separate file format, and a stored test hash confirms that pixel files did not move.

Each block is coded as interior coefficients, then edges, then DC. Its 18 contexts draw on ideas from Lepton [3]: neighbouring blocks, coefficient counts and co-located channels. Integer prediction from block-edge continuity supplies the edge and DC estimates, because it must be reproducible.

Table 4: Hold-out results. Each set is 24 images of about 2.8 megapixels. "Bigger" compares total size with Truepix on that set, and "wins" counts images where Truepix is smaller. All Truepix files decoded exactly. Set A is shown with tiles, and Set C alone used the final build.

	Set A (0801–0824)			Set B (0841–0864)			**Set C (0865–0888), final build**		
Method	bpp	bigger	wins	bpp	bigger	wins	bpp	bigger	wins
Truepix	**7.964**	–	–	**7.697**	–	–	**8.110**	–	–
JPEG XL, effort 10	8.305	4.28%	23/24	8.027	4.29%	24/24	8.404	3.62%	24/24
JPEG XL, effort 9	8.481	6.49%	24/24	8.232	6.95%	24/24	8.596	5.99%	24/24
FLIF	8.795	10.44%	24/24	8.439	9.63%	24/24	8.863	9.28%	24/24
WebP lossless	9.348	17.37%	24/24	9.095	18.16%	24/24	9.447	16.48%	24/24
PNG as distributed	12.497	56.92%	24/24	12.547	63.00%	24/24	12.832	58.22%	24/24

Table 5: Kodak (design set, not a fair score). Truepix: 6,750,776 bytes, 5.723 bpp. paq8px: 6,649,818 bytes.

Method	Bytes	bpp	Bigger	Wins
JPEG XL e10	9,955,343	8.439	+47.5%	24/24
JPEG XL e9	10,008,326	8.484	+48.3%	24/24
FLIF	10,254,009	8.692	+51.9%	24/24
WebP lossless	11,204,040	9.498	+66.0%	24/24
JPEG 2000	11,296,508	9.576	+67.3%	24/24
JPEG XR	12,846,422	10.890	+90.3%	24/24
AVIF lossless	13,426,844	11.382	+98.9%	24/24
bzip2	13,563,303	11.498	+100.9%	24/24
7z/LZMA	14,353,724	12.168	+112.6%	24/24

Table 6: Development steps on 16 tuning crops (cumulative bytes). Rejected steps were removed before continuing.

Step	Green	Colour
Starting point	2,718,580	5,730,204
Several spreads (rejected)	2,720,159	–
Wide OLS, local weighting	2,712,692	–
Texture contexts, fifth mixer	2,700,742	–
Exact-difference contexts	–	5,645,837
Neighbour pairs (rejected)	–	5,640,271
Final build, all kept changes	2,697,523	5,638,301
paq8px, for reference	2,566,223	5,267,444

The model was tuned on crops of DIV2K 0825–0840 and a few other JPEGs. The photographs of Table 9 were spent for pixel coding, but the JPEG model had never seen them. It was frozen, then scored once. Quality 85 was the primary test: JPEG XL at effort 10 saved 16.31%, and paq8px saved more than either, at 26.41%. Truepix took 0.9 s each way per image on this machine. A seven-build check of ten files covering pixels and JPEGs matched byte for byte in all 70 cases, and all 490 cross-build decodes were exact. All files came from one encoder, so other encoders may behave differently.

8 Discussion

What is and is not shown. Our claim is narrow: among the practical codecs tested, on these DIV2K images, Truepix gave the smallest files, by about 3.6–4.3% over JPEG XL's top effort. DIV2K may not represent all photographs, and these sets come from one source. All hold-out sets are now used, though 12 of the 100 DIV2K validation images remain unseen.

Table 7: Mean seconds per image for the scored build on five Set C images (2.8 MP on average), with total size against Truepix with 2×2 tiles.

Method	Encode	Decode	Size
Truepix whole	47.8	32.08	-0.27%
Truepix 2×2 tiles	17.2	10.59	+0.00%
Truepix 4×4 tiles	8.4	5.73	+0.79%
JPEG XL e10	42.0	0.08	+3.71%
JPEG XL e9	1.8	0.08	+6.17%
FLIF	14.3	0.95	+10.62%
WebP lossless	10.0	0.07	+20.27%

Table 8: Decode time saved by the bit-exact changes, cumulative, on one thread, for a 0.48 MP crop over three alternating runs.

Change	Saved
Least squares, rounding, quantiser, regressors	16%
AVX2 lanes, batched squashes, prefetch	34%
Cached interval edges	37%

Limits of the method. The Photo CD form was fitted to Kodak itself, not to independent Photo CD scans, and the rest of its misses are unexplained. Our literature search was limited. Truepix handles only 8-bit images. Its JPEG path covers Huffman-coded files and refuses arithmetic-coded or truncated ones.

Future work. Decode speed remains the priority. Section 7.5 used up the changes that leave files alone, so further gains need a cheaper model, such as cheaper counters and fewer mixer inputs. Chroma subsampling is visible in mode 2's position-dependent costs (0.52, 0.36, 0.77 and 0.66 bits per sample in the second plane, by position), but is not yet explicitly modelled. The recurring DIV2K lattices need an explanation, and the remaining green gap to paq8px is the largest open compression question. ARM execution must also be tested.

9 Conclusion

The surprising strength of a slow compressor on a famous benchmark was not a matter of better modelling alone. It was the benchmark's history: an integer colour lattice

Table 9: JPEG size saved on 24 DIV2K photographs (0865–0888). Each JPEG was rebuilt byte for byte. Wins are against JPEG XL effort 9.

Pillow JPEG	Truepix	JPEG XL	Wins
Quality 85	21.84%	16.40%	24/24
Quality 75	22.60%	16.54%	24/24
Quality 95	22.54%	18.51%	24/24
Quality 85, progressive	17.06%	11.31%	24/24

from Photo CD, recoverable from pixels and matching constants Kodak published decades ago. Turning that observation into exact-difference contexts yields a practical gain with no history detection at all. Truepix's lead over JPEG XL held across three unseen sets, reproducibly across seven builds, at the stated cost in decode time.

Reproducibility. Results were measured as PXM, the name in the raw files. The release differs from the scored build in its name, its file marker, the speed changes of Section 7.5 and the separate JPEG path of Section 7.6. On eight test images, the renamed build's files matched byte for byte after that marker. The raw results, the script behind this paper's numbers and plots, and the measurement scripts are in `paper/`. Hard-coded development figures in Tables 6 and 3 and Figure 1 come from the session logs. Independent replication has not been attempted.

Acknowledgement. Implementation, experiments and drafting were carried out with Claude Code, an AI coding assistant.

References

[1] J. Alakuijala et al., "JPEG XL next-generation image compression architecture and coding tools," in *Proc. SPIE Applications of Digital Image Processing XLII*, 2019.

[2] M. Mahoney, "Adaptive weighing of context models for lossless data compression," Florida Tech. Tech. Rep. CS-2005-16, 2005.

[3] D. R. Horn et al., "The design, implementation, and deployment of a system to transparently compress hundreds of petabytes of image files for a file-storage service," in *Proc. USENIX NSDI*, 2017.

[4] paq8px contributors, *paq8px*, version 217 used here, `https://github.com/hxim/paq8px`.

[5] R. Franzen, "Kodak lossless true color image suite," `https://r0k.us/graphics/kodak/`.

[6] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Trans. Image Process.*, vol. 9, no. 8, 2000.

[7] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *IEEE Trans. Commun.*, vol. 45, no. 4, 1997.

[8] X. Li and M. T. Orchard, "Edge-directed prediction for lossless compression of natural images," *IEEE Trans. Image Process.*, vol. 10, no. 6, 2001.

[9] B. Meyer and P. Tischer, "TMW — a new method for lossless image compression," in *Proc. Picture Coding Symp.*, 1997.

[10] I. Matsuda, N. Ozaki, Y. Umezu, and S. Itoh, "Lossless coding using variable block-size adaptive prediction optimized for each image," in *Proc. EUSIPCO*, 2005.

[11] A. Weinlich, P. Amon, A. Hutter, and A. Kaup, "Probability distribution estimation for autoregressive pixel-predictive image coding," *IEEE Trans. Image Process.*, vol. 25, no. 3, 2016.

[12] J. Sneyers and P. Wuille, "FLIF: Free lossless image format based on MANIAC compression," in *Proc. IEEE ICIP*, 2016.

[13] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Practical full resolution learned lossless image compression," in *Proc. IEEE/CVF CVPR*, 2019.

[14] Z. Fan and R. L. de Queiroz, "Identification of bitmap compression history: JPEG detection and quantizer estimation," *IEEE Trans. Image Process.*, vol. 12, no. 2, 2003.

[15] R. Neelamani, R. de Queiroz, Z. Fan, S. Dash, and R. G. Baraniuk, "JPEG compression history estimation for color images," *IEEE Trans. Image Process.*, vol. 15, no. 6, 2006.

[16] A. B. Lewis and M. G. Kuhn, "Exact JPEG recompression," in *Proc. SPIE Visual Information Processing and Communication*, 2010.

[17] D. Monniaux, "The pitfalls of verifying floating-point computations," *ACM Trans. Program. Lang. Syst.*, vol. 30, no. 3, 2008.

[18] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019.

[19] Eastman Kodak Company, "Fully utilizing Photo CD images, article no. 4: PhotoYCC color encoding and compression schemes," Photo CD Information Bulletin PCD 045, 1994.

[20] E. Agustsson and R. Timofte, "NTIRE 2017 challenge on single image super-resolution: dataset and study," in *Proc. IEEE CVPR Workshops*, 2017.