THE SHANNON BENCH

Workbook

Sixteen chapters from the bit to the model — exercises at the bench, the theory underneath, and the history around it.

Shannon Bench · every number is measured, nothing is decorative

The companion to the 16 lessons, one chapter per lesson. Each chapter has four parts:

- **At a glance** — the short version, if you only have a minute.
- **Exercises** — to do *at the bench*. Every exercise states what "done" looks like; solutions are sketched at the very end, not next to the exercise. Predict first, measure second — that is the whole method.
- **In depth** — the theory under the instrument: the formula, its assumptions, and where it sits in the 1948 paper.
- **Fun facts & trivia** — history and connections, strictly optional.

Every number printed in this book is produced by a tested engine function of the bench (`src/engine/`); nothing here is decorative. Amber is your payload, blue is the machinery protecting it — in the book as on the bench.

Chapter 1 — The Bit

At a glance

Every message, reduced far enough, is a stack of yes/no answers, and the bit is the smallest possible answer. A symbol drawn from 8 equally likely choices costs $\log_2 8 = 3$ bits; a symbol you can predict perfectly costs nothing. Load ALL ZEROS and the needle sits at 0.00 — certainty says nothing.

Exercises

1.1 Load ALL ZEROS (16 zeros, needle at 0.00). Predict: if you type a single `1` at the end, where does the needle land — nearer 0.1, 0.3, or 0.5? Type it and check against the gauge.

1.2 Build a message whose needle reads exactly 1.00 without using the RANDOM BITS preset. What is the shortest such message? Verify on the gauge.

1.3 The gauge tops out at 5 bits/symbol. Using only the deck's alphabet (A–Z, 0–9, space — 37 symbols), what is the highest reading you can possibly produce, and how many symbols does it take? *(The needle answers $-\log_2$ of nothing: it averages. Think uniform.)*

In depth

Shannon's first move in 1948 was to make "information" a measurable quantity and to pick its unit. A fair yes/no question — one binary digit — is the natural atom, because any choice among N equally likely possibilities decomposes into $\log_2 N$ of them. The needle generalizes this to *unequal* possibilities: it shows $H = \Sigma\, p\cdot(-\log_2 p)$, the average number of ideal yes/no questions per symbol. Two properties make H the right measure: it is zero exactly when the message is certain, and it adds up — two independent choices cost the sum of their costs. Chapter 6 shows the practical face of the same number: H is the floor no compressor can beat.

The gauge carries two readings on purpose. The amber needle measures the message on the deck — an *empirical* entropy, recomputed on every keystroke. The cool mark is the *designed* entropy of the source the message claims to come from. Short samples read low (a 16-symbol sample cannot show a rare symbol's true weight), and the gap between needle and mark is not an error: it is the difference between a message and a source, which is the first lesson of the whole bench.

Fun facts & trivia

- The word **bit** — *binary digit* — was coined by John Tukey at Bell Labs; Shannon's 1948 paper is its first appearance in print.
- Shannon considered calling H "uncertainty". John von Neumann reportedly told him to call it **entropy**: "no one really knows what entropy is, so in a debate you will always have the advantage."
- One fair coin flip per second, forever, is a source of exactly 1 bit/s. Your 4K video stream is roughly 25 million of those coins.

Chapter 2 — Surprise

At a glance

Information is what you didn't see coming. A symbol with probability p carries $-\log_2 p$ bits: halve a symbol's probability and it carries exactly one bit more. On THE ZEBRA, the rack prices E (2 of 9 symbols) at 2.17 bits and Z (1 of 9) at 3.17 — Z's lamp burns brighter because Z is rarer. Entropy is the average of that surprise.

Exercises

2.1 Type THE ZEBRA. Predict: add one more Z — does Z's price rise or fall, and does the needle (the average) rise or fall? Check both on the rack and the gauge.

2.2 Find a 10-symbol message in which one symbol costs more than 3.3 bits. What is the cheapest symbol in the same message, and why can't the two prices be equal?

2.3 The rack's lamps light the *rarest* symbols — unless every count is equal, in which case none light. Build a message where every lamp would have to light, and explain what the rack does instead.

In depth

The price $-\log_2 p$ is not an arbitrary choice; it is the only continuous price with the property that independent surprises add: if two events are independent, $P(\text{both}) = p \cdot q$, and $-\log_2(p \cdot q) = -\log_2 p - \log_2 q$. Rarity is a property of *your message*, not of the letter — add a second Z and Z gets cheaper (2.32 bits in a 10-symbol message), because the rack re-prices from the deck's own counts on every keystroke.

Entropy $H = \Sigma\, p \cdot (-\log_2 p)$ is then simply the *expected* bill. It is maximal when all symbols are equally likely (nothing is expected, so everything surprises) and zero when one symbol is certain. Everything the bench does downstream — the

compressor's tree, the channel's capacity, the model's bill — is arithmetic on this one price list.

Fun facts & trivia

- The rack is a running average of what newspaper typesetters knew by hand: E is cheap, Z is dear. A case of lead type held about ten times as many E's as Z's.
- Scrabble's letter values (E = 1, Z = 10) are a folk $-\log_2 p$, priced from letter frequencies of a dictionary — sixty years before anyone wrote the formula down.
- "Surprisal" as a name for $-\log_2 p$ is due to Myron Tribus (1961). Physicists call the same quantity self-information.

Chapter 3 — The Redundancy of Language

At a glance

English is mostly scaffolding: delete letters and the sentence still reads — proof the letters weren't all carrying information. In isolation, English letters run ≈4.1 bits each (the gauge's cool REF mark at 4.14). With context — spelling, grammar, sense — Shannon measured ≈1.3. Roughly three quarters of English is structure holding the message steady.

Exercises

3.1 Load ENGLISH TEXT (INFORMATION IS SURPRISE). The needle reads 3.50 against the mark's 4.14. Both numbers describe English letters — why do they disagree? *(One is a 23-symbol sample; one is the language.)*

3.2 Delete the vowels from the deck (backspace-edit to NFRMTN S SRPRS). Predict first: does the needle go up or down? Then check — and separately note whether *you* can still read the message. Which of the two drops faster, the needle or your comprehension?

3.3 TH_ S_NT_NC_ ST_LL R__DS. Estimate what fraction of letters you can delete from an English sentence before it stops being recoverable. Test your estimate on a friend with a sentence of your own.

In depth

The needle only sees letter frequencies — first-order statistics. It cannot see that Q is always followed by U, or that INFORMATIO_ has only one plausible ending. Those constraints are *redundancy*: parts of the message that carry no new information because the reader could have filled them in. Shannon's estimate — ≈4.1 bits/letter in isolation falling to ≈1.3 with context — means English text could, in principle, be compressed to about a third of its letter-by-letter cost. Chapter 4 measures a slice of that gap on this bench; Chapter 6 builds the machine that cashes in the first-order part.

Redundancy is not waste. It is what lets you read a typo'd word, hear a sentence over a bad phone line, and solve exercise 3.3 at all. Natural language evolved its own error-correcting code — the channel room (chapters 7–11) builds the engineered version of the same idea.

Fun facts & trivia

- Shannon's redundancy estimate implies crossword puzzles are possible: a language with no redundancy would make every letter grid a valid fill, and a language with too much would make none. English sits in the playable middle, and Shannon said so in print in 1948.
- Abjads — writing systems of Hebrew, Arabic — drop most vowels as a matter of course: exercise 3.2 is how hundreds of millions of people read every day.
- Text messaging's "txtspk" rediscovered the same ≈75%: u cn rd ths.

Chapter 4 — The Guessing Game

At a glance

Cover the page and guess the next letter — you'll be right far more often than letter frequencies allow. Shannon made a game of that in 1951, and the game is a measuring instrument: the tally of how many guesses each letter took converts into bounds on the entropy of the language *in your head*. The bench's CONTEXT reading does the machine version: H(next|prev) from your message's letter pairs. On WEATHER CHAIN the needle reads ≈1.00 but CONTEXT reads ≈0.50 — one symbol of memory halves the bill.

Exercises

4.1 Load RANDOM BITS, note the needle. Load WEATHER CHAIN, note the needle. Explain how the same needle reading (≈1.00 on both) can describe one source that no code can shrink and one that a code with memory can cut roughly in half. Which instrument on the rail tells them apart?

4.2 Play the Guessing Game (source room tab) on The Bench passage for at least 40 letters. Report your mean guesses and the H ≤ bound. Then hand the keyboard to someone else on the same passage — whose head holds the better model of English?

4.3 Load ENGLISH TEXT and read the Prediction section: CONTEXT shows a low number but flags THIN. Compute why: how many sliding pairs does a 23-symbol message have, and how many cells does its 13-letter alphabet's pair table have? At what message length would the flag clear for this alphabet (rule: pairs ≥ 3·distinct²)?

4.4 Open the Approximations tab and CRANK at order 0, then order 3. The F-series above the output falls $F_0 \geq F_1 \geq F_2$ — memory lowers the rate. Paste a page of your own prose as corpus: where does *your* F_2 land against the book's?

In depth

The 1951 paper's trick is that a human predictor can be used as a measuring device without opening the head. If the subject's guesses were generated by an ideal predictor, the fraction q_i of letters solved on the i-th guess bounds the source entropy:

$$\sum i\cdot(q_i - q_{i+1})\cdot\log_2 i \quad \le \quad H \quad \le \quad -\sum q_i\cdot\log_2 q_i$$

The bench computes both bounds live from your tally (they are labelled as computed *from your tally* — a human can violate the ideal-predictor assumption, and the instrument says so rather than pretending).

The CONTEXT reading is the smallest machine version: a bigram table. H(next|prev) = Σ p(prev)·H(next|prev = that symbol), estimated from the message's sliding pairs. For a two-symbol Markov source, 63 pairs fill a 2×2 table many times over and the estimate is stable (WEATHER CHAIN: 0.50 against the chain's designed rate of 0.54). For English, a 64-symbol deck gives 63 pairs against a table of hundreds of cells — the estimate *memorizes* the sample instead of measuring the language, so the instrument flags THIN. That flag is the chapter's real lesson: an estimate without enough data isn't a small truth, it is a confident lie, and honest instruments refuse to tell it.

Fun facts & trivia

- Shannon's 1951 subject — never named in the paper — was Mary Elizabeth "Betty" Shannon, his wife, herself a Bell Labs computer. Household entropy measurements ran to 100 letters of context.
- The upper bound at 100 letters of context came out near 1.3 bits/letter; modern neural language models judged the same way (chapter 16) sit close to 1 bit per character on English — seventy years to shave half a bit off a parlor game.
- The Mind Reader tab is the same idea inverted: a 1953-style machine (Hagelbarger's, then Shannon's improvement) predicts *you*. Against a truly fair coin it can only break even; against a human it wins, because humans are terrible at being random.

Chapter 5 — The Code Forge

At a glance

Two laws make a table of codewords a code. Prefix-freedom: no codeword may begin another, or the received stream is ambiguous. The Kraft budget: the shares 2^−length, summed over the table, may never exceed 1 — and McMillan proved the same bound for EVERY uniquely decodable code, prefix or not. Your bill is L = Σ p·length; entropy H is the floor. On THE FOUNDRY every probability is a power of two, so the best table lands on H exactly.

Exercises

5.1 Load THE FOUNDRY, open the CODE FORGE tab and write the table A: `0`, B: `10`, C: `110`, D: `1110`, E: `1111`. Done when the status line reads *L = H exactly* — budget 1.00/1.00, gap 0.00.

5.2 Now beat it: find ANY table with L below 1.88 that keeps both lamps dark. Done when you can say, in one sentence, which lamp stops every attempt and why. *(Careful — this exercise has no winning move.)*

5.3 Type `THE ZEBRA` into the deck and write a decodable table for its eight symbols by hand. Then check your L against the Huffman tab's average. Done when your table is legal and you can name the gap.

In depth

Kraft (1949) proved the budget for prefix codes in both directions: every prefix code obeys $\Sigma\, 2^{-\ell_i} \le 1$, and for every length list obeying it, a prefix code exists — the proof is a walk down a binary tree, spending subtrees. McMillan (1956) extended the bound to all uniquely decodable codes, which is why the forge's two lamps are one law: a table over budget cannot be uniquely decoded no matter how ingenious the decoder. From the budget, Gibbs' inequality forces L ≥ H for every legal table — the source-coding theorem's converse, met personally in exercise 5.2. On dyadic sources (all $p_i = 2^{-k}$) the optimal lengths ℓ_i

$= -\log_2 p_i$ are integers, so L = H exactly; everywhere else the integer constraint costs strictly less than one bit, which is The Compressor's territory.

Fun facts & trivia

- The budget is a master's thesis: Leon Kraft wrote the inequality at MIT in 1949. Brockway McMillan's 1956 strengthening came out of Bell Labs — the same building as the theorem it serves.
- Arithmetic coding (1970s) sidesteps the integer-length constraint by coding the whole message as one long fraction — it pays the sub-bit fractions the forge's table cannot, and modern video codecs live on it.
- The forge's "budget bar" picture is literal: prefix codes are exactly the ways to cut a unit interval into dyadic pieces. Your codeword IS its interval's address.

Chapter 6 — The Compressor

At a glance

If common symbols got short names and rare ones long names, messages would shrink. Huffman's tree does exactly that, provably as well as any prefix code can: merge the two lightest nodes until one root holds everything; each symbol's codeword is its path from the root. On HELLO WORLD: 88 bits as ASCII, 33 as a fixed 3-bit code, 32 as Huffman — and L (×3) gets a 2-bit word while D (×1) pays 4.

Exercises

6.1 On HELLO WORLD, the needle reads 2.85 and Huffman averages 32/11 ≈ 2.91 bits/symbol. Why can't the average be 2.85 exactly? *(Look at the codeword lengths — what kind of numbers are they?)*

6.2 Type LLLL after HELLO WORLD. Predict: does Huffman's *lead* over the fixed code grow or shrink? Check the counters, then explain the result in one sentence about skew.

6.3 Open the Beat Huffman tab and build your own tree for HELLO WORLD by merging roots. The bench bills any full binary tree honestly. Can you beat 32 bits? Can you *tie* it with a tree that is shaped differently from the compressor's? *(BUILD_NOTES #7 is a hint that ties exist.)*

6.4 A message of one repeated symbol (say AAAA) has H = 0 but Huffman still spends 1 bit per symbol. Verify on the bench, then explain where the wasted bit comes from.

In depth

Huffman's algorithm is greedy and yet optimal: repeatedly merging the two lightest weights builds the tree with the minimum weighted path length, which is exactly the total bill in bits. The prefix property — no codeword is a

prefix of another — comes free, because symbols live only at the leaves; that is what lets the bit log decode a stream with no separators.

Two limits frame the machine. Below: no code of any kind can average under H bits/symbol (that is the source coding theorem — the needle is a legal floor, not a hint). Above: Huffman wastes less than 1 bit/symbol against H, because codeword lengths must be whole numbers (exercise 5.1's answer). The waste is worst exactly where exercise 5.4 lives — a certain source, H = 0, still pays 1 bit — and shrinking it is what arithmetic coding was later invented for.

One honest caveat: the compressor prices symbols *in isolation* (first-order). It can never cash in the context that chapter 4 measured — on WEATHER CHAIN, Huffman refuses (1 bit/symbol, no gain) while CONTEXT reads 0.50. Coding with memory is real (and is how ZIP works), but it is a different machine than the one on this canvas.

Fun facts & trivia

- David Huffman invented the code in 1951 for a term paper in Fano's MIT class, to get out of the final exam. Fano and Shannon had both worked on the problem and gotten only the top-down (suboptimal) version; the student's bottom-up merge was optimal.
- Morse code is a pre-mathematical Huffman: E is one dot. Morse estimated frequencies by counting a printer's type cases.
- Huffman coding is inside JPEG, MP3, ZIP/DEFLATE and PNG to this day — usually as the final stage after a transform exposes the skew.

Chapter 7 — The Incompressible

At a glance

Compression is a bet against uniformity, and some messages call the bluff. Load RANDOM BITS: HUFFMAN equals FIXED (32 = 32) and the rail says "compression honestly refuses". Load LOADED DIE and skew is back: 30 bits against the fixed code's 48. Entropy is the floor, and random data already lives on it.

Exercises

7.1 On RANDOM BITS the needle reads 1.00 (well, 0.997 — count the ones and zeros). Why does Huffman assign 1-bit codewords and stop, and why is that the honest optimum rather than a failure?

7.2 LOADED DIE: 16 throws, needle at 1.80, fixed code 48 bits, Huffman 30. Multiply the needle by the message length. What does the result (≈28.8) tell you about how much of Huffman's possible win is still on the table?

7.3 Try to type a 20-symbol message that Huffman compresses to less than *half* the fixed code's bill. What two properties must your message have? Show that with only 4 distinct symbols it cannot be done at all.

In depth

The floor is not Huffman's — it is the source's. Shannon's source coding theorem: no lossless code can average below H bits/symbol, and codes exist that come within a vanishing margin of H on long messages. A uniform source over 2 symbols has H = 1, and its Huffman code *is* the fixed code; there is nothing to squeeze because no symbol is more expected than another. This is why "random data is incompressible" is a theorem, not an engineering complaint — and why compressing a file twice gains nothing: a good compressor's output is close to uniform by construction, or it would have kept compressing.

The LOADED DIE shows the other side of the ledger. Its designed distribution (½, ⅛, ⅛, ⅛, 1⁄16, 1⁄16) has H = 2.125; the 16-throw sample measures 1.80, so the entropy floor for the whole message is ≈28.8 bits. Huffman lands at 30 — within 1.2 bits of the floor, and 18 under the fixed code. The gap between 30 and 28.8 is the whole-number-lengths tax from chapter 6.

Fun facts & trivia

- Kolmogorov generalized the idea: the complexity of a string is the length of its shortest description. Most strings have no description shorter than themselves — incompressibility is the *typical* case, which is why compressible data (language, images) is the interesting exception.
- Every "compress your files by 90%, guaranteed, any file" patent application fails against a counting argument three lines long: there aren't enough short strings to go around.
- Lottery-number "systems" die on this chapter: a fair draw has maximal entropy, and no pattern-finder can average a single bit of advantage.

Chapter 8 — Noise

At a glance

So far every bit arrived as sent; the channel room takes that promise away. The wire flips each bit with probability p — the noise fader. Set NONE, noise 0.10, press SEND: struck cells flash on the wire, and with no code armed, BER OUT equals BER RAW — nothing stands between the noise and your payload. Expect about p·n wrong bits in an n-bit message.

Exercises

8.1 Arm NONE, set noise to NOISY (0.10), SEND a 40-bit payload. Predict the number of flips before looking (p·n = 4), then count the ⚡ cells. Send again — same count? Why not, and what number *is* stable across many sends?

8.2 The bench's sends are seeded: RESET truly rewinds the world, and the same clicks reproduce the same flips. Verify by RESET + SEND twice. Why would an honest instrument *choose* deterministic noise?

8.3 With NONE armed and any p > 0, the capacity box already pulses ABOVE CAPACITY. Rate 1.0 over a noisy wire violates what, exactly? (You will meet the theorem in The Limit — for now, find the two numbers the box is comparing.)

In depth

The binary symmetric channel is the simplest noise model there is: each bit flips independently with probability p, and — the treacherous part — a flipped bit looks exactly as confident as a true one. There is no smell of error on a single bit; errors only become visible against *structure*, which is precisely what the codes of the next chapters add.

The two BER instruments frame everything that follows. BER RAW counts flips on the wire — it tracks the fader, and no code can change it. BER OUT counts payload bits still wrong after decoding. A code earns its keep exactly in the gap between the two; with NONE armed the gap is zero by definition. Keep an eye

on that pair through chapters 8–9: repeat opens the gap at a terrible price, parity only *sees* errors, Hamming opens it with surgical cheapness — and chapter 15 says how wide the gap can ever be opened.

Fun facts & trivia

- Real channels earn the model: deep-space probes at the edge of link budget, DRAM cells hit by cosmic rays, scratched CDs. The BSC's p is the single number a link engineer quotes first.
- The bench's noise is seeded (mulberry32, seed 0x5EED + send id) so the trace scrubber can *replay* history exactly — determinism is what makes the past inspectable (BUILD_NOTES #10).
- A single cosmic-ray bit flip in a Belgian voting machine in 2003 gave a candidate 4,096 extra votes — noticed only because the total exceeded the electorate. Structure reveals errors; raw bits hide them.

Chapter 9 — Redundancy on Purpose

At a glance

Language survives typos on accidental redundancy; codes add it on purpose. REPEAT×3 sends every bit three times and lets the copies vote: rate ⅓, but one flip per triple is outvoted. PARITY spends one bit per four to *smell* odd numbers of flips — it detects, but cannot point. The decoder panel shows every election and every alarm, live.

Exercises

9.1 Arm REPEAT×3 at WHISPER (0.05) and SEND until you see a healed row. Click it: the panel shows the triple's vote. Now compute the odds a triple lies: with p = 0.05, what is the probability of 2 or 3 flips in one triple? *($3p^2(1-p) + p^3$ — about 0.7%.)*

9.2 Arm PARITY and send until a row reads *flagged · can't fix*. The alarm rang — why can't the decoder repair what it detected? What exactly does it know, and what would it need to know?

9.3 Parity's blind spot: what is the smallest number of flips in one block that parity misses entirely? Force the case: use the custom nibble and the hand-flip tools to build a send where the check passes and the payload is wrong.

9.4 Repeat×3 at STORM (0.20): predict whether BER OUT stays under BER RAW, then run twenty sends and check the stats. Where did the votes start lying?

In depth

Both chips are one idea — spend extra bits to make valid codewords *far apart* — at opposite corners of the design space. Repeat×3's codebook is {000, 111}: any single flip leaves you nearer the truth than the lie, so majority vote corrects it. The price is brutal: rate ⅓, and two flips in a triple (probability $3p^2(1-p) + p^3$) outvote the truth *silently* — the decoder delivers a wrong bit with full confidence. Parity's codebook is all 5-bit words with an even number of ones:

any odd number of flips lands outside the codebook and rings the alarm, but every neighbour of a received word is equally plausible, so the decoder can detect and not correct — a fire alarm that can't find the fire.

The vocabulary the bench uses is precise (BUILD_NOTES #13): *healed* means the payload arrived intact despite flips; *flagged* means the decoder detected an error it could not fix (parity's specialty); *lost* means wrong payload delivered — including repeat's outvoted truths and (next chapter) Hamming's confident miscorrections. Detection you can act on (ask for a resend — that is what TCP does); silent wrongness you cannot. Chapter 10 buys correction cheaply; chapter 15 prices the whole market.

Fun facts & trivia

- Say-everything-twice is older than coding theory: aviation's "niner" and the NATO alphabet are repeat codes tuned for a voice channel's known confusions.
- Parity bits guarded punched-card decks and core memory from the 1950s on; the ninth bit on old RAM modules is exactly this chip.
- RAID storage still speaks this chapter's language: RAID 1 is repeat×2; RAID 5 is a parity stripe. Two flips in the wrong places and either can lie to you — the math didn't change.

Chapter 10 — The Healer

At a glance

Repeat×3 pays 200% overhead to fix one flip; Hamming (7,4) fixes it for 75% — by making the parity bits triangulate. Three checks each watch four of the seven positions, and their verdicts, read as a binary number $S_3S_2S_1$, *are* the address of the flipped bit. Codeword 0110011 with bit 5 flipped reads syndrome $101_2 = 5$: the checks spell the crime scene.

Exercises

10.1 Type 1011 into the custom nibble, note the codeword (0110011), and STEP until noise strikes. Read the decoder panel: C1, C2, C3 recomputed from the received bits, mismatches highlighted, address assembled. Verify the arithmetic of one check by hand.

10.2 Work out on paper why the parity bits live at positions 1, 2, 4. What is special about those position numbers in binary, and why does that make the syndrome *be* an address?

10.3 Hamming corrects any single flip in seven bits. Two flips, and the syndrome lies with confidence — it points at a third, innocent bit. Force this with the hand-flip tools and watch the panel tag "healed the wrong bit" (the panel knows the ground truth; the chip never does — BUILD_NOTES #14). What would the chip need to at least *detect* the double flip?

10.4 Open Your Chip and rewire the 3×4 coverage toggles. Find a wiring where two different single flips produce the same syndrome — the panel flags the ambiguity instead of guessing. What property of Hamming's wiring guarantees this never happens?

In depth

The syndrome trick is positional: number the seven cells 1–7 and put check bit C_k at position 2^{k-1}, watching every position whose binary address has bit k set. A

flip at position j then fails exactly the checks named by j's binary digits — so reading the three verdicts as a number *is* reading the address. Distance explains why it works and where it ends: every pair of valid codewords differs in at least 3 positions (minimum distance 3), so one flip leaves the true codeword uniquely nearest — and two flips leave a different codeword nearest, which is the confident miscorrection of exercise 9.3. One extra parity bit over the whole block (the extended (8,4) code) buys double-flip *detection*; pointing at two flips needs distance 5 and more machinery.

Rate is the other axis: 4/7 ≈ 0.571 against repeat's 0.333, for the same single-flip healing. That is the whole direction of coding theory — distance per redundant bit — and chapter 15 will show the ceiling this race runs under: at NOISY (0.10), capacity is 0.531, and 0.571 is *above* it. The healer that saves every demo block cannot save a long message on that wire; the pulsing box already knows.

Fun facts & trivia

- Richard Hamming built the code out of irritation: Bell Labs' relay computer ran his weekend jobs unattended, and parity checks kept *detecting* errors and aborting the run. "If it can tell something is wrong, why can't it tell me where?" — the 1950 paper answers with the address trick.
- ECC server memory runs Hamming's extended (72,64) descendant on every read cycle: same syndrome idiom, 12.5% overhead, invisible until the logs show corrected cosmic rays.
- QR codes heal in public: print one, ink out a corner, and it still scans — Reed–Solomon, the heavy-duty cousin, doing chapter 10's job on bursts instead of single flips.

Chapter 11 — The Arena

At a glance

A 32×32 image crosses the wire in 256 four-pixel blocks. Uncoded, every wire flip is a wrong pixel, and the binary symmetric channel flips, on average, a fraction p of everything it carries: expect p·1024 wrong pixels, with a spread around it that RE-DEAL makes visible. A picture is the one payload where a bit error rate needs no explaining.

Exercises

11.1 Arm NONE, load the Smiley, set p = 0.05 and RE-DEAL five times, recording WRONG PIXELS each run. Done when your five counts straddle the expectation p·n ≈ 51.

11.2 Draw your own payload on the SENT canvas and raise p slowly from zero. Done when you have found YOUR recognition threshold — the p where your own drawing stops being yours — and can say why the decay is gradual rather than a cliff.

11.3 Load the Checker and push p to 0.50. Done when you can explain what the received frame shows and why p = ½ is the worst possible wire — worse than p = 0.9.

In depth

Each pixel survives independently with probability 1−p, so the wrong-pixel count is binomial: mean np, variance np(1−p) — at p = 0.05, n = 1024, that is 51.2 ± 7 or so, exactly the wobble RE-DEAL shows. At p = ½ the received bit is independent of the sent bit: mutual information zero, the wire a perfect eraser (Chapter 14 prices this). Beyond ½ the wire becomes a reliable liar — flip everything back and it is a good channel again, which is why every instrument here tops out at 0.5. The Arena's dealer is seeded per block from one master seed, code-blindly,

so a run replays exactly and different chips face the same storm — the fairness Chapter 13 stands on.

Fun facts & trivia

- Mariner 4's 1965 Mars pictures came down at 8⅓ bits per second, uncoded; engineers at JPL famously hand-colored a printout of the raw numbers faster than the official processing produced the image.
- Analog television "snow" is the same experience pre-digital: a signal drowning in additive noise, every speck honest.
- The 32×32 one-bit frame is not a toy size: early network icons and cursors were exactly this, 128 bytes each.

Chapter 12 — The Armor

At a glance

Protection is a purchase. REPEAT×3 pays overhead ×3 and outvotes one flip per triple; REPEAT×5 pays ×5 and survives two; PARITY pays ×1.25 and can only flag; HAMMING pays ×1.75 and heals one flip per block of seven. Two meters settle every argument: OVERHEAD and WRONG PIXELS. The collapse past each chip's comfort zone is a curve, not a cliff — blocks gamble independently.

Exercises

12.1 At p = 0.02, duel-free: send the Smiley with NONE, note WRONG PIXELS, then arm HAMMING at the same seed. Done when Hamming's count is at most a third of naked — and you have checked the price on the OVERHEAD meter.

12.2 Work the REPEAT×3 failure rate at p = 0.10 by hand: $3p^2(1-p) + p^3$. Done when your number is ≈ 0.028 and the bench's per-pixel behavior at that noise agrees with it.

12.3 Arm PARITY at p = 0.06 and explain the blue blocks. Done when you can say what parity knows, what it cannot know, and why no amount of cleverness turns one check bit into an address.

In depth

Per block, the failure arithmetic is elementary and merciless. A triple fails when ≥2 of 3 flip: $3p^2(1-p) + p^3$. Five copies fail at ≥3 of 5: $10p^3(1-p)^2 + 5p^4(1-p) + p^5$ — at p = 0.10 that is ≈ 0.9%, three times better than ×3 for ×1.67 the price. Hamming's block of seven fails when ≥2 of 7 flip: $1 - (1-p)^7 - 7p(1-p)^6$, and unlike the votes it fails by CONFIDENTLY healing the wrong bit (chapter 10's syndrome pins this). A single parity bit sees only the flip count's parity — one bit of evidence can point at "something", never at "where". The exchange rate the meters show is the engineer's entire trade: reliability is bought in wire bits, and the Duel (Chapter

13) is where two purchases are compared honestly. What no purchase on this rail reaches is the capacity line — Chapter 15 prices the headroom.

Fun facts & trivia

- Every stick of ECC server RAM runs a Hamming descendant (SECDED: one extra parity bit makes double flips detectable instead of silently mis-healed — exactly the failure you watched at p = 0.12).
- CDs survive scratches with Reed–Solomon codes: correction not per bit but per byte-run, which is why a radial scratch is survivable and a tangential one is death.
- Deep-space missions run overheads far beyond ×5 — Voyager's Golay code paid ×2 for 3-flip healing in 24-bit blocks; modern probes pay less for more, living off the 45-year march toward capacity.

Chapter 13 — The Duel

At a glance

Two schemes, one storm: the DUEL switch runs a second lane over the same master seed, so both contenders face noise from the same dealer, block by block. Each lane answers with two numbers — OVERHEAD and WRONG PIXELS — and the comparison is the verdict on your construction, not a reading off someone else's.

Exercises

13.1 REPEAT×5 vs NONE at p = 0.05, same seed. Done when you have recorded both meter pairs and can state the paradox precisely: the armored lane suffered roughly five times the wire flips and still delivered the cleaner image.

13.2 HAMMING vs REPEAT×3 across the fader. Done when you have found the noise band where the cheaper chip stops winning — and RE-DEAL three times to check the crossover belongs to the designs, not the dice.

13.3 Wire YOUR CHIP (decoder panel, 12 switches) into something deliberately worse than Hamming — cover one data bit with only one check. Duel it against HAMMING. Done when the residual difference matches what audit's healable-positions count predicted.

In depth

Comparing two random systems on independent noise wastes most of the sample on the noise itself; running both on the SAME realization makes their difference visible almost immediately. Simulation science calls this common random numbers, and it is the Duel's whole design: the block dealer is seeded identically and code-blindly, so every difference on the meters is attributable to the codes. The honest limit — stated on the bench — is that different wire lengths cannot see bitwise-identical noise; same dealer, same deal, more cards for the longer hand is the strongest fairness that exists. Shannon's 1948

existence proof is a duel of cosmic size: average over ALL random codes at once and show the average already beats any rate below capacity — no single design named, every single design judged.

Fun facts & trivia

- Common random numbers is why flight-sim Monte Carlo comparisons and A/B backtests re-use seeds: variance of the DIFFERENCE collapses while each side's own variance stays put.
- The Duel's "two numbers each, nothing else" is deliberately the shape of an engineering datasheet: rate and residual BER are the columns real codecs are bought by.
- Shannon never built the optimal codes his theorem promised — the duel he staged was against the average of all possible codes, an opponent that cannot be out-designed, only matched. LDPC codes finally matched it within hundredths of a dB.

Chapter 14 — The Bridge

At a glance

Two rooms, one question: how many of the source room's bits actually make it across the channel room's wire? The answer is mutual information I(X;Y) — and for a fair input on the BSC it equals $1 - H_2(p)$, which is exactly the number the capacity box has shown all along. At NOISY (0.10): 0.531 bits of every wire bit survive; $H_2(0.10) = 0.469$ is eaten by the noise. The ledger always sums to one.

Exercises

14.1 Set noise to 0.10 and read C = 0.531. Account for the missing 0.469 without the word "capacity": what, physically, is the wire doing with it? *(What would you still need to be told, per bit, to undo the noise?)*

14.2 Slide the fader from QUIET toward 0.50 and watch C fall to 0.000. State in one sentence what the receiver knows about the input when p = ½ exactly. What happens *past* 0.5 — and why does the bench's capacity curve climb back up? *(capacity(1) = 1 is not a bug.)*

14.3 The engine's `mutualInformation(p, π)` takes the input skew π. At p = 0.10, a fair input gets 0.531 through; a 25/75 input gets 0.412. Why does a skewed input carry less? *(What is H(X) itself for that input?)* Confirm the maximum-at-½ claim by evaluating a few skews in the tests.

In depth

Mutual information is the meeting point of the two rooms' bookkeeping: $I(X;Y) = H(Y) - H(Y|X)$ — the output's total uncertainty minus the part the noise manufactures. For the BSC with a fair input, $H(Y) = 1$ and $H(Y|X) = H_2(p)$, so $I = 1 - H_2(p)$. The subtraction is the honest accounting of chapter 8's treachery: the wire delivers a full bit of *data* per use, but only $1 - H_2(p)$ bits of *information about the input*; the other $H_2(p)$ bits are noise wearing data's clothes.

Capacity is then a definition, not a new phenomenon: C = max over input distributions of I(X;Y). For this symmetric channel the maximum sits at the fair input (any skew lowers H(X) faster than it helps, exercise 10.3), which is why the blue box could show C all along without asking what you were sending. The definition has teeth in both directions, and that is the next chapter: below C the promise, above C the wall.

Fun facts & trivia

- The two-room split *is* the 1948 paper's table of contents: Part I, the source and its entropy; Part II, the channel and its capacity. This chapter is the staple between the halves.
- I(X;Y) is symmetric — the output says as much about the input as the input says about the output — which surprises everyone the first time, since the wire only pushes one way.
- Line rate vs. throughput in your Wi-Fi settings is this chapter live: the PHY pushes wire bits (H(Y)), you receive payload (I), and the gap is retransmissions and coding — H(Y|X) paying its bill.

Chapter 15 — The Limit

At a glance

Every bench in this family ends at a wall; this one is Shannon's. A channel that flips bits with probability p carries at most C = 1 − H_2(p) payload bits per wire bit. Below C, codes exist that come arbitrarily close to perfect. Above C, no code — present or future — can win. Both halves are proven. At NOISY (0.10), C ≈ 0.531 and Hamming's rate 0.571 pulses ABOVE CAPACITY; at WHISPER (0.05), C ≈ 0.714 and it calms.

Exercises

15.1 Find the noise level where REPEAT×3 (rate 0.333) crosses its own line — slide the fader until the pulse starts. *(The tests pin the crossing between 0.17 and 0.18.)* What does it mean, operationally, that even 200% overhead has a wire it cannot survive?

15.2 At noise 0.10 (C = 0.531), which of the bench's four chips are above the line and which below? For one chip below the line, explain what the theorem promises — and what it pointedly does *not* promise about *that chip*.

15.3 The proof of the achievability half uses *random* codebooks: a code drawn at random is, with high probability, nearly as good as the best one — yet it took 45 years to build practical codes that approach C. What does a random codebook cost that a usable one cannot? *(Think about the decoder's job, not the encoder's.)*

In depth

The theorem has two independent halves. The converse (the wall): at rate R > C, the error probability of *any* code is bounded away from zero — information the noise destroyed (H_2(p) per wire bit, chapter 14's ledger) cannot be reconstructed by cleverness, only re-sent. The achievability half (the promise): for any R < C there exist codes, of long enough block length, with error probability as small as

you like. Shannon proved they exist by averaging over random codebooks — without exhibiting a single one you could decode in your lifetime.

That gap between *exists* and *shipped* is the honest footnote on the bench (lesson "The Limit", The Fine Print): none of the four chips approaches capacity, and for 45 years nothing practical did. Turbo codes (1993) and the rediscovery of Gallager's LDPC codes closed the gap to within hundredths of a decibel. The wall, meanwhile, has never moved: it is the one instrument on the bench that judges not what the chips do, but what any chip *could* do — which is what makes it a summit and not a scoreboard.

Fun facts & trivia

- When Berrou and Glavieux reported turbo codes within 0.5 dB of the Shannon limit in 1993, reviewers assumed a measurement error — the community had priced "near capacity" as practically unreachable.
- Gallager's LDPC codes (1960 PhD thesis) were forgotten for three decades because the decoders were unbuildable at the time. Your Wi-Fi (802.11), 5G phone and satellite TV all run them now.
- Voyager 1 still phones home from interstellar space through a concatenated code at a few bits per second — link budget engineering is the art of living politely below C.

Chapter 16 — The Model

At a glance

Shannon's guessing game never ended — it moved into machines. A model that predicts the next symbol is judged by the bill it pays: $-\log_2 q(x)$ per arrival, cross-entropy on average, and it can never pay less than the source's own entropy (Gibbs' inequality). Perplexity 2^H turns the bill into a headcount. The book's letter model pays 4.21 bits on ENGLISH TEXT (perplexity 18.5) against the needle's floor of 3.50 (perplexity 11.3) — the 0.71-bit gap is the model's ignorance, priced.

Exercises

16.1 Load ENGLISH TEXT and read the Prediction section: MODEL H× 4.21, needle 3.50. Now type the book's own text into the deck... which you can't — it is 357 symbols and the deck caps at 64. Take the first 64 characters instead and compare MODEL H× to the needle. Why did the gap shrink?

16.2 Type a 7 after any message. MODEL H× and Perplexity jump to ∞. State Gibbs' inequality, then explain why *no* smoothing-free frequency model can avoid the infinite bill — and what every real language model does about it.

16.3 The identity test: the engine pins that a model trained on the sample itself pays exactly the sample's entropy. Why is that the *only* model that achieves the floor, and what does that make cross-entropy minus entropy (the gap) a measure of? *(It has a name: KL divergence.)*

16.4 Rank four of the rail's readings for ENGLISH TEXT — needle 3.50, CONTEXT (thin), MODEL 4.21, REF 4.14 — from "knows the message best" to "knows it least". Defend the ordering in one sentence each.

In depth

Cross-entropy $H(p, q) = -\Sigma\, p(x) \cdot \log_2 q(x)$ is the bill a model q pays on data from source p, and Gibbs' inequality $H(p, q) \geq H(p)$ — equality only at $q = p$ — is why

prediction is a fair exam: you cannot beat the source's entropy, only approach it by knowing the source better. The gap H(p, q) − H(p) is the Kullback–Leibler divergence, the bench's "model's ignorance, priced in bits". The ∞ of exercise 12.2 is the honest edge of the definition: an event assigned probability zero has no finite price, which is why real models smooth — reserving a sliver of probability for everything they have never seen.

This exact yardstick is how modern language models are trained and scored: the training loss of an LLM is cross-entropy per token, and "perplexity on the eval set" in every paper is 2 to that loss. The lineage is direct — Shannon 1948 defined the price, Shannon 1951 played the game with a human predictor, and today's models play the same game with a trillion parameters. The bench stops at letter frequencies on purpose: it is a 1948 instrument, and the point lands harder for it — everything after this chapter is a better q, chasing the same H.

Fun facts & trivia

- **The circle closes:** in 1937 a master's student named Claude Shannon showed Boole's algebra could live in relay circuits — the thesis Boole Bench is built on. He then spent 1948 asking what those circuits carry and 1951 asking how well it can be predicted. Three benches, one person's homework.
- Perplexity entered speech recognition in the 1970s (Jelinek's group at IBM) precisely because "2^H equally likely words" was a number managers could compare across vocabularies.
- Claude Shannon juggled, unicycled, built a flame-throwing trumpet, a Roman-numeral computer (THROBAC), and Theseus the maze-solving mouse — the first machine that visibly *learned*. The best instruments are built by people who play.

Solution sketches

1.1 17 symbols, one of them a `1`: $H = -(1/17)\log_2(1/17) - (16/17)\log_2(16/17) \approx 0.32$. Nearer 0.3 — one surprise is all it takes, but only one.

1.2 Any message with two symbols at equal counts: `01` (2 symbols) is the shortest. H = 1.00 exactly.

1.3 Uniform over all 37 deck symbols: $\log_2 37 \approx 5.21$ — just past the gauge's 5-bit scale, and it takes all 37 symbols once each (37 symbols). With the 64-cap you can also do 37 distinct + 27 repeats, but equal counts are lost — one appearance each is the clean answer.

2.1 Both fall. Z's price drops 3.17 → 2.32 (it is less rare), and the needle eases 2.95 → 2.92: a message with two repeated pairs is more predictable than one with a lone rarity. The instrument distinguishes a symbol's price from the whole message's average — they often move together, but for different reasons.

2.2 Any 10-symbol message with a 1/10 symbol: its price is $\log_2 10 \approx 3.32 > 3.3$. The cheapest symbol must then appear ≥ 2 times — two symbols can't both be the rarest at 1/10 and also differ in price.

2.3 Uniform message (e.g. ABCD): every symbol is "rarest". The rack lights none (BUILD_NOTES #6) — a lamp that always burns says nothing.

3.1 The needle measures a 23-symbol sample (it cannot contain J, Q, X, Z at their true weights — or at all); the mark is the designed first-order entropy of English at large (4.14). Samples wobble around sources.

3.2 The needle drops by about a fifth (3.50 → 2.84) — vowels were among the message's more even-handed letters. Your comprehension drops far less: NFRMTN S SRPRS still reads. The mismatch is the point: the needle prices letter statistics, and the redundancy you are reading with lives in spelling and sense, which no first-order gauge can see.

3.3 Around half, if you choose which letters (keep word onsets and consonants). Shannon's ≈75% redundancy is the long-run ceiling with all of grammar and sense as context, not per-letter vowel deletion.

4.1 The needle sees only letter frequencies — both are ≈50/50 coins to it (0.997 vs 0.997, pinned equal on purpose). The CONTEXT reading tells them apart: random ≈1.0 conditional (no memory), weather ≈0.50 (yesterday predicts today). Only the chain can be shrunk — by a code over *pairs*, not letters.

4.2 Empirical — but expect mean guesses ≈1.5–2.5 and H ≤ between 1 and 2 for a fluent reader; Shannon's subjects reached ≈1.3 with long context. The better reader of English pays fewer guesses.

4.3 22 pairs; $13^2 = 169$ cells. Rule pairs ≥ 3·169 = 507 → message length 508. The deck caps at 64, so the flag can never clear for full English on this bench — which is exactly what the lesson's footnote says, and why the Guessing Game (a human) is the English instrument.

4.4 $F_0 = \log_2(\text{alphabet})$ is fixed by the alphabet; F_1 falls to letter frequencies; F_2 uses digram memory. Prose lands near the book's numbers; repetitive text (lyrics with a chorus) drives F_2 far lower.

5.1 Kraft: ½+¼+⅛+1⁄16+1⁄16 = 1.00. L = ½·1+¼·2+⅛·3+1⁄16·4+1⁄16·4 = 1.875 = H. The dyadic case is the only one where the wall is touched exactly.

5.2 No winning move exists: any complete table with L < H must spend $\Sigma 2^{-\ell} > 1$ (Gibbs + Kraft), so the budget lamp fires; the shortcut of reusing short words fires the prefix lamp instead. The one-sentence answer: "the budget bar — L < H forces an overdraw."

5.3 THE ZEBRA has 8 distinct symbols (T,H,E,␣,Z,B,R,A — E twice). Any legal table needs $\Sigma 2^{-\ell} \le 1$ over 8 rows; a balanced 3-bit table gives L = 3.00 against H ≈ 2.95 and Huffman lands within a whisker of the needle. Gaps under 0.05 bits are the honest best here.

6.1 Codeword lengths are whole numbers; H = 2.85 would need fractional-bit words. Huffman gets 32/11 ≈ 2.91 — the best whole-number approximation to

this distribution, less than a bit over the floor (source coding theorem, integer tax).

6.2 The lead grows: the counters move to Huffman's favour as L's count climbs — more skew, more to win. One sentence: Huffman's profit is exactly the distance from uniform.

6.3 You cannot beat 32 (Huffman is optimal), but you can tie it with a differently-shaped tree: optimal trees are not unique when merge weights tie (BUILD_NOTES #7 — the mockup's hand-derived tree and the app's differ, same 32 bits, same length multiset {2,3,3,3,3,3,4,4}).

6.4 H = 0 but a codeword cannot be 0 bits long — the decoder must see *something* per symbol. Single-symbol sources get code "0" (BUILD_NOTES #8): the 1-bit floor is the prefix-code machinery's minimum fare, and the gap H = 0 vs 1.0 is the worst case of the integer tax.

7.1 Two symbols at ~50/50: the optimal tree is one root with two 1-bit leaves — which *is* the fixed code. Optimum reached, nothing to squeeze; "refusing" is Huffman reporting the truth.

7.2 16 × 1.80 ≈ 28.8 bits: the entropy floor for this exact message. Huffman's 30 is 1.2 bits above it — nearly everything winnable has been won; the remainder is the integer tax, not slack.

7.3 Two properties: extreme skew *and* at least 5 distinct symbols. With 5 distinct the fixed code pays 3 bits/symbol: 16 A's + B, C, D, E → fixed 60, Huffman 16·1 + 4·3 = 28 < 30 ✓. With only 4 distinct, fixed = 2 bits/symbol = 40, while Huffman's 1-bit-per-symbol minimum fare puts its best case (17 A's + B + C + D) at 17 + 2 + 3 + 3 = 25 > 20 — more than half is unreachable. Verify both bills on the bench.

8.1 Counts vary send to send (4 is the *expectation* of a seeded but p-faithful process); the stable number across many sends is BER RAW → p. The instrument distinguishes a rate from a count.

8.2 Reproducibility is what makes claims checkable: the trace scrubber can replay any past send exactly (records, not re-rolls), and "the same clicks give the

same world" turns anecdotes into experiments (BUILD_NOTES #10).

8.3 It compares rate R = 1.0 against C = 1 − $H_2(p)$ < 1. Rate 1 over any noisy wire is above capacity — a theorem violation, not a warning (BUILD_NOTES #12).

9.1 $3(0.05)^2(0.95) + (0.05)^3 = 0.007125 + 0.000125 \approx 0.00725$ — about 0.7% of triples lie, silently. At 40 payload bits that is a lost row every few storms.

9.2 The check says "the count of ones is odd — something flipped". All five positions are equally suspect: detection without localisation. It would need more structure (more checks watching *different* subsets) — which is precisely Hamming's move.

9.3 Two flips: even count again, check passes, payload wrong. Hand-flip any two cells of one parity block; the log keeps the tampered row honest (BUILD_NOTES #25).

9.4 At p = 0.20, triple-failure = 3(0.04)(0.8) + 0.008 = 0.104 — one triple in ten lies. BER OUT stays under BER RAW but the gap narrows; the votes started lying wherever storms put two strikes in one triple (look for lost rows).

10.1 Codeword 0110011: C1 watches positions 1,3,5,7; C2 watches 2,3,6,7; C3 watches 4,5,6,7. Recompute each from the received bits; a failed check contributes its weight (1, 2, 4) to the address.

10.2 Positions 1, 2, 4 are the powers of two — each has exactly one binary digit set, so each check bit is watched by *only its own check*. Every data position (3, 5, 6, 7) has ≥ 2 digits set and is watched by that combination of checks; a flip at address j fails exactly the checks in j's binary expansion, so the verdicts spell j.

10.3 One more parity bit over all seven (the extended (8,4) code): double flips then show "syndrome nonzero but overall parity even" — detectably wrong, uncorrectable, flagged instead of mis-healed.

10.4 Distinct, nonzero column patterns: in Hamming's wiring every position is watched by a *unique* nonempty subset of checks (the 7 nonzero 3-bit patterns). Any wiring that repeats a pattern gives two positions the same syndrome — the panel's ambiguity flag (BUILD_NOTES

24).

11.1 Binomial mean 51.2, σ ≈ 7 — five runs typically land 40–65. If all five sat on one side of 51, RE-DEAL more: the dealer owes you nothing per run, only on average.

11.2 Thresholds cluster near p ≈ 0.10–0.15 for line drawings (the Sigma dies earlier than the Smiley — thin strokes have no redundancy of their own). Gradual because blocks fail independently: there is no p at which "the image" fails, only a rising fraction of its blocks.

11.3 At p = ½ the frame is a fresh coin flip per pixel — the checker is gone entirely, replaced by uniform snow. Worse than p = 0.9, where flipping every received bit back recovers 90% of pixels: a reliable liar carries information, a fair coin carries none.

12.1 At p = 0.02 naked loses ≈ 20 pixels; Hamming heals all single-flip blocks and typically keeps 0–4 — well under a third. The OVERHEAD meter reads ×1.75: that is the whole invoice.

12.2 $3(0.01)(0.9) + 0.001 = 0.027 + 0.001 = 0.028$. Per pixel after the vote: ≈ 2.8%, against 10% naked — the bench's WRONG PIXELS meter at p = 0.10 with REPEAT×3 sits near $0.028 \times 1024 \approx 29$.

12.3 Parity's one XOR sees only whether an odd number of bits flipped: one bit of evidence, 1-in-5 positions to blame — pointing needs $\log_2 5 \approx 2.3$ bits of evidence, and Hamming duly spends three checks. Blue blocks are the honest middle state: known-damaged, unfixable, kept.

13.1 Typical seed-77 reading: NONE ≈ 51 flips → ≈ 51 wrong pixels; REPEAT×5 ≈ 256 flips → single digits wrong. The wire taxed the armored lane five times harder and lost anyway — the premium bought the vote.

13.2 The crossover band sits near p ≈ 0.08–0.12: below it Hamming matches the vote's residual at half the overhead; above it triples and sevens both crumble, and only ×5 is left standing (at five times the price). The exact edge moves per seed — the design owns the band, the dice own the edge.

13.3 A data bit covered by a single check is healable only when that check's OTHER covered bits stay clean; audit() drops its healable count below 7, and the duel shows the residual gap growing exactly at the noise levels where that bit's block gets struck. The wiring predicts the verdict.

14.1 The wire converts 0.469 bits per bit into *uncertainty about what was sent*: to undo the noise you would need to be told, per bit, which flips happened — a correction message of $H_2(0.10) = 0.469$ bits per wire bit. The eaten share is exactly the size of the apology the channel would owe you.

14.2 At p = ½ the output is a fair coin independent of the input — the receiver knows *nothing* it didn't know before. Past ½ capacity climbs back: a wire that flips 90% of bits is a good wire mislabeled (invert everything; capacity(1) = 1).

14.3 A 25/75 input only *has* $H_2(0.25) = 0.811$ bits per symbol to send — you cannot deliver information you did not put in, and the noise still takes its cut. The engine pins I(0.10, 0.25) = 0.412 < 0.531 = I(0.10, 0.5), and the tests sweep skews to confirm ½ is the peak.

15.1 Between 0.17 and 0.18 (pinned: C(0.17) > ⅓ > C(0.18)). Operationally: past that noise, even sending everything three times leaves *more* information destroyed than the code's redundancy can ever buy back — reliability is impossible at that rate, not merely hard.

15.2 Above at 0.10: NONE (1.0), PARITY (0.8), HAMMING (0.571). Below: REPEAT×3 (0.333). The theorem promises that *some* code at rate ≤ 0.531 achieves any target reliability — it does not promise repeat×3 is that code (it isn't: its BER OUT at 0.10 is merely better, not arbitrarily good; only longer, cleverer codes cash the promise).

15.3 A random codebook has no structure to decode by — the receiver must compare against all 2^(Rn) codewords. Usable codes buy a polynomial-time decoder with algebraic structure, and for 45 years that structure also kept them measurably below C.

16.1 The first 64 characters of the book are a sample *of the model's own training text* — q matches p more closely, so the bill drops toward the needle (Gibbs

equality is reached only at exact match, which a 64-symbol slice still isn't).

16.2 $H(p,q) \geq H(p)$, equality iff $q = p$. Any symbol with $q(x) = 0$ that actually occurs makes $-\log_2 q(x) = \infty$, and a frequency model assigns exactly 0 to everything unseen. Real models smooth: Laplace's +1, or a neural softmax that never emits exact zeros.

16.3 Gibbs' equality condition is $q = p$ — only the source's own distribution avoids all mispricing. The gap is $D(p\|q)$, KL divergence: the number of extra bits per symbol you pay for believing q in a world that runs on p.

16.4 Needle 3.50 (the message's own empirical distribution — knows it best by definition) < REF 4.14 (English at large: right family, wrong member) < MODEL 4.21 (the book: a *particular* other text) < CONTEXT (thin — 22 pairs of pretend-knowledge; the flag outranks the printed digits). Trust instruments in the order they trust themselves.