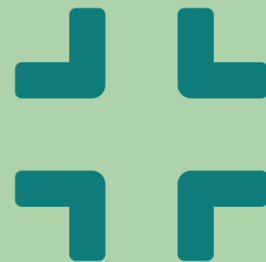


WEB PERFORMANCE

Choosing The Right Image Format



Solving the Modern Page-Load Crisis with Next-Gen Compression

Shailesh | Sr Software Engineer/ Lead Architect

Every Second Counts.

The average webpage has grown 300% in a decade — and users are paying the price.



2.3 MB

Avg. page weight
in 2024



53%

Users abandon after
3-second load



1s

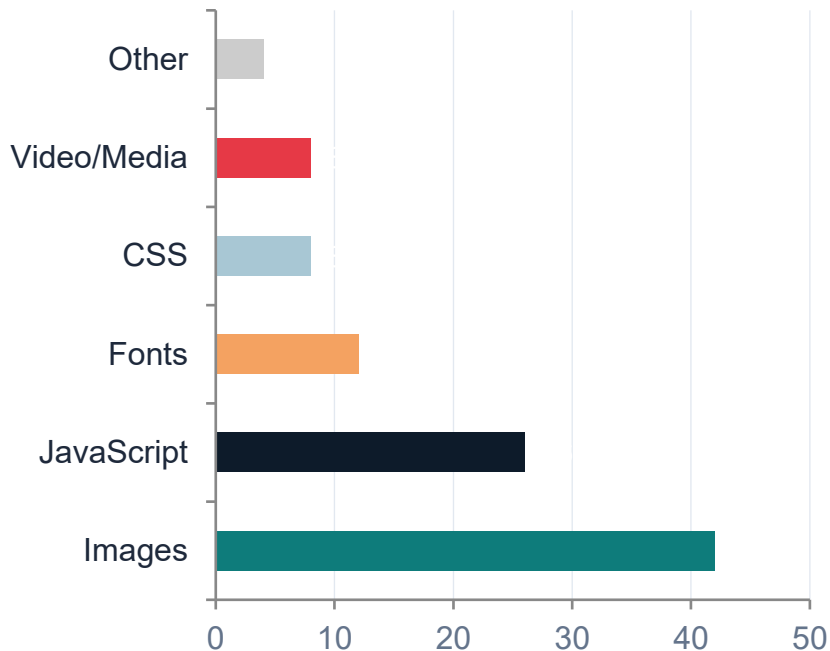
Delay = 7% drop
in conversions



\$2.6B

Annual revenue lost
to slow sites

What's Weighing Your Pages Down?



Images dominate at 42%

Most still served as PNG/JPEG instead of AVIF or WebP



JS bundles keep growing

Tree-shaking is underused; many apps ship unused code



No server-side compression

Gzip is still the default on 40%+ of servers in 2024

Next-Gen Compression: The Landscape

Three categories — each solving a different layer of the stack

Text & Data

HTTP Transfer Compression

- Brotli (BR)
- Zstandard (Zstd)
- LZ4 / Snappy

Images

Next-Gen Image Formats

- AVIF
- WebP 2
- JPEG XL

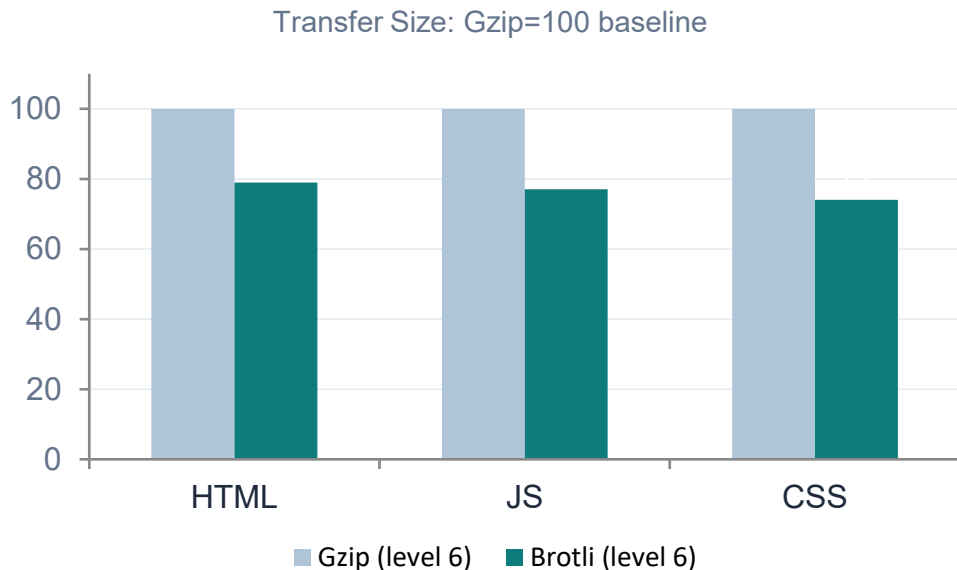
Assets

Bundle & Resource Compression

- CodSplitting
- Tree Shaking
- HTTP/3 + QUIC

Brotli: Google's Compression Gift to the Web

Replaces Gzip for HTTP text payloads — 20–26% smaller at equivalent speed



Supported by 96%+ of browsers worldwide



HTTPS-only — works on all modern CDNs



Static pre-compression for assets (.br files)



```
nginx: brotli_static on; brotli_comp_level 6;
```



```
Express: compression({ level: 6, brotli: true })
```

Finding & Replacing Repetition

LZ77 scans text for repeated byte sequences and replaces them with (distance, length) back-references.

INPUT:

abcde

abcde

fgh

abcde

fgh



OUTPUT:

abcde

<5, 5>

fgh

<5, 8>

<3, 5>



Sliding Window

Brotli looks back up to 16 MB — far larger than gzip's 32 KB window



Static Dictionary

~13,500 common web words (HTML tags, JS keywords) pre-loaded — zero bytes needed

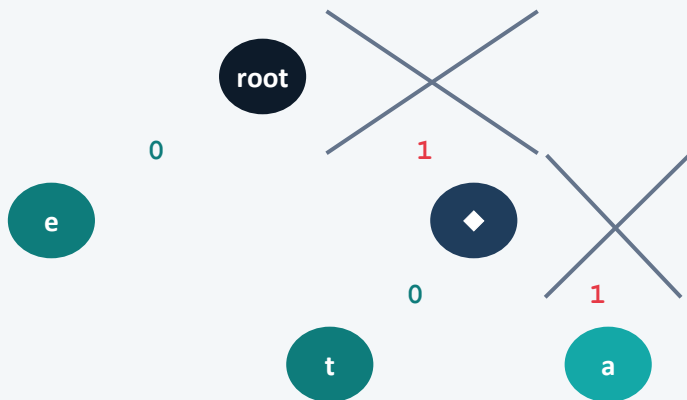


Copy Commands

A single (distance, length) token can reproduce hundreds of bytes

Huffman Coding & Context Modelling

HUFFMAN TREE



Encoding Table

"e"	→	0	1 bit
"t"	→	10	2 bits
"a"	→	11	2 bits

Frequent chars get shorter codes → fewer bits total

CONTEXT MODELLING

Brotli doesn't use ONE Huffman table — it uses up to 256 context-specific tables.

Context → Prediction

After "<" → HTML tag name likely

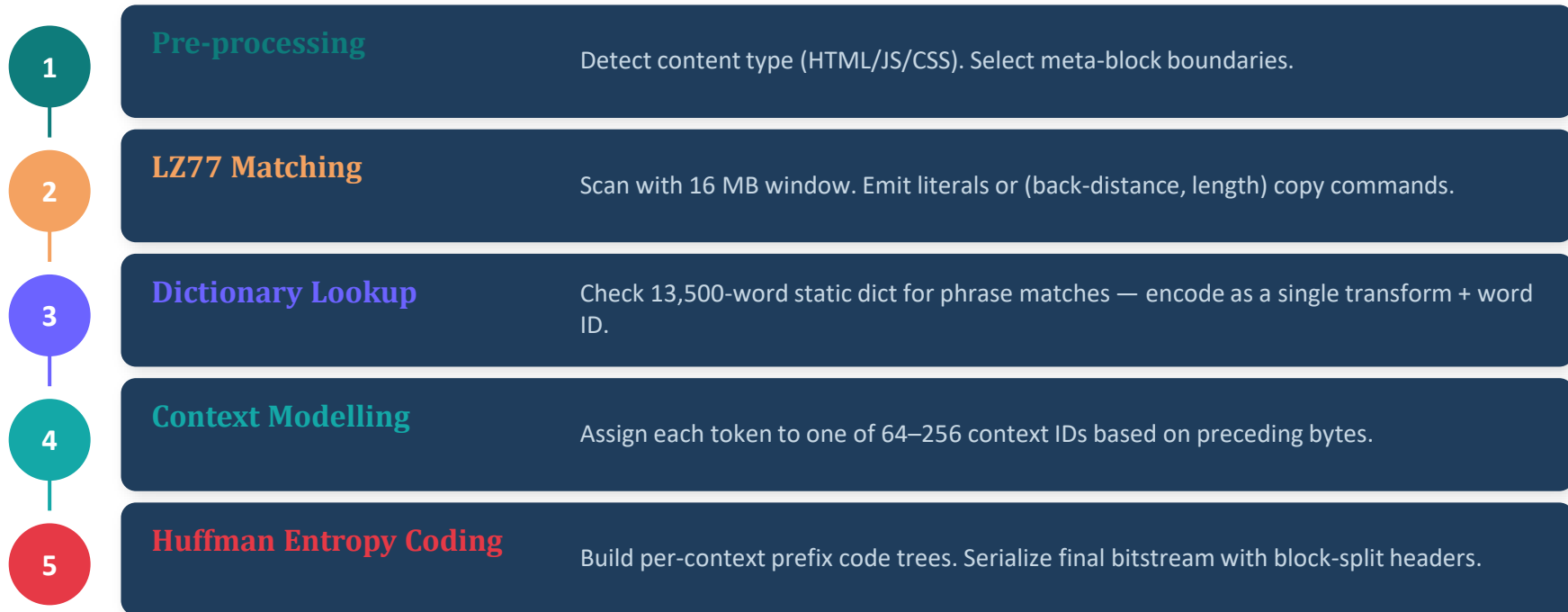
After "var " → JS identifier likely

After ":" → CSS value likely

After "http" → URL characters likely

Each context gets its own optimal Huffman table — so common patterns in HTML, CSS, and JS each compress with a table tuned to them

Five stages, from raw bytes to a compressed bitstream





ALGORITHM DEEP DIVE · 02

Zstandard

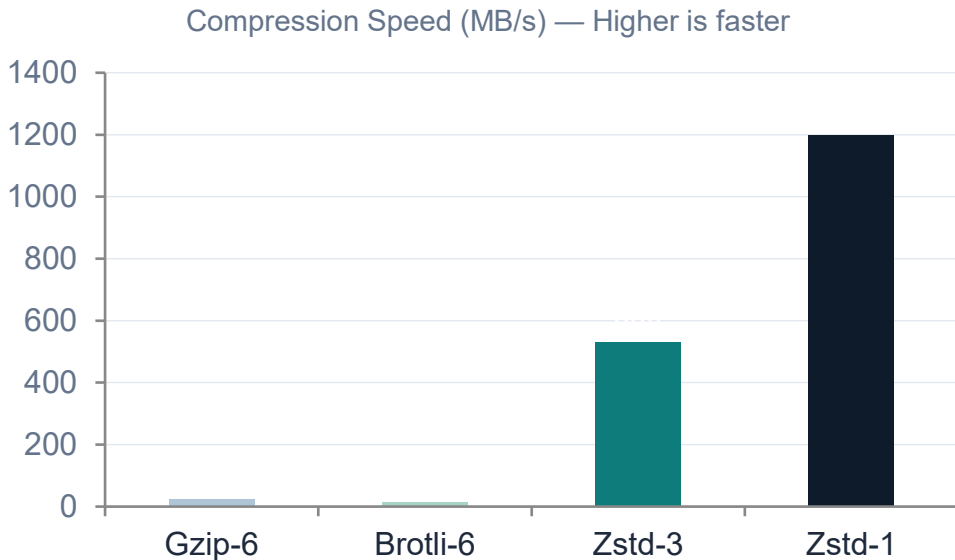
How it compresses

LZ4-style matching + FSE entropy coding + trained dictionaries — built for throughput

ZSTANDARD

Zstd: Facebook's Answer to Real-Time Compression

Designed for high-throughput data pipelines, APIs, and edge caching



API Payloads

JSON responses compressed 5-10× with near-zero latency overhead



Log Streaming

Meta uses Zstd for all internal log pipelines at 10+ GB/s



Edge Caching

Cloudflare & Fastly support Zstd; activate via Vary: Accept-Encoding

Block Parsing, LZ Matching & FSE Entropy

Block Splitting

1

Input divided into independent blocks (128 KB default). Each block compressed stand-alone — enables streaming and parallel decode.

LZ Sequence Search

2

Hash table maps 4-byte fingerprints to recent positions. Finds (offset, match-length) pairs. Hash updates in $O(1)$ — no tree traversal.

FSE Entropy Coding

3

Replaces Huffman with Finite State Entropy (tANS). Handles non-power-of-2 symbol frequencies with near-zero overhead. SIMD-friendly.

FSE vs. Huffman

Aspect

Huffman

FSE (Zstd)

Granularity

Integer bits only

Fractional bits possible

Frequency

Power-of-2 optimal

Any frequency table

Speed

~300 MB/s

~800 MB/s (SIMD)

Ratio

 ≥ 1 bit/symbol waste

Near-entropy limit

Table size

Recursive tree

Flat state array

Teaching Zstd Your Data's Language

Unlike Brotli's static web dictionary, Zstd can train a custom dictionary on your exact dataset.



Sample Data

1000+ representative records from your dataset



Train

`zstd --train samples/`
Outputs a dict file



Dictionary

Compact shared prefix table (≤112 KB)



Compress

Each record compressed using shared dict

Real-World Dictionary Gains (small records)

Content Type	Without Dict	With Dict	Gain
Small JSON records (< 1 KB)	30–40%	60–80%	+40%
HTTP API responses	25–35%	55–70%	+35%
Log line batches	50%	75–85%	+30%

Your 4-Phase Compression Roadmap

01

Audit

Week 1

Lighthouse + WebPageTest. Identify top-10 largest assets and their formats.

02

Images

Week 1-2

Convert images to AVIF/WebP using Sharp or Squoosh CLI. Use <picture> for fallbacks.

03

Server

Week 2-3

Enable Brotli on nginx/Apache/CDN. Pre-compress static assets. Set Cache-Control headers.

04

Validate

Week 3-4

Re-run audits. Measure LCP, TTFB delta. A/B test and document savings.



ALGORITHM DEEP DIVE · 03

AVIF & WebP

How they compress pixels

Transform coding, chroma subsampling, and perceptual quantization — the science of seeing less

AVIF & WebP: The Image Revolution

WebP

2010 — Google

25–35% vs JPEG smaller

96% browser support

- ✓ Lossless & lossy
- ✓ Transparency (alpha)
- ✓ Animation support
- ✓ Wide CDN & CMS support

Adopt Now

AVIF

2019 — AOMedia

50–70% vs JPEG smaller

90% browser support

- ✓ Best-in-class compression
- ✓ HDR & wide color gamut
- ✓ Film grain synthesis
- ✓ Smaller at same quality

Best Quality

JPEG XL

2022 — ISO

50–60% vs JPEG smaller

~60% (growing fast)

- ✓ Lossless JPEG re-encoding
- ✓ Progressive decode
- ✓ Very high quality ceiling
- ✓ Future standard

Watch Closely

The Pipeline Every Modern Codec Shares



Colour Space Conversion

RGB → YCbCr.
Separates luma (Y) from chroma (Cb/Cr).
Eye is 4× more sensitive to brightness than colour.



Chroma Subsampling

Chroma channels stored at $\frac{1}{2}$ or $\frac{1}{4}$ resolution (4:2:0).
Biggest single savings step — ~50% size reduction alone.



Block Transform

Image split into 4×4 or 8×8 tiles. Each tile run through DCT or ADST — converts spatial data to frequency domain.



Quantization

High-frequency coefficients rounded aggressively. Quality setting controls the quantization step size.



Entropy Coding

Quantized coefficients compressed with arithmetic coding (AVIF/WebP) or Huffman (WebP legacy). Final bitstream.

Where AVIF Goes Further Than WebP

Same foundation — but AVIF replaces the block transform, prediction, and entropy coding with AV1's superior tools.

WebP

Transform	8×8 DCT (like JPEG)
Intra-pred	VP8: 10 prediction modes
Entropy	Huffman (lossy) / LZ77+Huffman (lossless)
Tile size	16×16 macro-blocks
Color depth	8-bit per channel
HDR	Not supported
Loop filter	Deblocking filter

AVIF (AV1)

Transform	4×4 to 64×64 DCT + ADST + WHT
Intra-pred	AV1: 56 prediction modes
Entropy	Asymmetric Numeral Systems (ANS)
Tile size	Variable superblocks (128×128 max)
Color depth	8, 10, 12-bit per channel
HDR	HDR10 + Dolby Vision support
Loop filter	CDEF + Loop restoration + Deblock

AVIF's 56 intra-prediction modes + ANS entropy coding + variable superblocks = 15–25% better ratio than WebP at identical perceptual

Who Did It — and What They Gained

Shopify

 Switched product images to WebP

→ 60% image size reduction; 2.1s faster LCP

Netflix

 AVIF for all artwork thumbnails

→ 56% bandwidth savings, improved mobile UX

Facebook

 Zstd across all data pipelines

→ 25% network cost reduction at 10+ GB/s

Google

 Brotli on all Google static assets

→ ~26% smaller JS/CSS vs Gzip baseline

Pinterest

 WebP + lazy loading across image grid

→ 40% reduction in page weight

Start Small. Ship Faster.



Page bloat is a revenue problem, not just a performance metric



Brotli replaces Gzip — enable it today, gain 20-26% for free



AVIF cuts image weight by 50-70% — the single biggest win available



Zstd for APIs and data pipelines — where speed matters most



A 4-week sprint can cut total page weight by 40-60%