handnet: Neural Networks with Hand-Written Weights

Birger Moëll*

November 2026

Abstract

We study neural networks whose weights are *designed* rather than learned. The architecture is standard (embeddings, convolutions, ReLU recurrences, multilayer perceptrons), the tensors are allocated at a fixed parameter budget, and every non-zero weight is placed by reasoning about the task; the vast majority of the budget stays zero. We describe a small toolbox of constructions (pattern kernels, first-of-run counting, run-length counters as a ReLU recurrence with a closed form, capped detectors from ReLU pairs, min gates, count-then-threshold) that lets ordinary task knowledge be written directly as weights, and we report experiments on seven networks across six tasks. Three are transformers whose attention heads are written by hand—multi-digit addition with carry, Dyck-1 bracket matching and an induction head—and are exact with no labels, where trained twins of the same architecture need on the order of a thousand examples. A multilingual text-quality network of 100,331 parameters (627 non-zero) separates kept from removed FineWeb-2 documents with a mean AUC of 0.907 over 33 languages and eleven scripts (0.82–0.98) without having seen a document, beating single-statistic baselines in all but one language. An SST-2 sentiment network written blind reaches 78.6 %, and two MNIST designs reach 79.4 % (weights that are ASCII drawings, written blind with bounded revisions) and 83.3 % (hand-drawn stroke kernels, a recurrent hole finder, 5×5 sketches and named evidence units; above a trained nearest-class-mean classifier at 81.6 %). Label-matched trained baselines show where designed weights win and where a few hundred labels beat the best drawing. We discuss where the approach stops (knowledge that cannot be articulated in the input features, such as morphology or whether a loop closes), and how designed networks relate to trained ones as initialisations, baselines and specifications. Code, results and an in-browser demo are at https://birgermoell.github.io/handnet/.

1 Introduction

A trained network of 100,000 parameters uses all of them a little. The question this paper asks is what happens when a network of the same shape uses a few hundred of them a lot, and the rest not at all—when the weights are *written* rather than learned.

The question is not academic. Much of what a small classifier needs to know is knowledge people already have in words: a document that is 40 % angle brackets is markup; *not* flips the next few words; a zero has a hole and a one does not. Classical pipelines put such knowledge into hand-designed features and learned the classifier on top. Here the classifier is designed too: the whole model is an ordinary neural network—embeddings, convolutions, a ReLU recurrence, a multilayer perceptron—whose tensors are allocated at a fixed budget, initialised to zero, and then filled in one justified weight at a time. The result loads into PyTorch unchanged, can be fine-tuned, and can be compared with a trained network of identical architecture on a per-label basis.

*The weights and code were written by Claude Fable 5.1 (Anthropic) in conversation with the author, under the protocol of Section 3.6; the author set the tasks, the budgets and the rules, and audited every unit.

The immediate prompt was a report that a language model, asked to hand-write the weights of a 100k-parameter classifier for real versus fabricated legal citations, used 3,793 non-zero weights and outscored the same architecture trained on 8,192 labelled examples (97.8 % against 94.6 %) [2]. One result on one convenient task does not say how far this carries, which tasks it carries to, or how to do it deliberately. We set out to answer those three questions.

Contributions.

- **A method** (Section 3.3 and surroundings): six constructions—pattern kernels, first-of-run counting, run-length counters as a ReLU recurrence with a closed form, capped detectors from ReLU pairs, min gates, count-then-threshold—that let ordinary task knowledge be written directly as weights, together with a budgeting and naming discipline that makes the network's explanation exact rather than post hoc.
- **A protocol** (Section 3.6) that turns "it works" into "it is worth N labels": blind writing, a sealed test set, bounded revisions each reported with the labels it inspected, and trained baselines of the same architecture at matched label counts.
- **Seven networks on six tasks**, released with tests, frozen revisions and results. Three transformers with hand-written attention—addition with carry, Dyck-1, induction—exact with zero labels, building forward the circuits mechanistic interpretability reverse-engineers (Section 4.1). A multilingual text-quality gate of 100,331 parameters, 627 non-zero, that separates kept from removed FineWeb-2 documents at mean AUC 0.907 across 33 languages and eleven scripts without having seen a document (Sections 4.2–4.5); an SST-2 sentiment classifier of 1,567 non-zero weights at 78.6 % written blind (Section 4.6); and two MNIST designs, one whose weights are 36 ASCII drawings (79.4 %, Section 4.7) and one built from stroke kernels, a recurrent hole finder and 5×5 sketches (83.3 %, above a trained nearest-class-mean classifier, Section 4.8).
- **A map of where the approach stops.** Designed weights win where the discriminative features are nameable (words, markup, mojibake, holes) and lose, past a few hundred labels, where they are statistical regularities nobody can enumerate (fine stroke differences). The text network is at chance on single-word grammar errors by construction; the boundary is a result in itself (Section 5).

Everything is at `https://github.com/BirgerMoell/handnet`: the networks in numpy with no other dependency, the evaluation scripts, the frozen earlier versions behind every revision row, and a browser demo of the text-quality network.

2 Related work

Hand-designed features with a learned classifier are the pre-deep-learning norm; here the classifier is hand-designed too. The closest precedents are the rule-based quality filters of large pretraining corpora—C4 [9], Gopher [8], FineWeb and FineWeb-2 [6, 7]—whose rules (fraction of letters, mean word length, lines ending in punctuation, repetition ratios) our text network reimplements as network units with a shared scoring scheme and script gating. The starting observation for this work is a report that a language model asked to hand-write the weights of a 100k-parameter citation classifier used 3.8k non-zero weights and beat the trained version [2]. Network "circuits"

with interpretable weights are the object of mechanistic interpretability [5]; we build circuits forward instead of reverse-engineering them. Hand-set recurrent counters resemble the constructions used to show what RNNs can compute in principle [11], and our hole finder is a cellular-automaton flood fill written as a ReLU recurrence. MNIST [4] needs no introduction; template-matching and nearest-prototype classifiers are its oldest baselines, and our sketch units are nearest-prototype with prototypes drawn by hand. Sparse networks are usually obtained by pruning a trained one [1]; ours are sparse by construction, with the sparsity pattern carrying the design.

3 Method

3.1 Setting

A hand-written network is an ordinary feed-forward or recurrent network f_θ whose parameter tensors are allocated at a fixed budget and initialised to zero, after which individual entries are set by the author. We require that (i) every layer is a standard operation (affine map, ReLU, convolution, ReLU recurrence, mean/max pooling), so that θ loads into any deep-learning framework unchanged, and (ii) every non-zero entry carries a comment stating why it has its value. We report two numbers for each network: the allocated parameter count $|\theta|$ and the number of non-zero entries $\|\theta\|_0$.

3.2 Input representation

The weights are only as readable as the features they read. We therefore use inputs whose meaning is fixed a priori and make the first embedding the identity. For text, each character is mapped to a binary attribute vector (Unicode general category, case, script block, punctuation class, combining mark, a same-as-previous-character bit, and a few task-specific markers such as the lead and tail bytes of UTF-8 mojibake). No vocabulary is involved, so one set of weights applies to every language and script.

3.3 Constructions

Six constructions account for every non-zero weight in the networks reported here.

Pattern kernels. A convolution channel with weight 1 on each required (offset, attribute) pair and bias $-(n-1)$ fires under ReLU only when all n attributes are present: $\mathrm{relu}(\mathrm{terminal}_{t-2} + \mathrm{space}_{t-1} + \mathrm{upper}_t - 2)$ detects a sentence start in cased scripts. A large negative weight on an attribute implements exclusion.

First-of-run counting. $\mathrm{relu}(x_t - \mathrm{same_prev}_t)$ is 1 only at the first character of a run of x, so that a separator line of eighty dashes contributes one count rather than eighty.

Run-length counters. A ReLU recurrence $h_t = \mathrm{relu}(h_{t-1} + u_t)$ with unit self-weight and input $u_t = \mathrm{inc}_t - B \cdot \mathrm{reset}_t$ (B large) counts increments since the last reset: characters since the last space (word length), since the last sentence end (sentence length), words since the last sentence end. Because the self-weight is exactly 1, the recurrence has the closed form $h_t = S_t - \min(0, \min_{k \le t} S_k)$ with S the cumulative sum of u (the Lindley recursion), so inference is vectorised; the step-by-step recurrence is used for the PyTorch export and the two are tested to agree.

Capped detectors. A penalty that rises from 0 at threshold a to its full value at b and saturates is $\min(\mathrm{relu}(x), 1) = \mathrm{relu}(x) - \mathrm{relu}(x-1)$ with x an affine function of the pooled features scaled so that $x = 1$ is "fully bad". The next layer multiplies the pair by a point budget. Every penalty and reward term is one such unit, and the sum of the named terms is the output; this is what makes `explain()` exact rather than approximate.

Gates. A detector that is meaningless for some input mode (word length when the script has no spaces; capitalisation when the script has no case) receives a weight of -100 on the rate of that mode. A soft gate $\min(g, p)$, used to switch rate-based penalties on only once the document is long enough for rates to be meaningful, costs one extra layer: $p - \mathrm{relu}(p - g)$.

Count-then-threshold. "More than one script present" is one capped unit per script, summed with bias -1 and passed through ReLU; the output is the number of extra scripts, itself capped.

3.4 Budgets and naming

The output is a base value plus named reward terms minus named penalty terms, grouped into sub-scores and clipped to the target range by a final $\mathrm{relu}(z) - \mathrm{relu}(z - 100)$. Budgets are chosen in output units: in the text-quality network a markup rate above 10% can cost at most 20 of 100 points, mojibake at most 35, and the rewards for the shape of running prose sum to 28, so that clean prose reaches 100 and a table reaches 0.

3.5 Development protocol

We develop against three sources in order: (1) hand-written samples across scripts and junk types, each a test case with a stated reason; (2) a labelled external set the network has never seen; (3) the worst mistakes in both directions on (2), fixed at the level of the mechanism rather than the sample. Every fix is recorded as a comment in the code and a line in Section 4.3. The network never sees a training signal; the author does.

3.6 Protocol for comparing designed and trained weights

The sentiment and MNIST experiments follow a protocol that turns "the designed network works" into "the designed network is worth N labels": (1) freeze a differentiable architecture, building min and clipping from ReLUs so that a trained model of the same shape is a meaningful baseline; (2) name every unit; (3) write all non-zero weights *blind*, before any labelled example from the benchmark is seen (general domain knowledge is allowed, the data is not); (4) evaluate once on the sealed test set and print only aggregates; (5) allow a bounded number of *revisions*, each of which inspects a development sample, states a hypothesis about language or shape, changes weights and is reported as its own row with the number of labelled examples it saw, the earlier file being kept so the row can be regenerated; (6) train the same architecture on 2^k labels for a range of k and plot. The author—here a language model, Claude Fable 5.1, in a chat session—may not call any optimiser, may not grid-search over labelled data, and may not import pre-trained resources. Fixed operations are allowed when parameter-free or hand-set (a Gaussian blur, ℓ_2 normalisation, max-pooling over offsets).

4 Experiments

4.1 Transformers with hand-written attention

The networks so far are convolutional, recurrent or feed-forward. A reader may grant that heuristics can be packed into those and still doubt that *attention*—the operation that defines current models—can be written by hand. This section writes three transformers whose correctness is exact, on tasks chosen because mechanistic interpretability has reverse-engineered the same circuits out of trained models [5]. We build them forward.

Architecture. A decoder-only transformer in numpy: token and position embeddings, L layers of causal multi-head attention and a ReLU MLP with residual connections, and an unembedding. There is no LayerNorm: every dimension of the residual stream is given a name by the author (`s`, `nine`, `carry`, `prev[7]`, ...) and a fixed scale, which normalisation would destroy. All tensors are allocated at full shape and zero; `to_torch()` yields the identical module, and with random initialisation, its trained twin.

Addition with carry. The input is the two addends as digit pairs, least-significant first, after a BOS token and with one extra 0+0 column for the final carry; the output at column i is digit i of the sum. The embedding of a pair (a, b) is the column sum $s = a + b$ together with the flags $[s = 9]$ and $[s \geq 10]$. The only hard part of addition is the carry chain: the carry into column i is decided by the nearest earlier column whose sum is not nine, since nines pass a carry through unchanged. One attention head finds that column: $\text{score}(i, k) = 200\,(1 - \text{nine}_k) + 5\,\text{pos}_k$ under a strict causal mask, so "not nine" dominates, recency breaks ties, and softmax gaps of at least 5 make the pattern close to one-hot (0.993 on $999999 + 1$, Figure 1). Its value is that column's $[s \geq 10]$, written into a `carry` dimension; BOS acts as a decider with carry 0. The MLP then computes $(s + \text{carry}) \bmod 10$ with 21 units $\text{relu}(s + \text{carry} - j)$, $j = -1, \ldots, 19$, whose differences $\text{relu}(x - k + 1) - 2\,\text{relu}(x - k) + \text{relu}(x - k - 1)$ are triangular bumps equal to 1 at $x = k$ and 0 at every other integer; bump k adds one to output digit $k \bmod 10$. One layer, one head, 2,851 parameters, 405 non-zero; exact on 2,000 random problems of up to twelve digits.

Dyck-1. Is a bracket string balanced? Opening and closing brackets embed as ± 1. A layer-1 head with all scores zero attends uniformly over positions $0..i$ and so outputs $\text{depth}_i/(i + 1)$; its MLP produces $\text{neg}_i = \min(\text{relu}(-C\,\text{depth}_i), 1)$ with C above the maximum length, so any negative prefix registers as exactly 1, and likewise a non-zero flag. A layer-2 head with $\text{score}(i, k) = 30\,\text{neg}_k$ concentrates on any flagged position and reads 1, or stays uniform over unflagged positions and reads 0: "was any prefix negative?" The string is invalid if that or the final non-zero flag fires. Two layers, 654 parameters, 26 non-zero; exact on 2,000 strings of up to 48 brackets, half balanced and half near-misses.

Induction. Given $x_1..x_L\,x_1..x_k$, predict x_{k+1}. A layer-1 "previous token" head (score $= 8\,\text{pos}_k$, strict causal mask, so $i - 1$ wins) copies the previous token's one-hot into a `prev` block. A layer-2 "induction" head uses the current token as query and `prev` as key (score 20 on a match, plus $0.5\,\text{pos}_k$ so the most recent match wins when a letter repeats) and returns that position's token: what followed last time. Two layers, 25,336 parameters, 278 non-zero; exact on patterns of distinct letters.

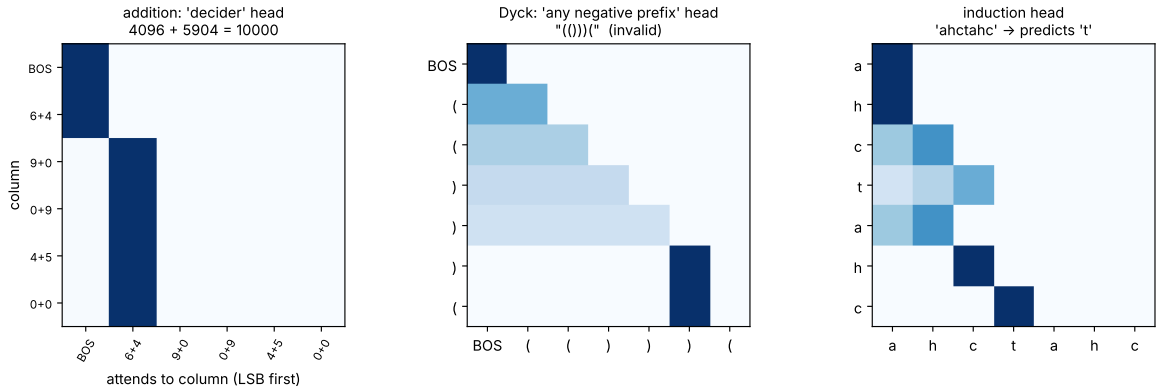


Figure 1: Attention patterns of the hand-written heads. Left: in $4096 + 5904$ every column after 6+4 attends to it, because the intervening columns all sum to nine and the carry chain runs through them. Middle: the Dyck head locks onto the bracket at which depth first went negative. Right: the induction head attends from each letter of the second copy to the successor of its first occurrence.

Trained twins. Table 1 trains the identical architecture from random initialisation on 2^k examples (Adam, 3,000 steps) and reports the same exact-match metric. The designed networks are exact with no labels. The trained twins are near chance at 64–256 examples, reach 97–99 % at 2^{10}, and at 2^{14} are still not exact (95–99.7 %; the addition twin's dip at 2^{14} is training variance under a fixed step budget). The gap between 99.7 % and 100 % is the gap between a statistical approximation of the carry rule and the rule itself.

labels seen	0 (designed)	2^6	2^8	2^{10}	2^{12}	2^{14}
addition	100.0	7.7	25.9	97.4	99.0	95.2
dyck	100.0	64.4	68.0	98.3	98.7	98.7
induction	100.0	5.6	37.2	98.6	99.6	99.7

Table 1: Exact-match accuracy (%) of the hand-written transformers (0 labels) and of the same architecture trained from random initialisation on 2^k examples, 3,000 Adam steps, 1,000 held-out problems. Addition: whole sum correct; Dyck: validity; induction: next token in the second copy.

4.2 A 100k-parameter multilingual text-quality network

Task. Score a document in $[0, 100]$ for usability as text in the pretraining-filter sense: clean running prose versus markup, tables, logs, mojibake, repetition, shouting, navigation lists, keyword lists, fragments and mixed scripts. The score must be defined for every language and script with one set of weights.

Architecture. Characters $\to$ 40 binary Unicode attributes $\to$ identity embedding (40×40) $\to$ width-5 convolution with 48 channels (18 pass-through rates, 17 pattern detectors, 13 script rates) $\to$ a 16-unit ReLU recurrence of run-length counters $\to$ pooling (channel means, counter maxima and means, log length, bias; 82 features) $\to$ an MLP $82 \to 512 \to 86 \to 16 \to 2 \to 1$. Layer 1 holds 70 capped detectors, 10 script-presence units and 2 length gates; layer 2 the named sub-scores, gates, reward and mixed-script count; layer 3 the $\min(\text{gate}, \text{penalty})$ terms; layer 4 the base-plus-terms score and its clip. In total $|\theta| = 100{,}331$ and $\|\theta\|_0 = 627$.

Script handling. Word-based detectors are gated off by the rate of Han, Kana and Thai-family characters; case-based detectors by Arabic, Indic, Hangul and Georgian (Unicode classes Mkhedruli as lowercase; it is written caseless). Sentence starts have a cased kernel (`.  A`) and a caseless kernel (an ideographic full stop U+3002 directly followed by a Han character, or a full stop, space and caseless letter as in Korean). Combining marks count as letters so that Indic and Arabic vowel signs do not depress the letter rate. Han and Kana count as one script. A CJK character is weighted as roughly 2.5 Latin characters in the length terms.

Evaluation data. FineWeb-2 publishes, per language, the documents that survived its filtering pipeline and the documents its filters removed. We take the first 300 of each from the first shard for Swedish, German, Finnish, Turkish, Georgian, Arabic, Hindi and Japanese (Latin, Georgian, Arabic, Devanagari and Japanese scripts) and report AUC for kept > removed. These eight languages were used for the error analysis in Section 4.3, so their numbers are development numbers. The 25 further languages of Section 4.5 were scored once, after the weights were frozen, and serve as the held-out check.

	swe	deu	fin	tur	kat	arb	hin	jpn
AUC	0.928	0.901	0.941	0.901	0.949	0.883	0.857	0.896
mean kept	95.2	95.6	97.4	94.5	95.2	88.6	91.9	90.8
mean removed	68.8	74.1	64.8	70.5	58.4	64.4	71.3	61.3

Table 2: FineWeb-2 kept vs removed, 300 + 300 documents per language. Scoring takes about 7 ms per document in numpy on one CPU core.

Hand-written samples. Encyclopaedic paragraphs in fourteen languages (Swedish, English, German, Georgian, Finnish, Japanese, Chinese, Arabic, Hindi, Russian, Greek, Thai, Korean, Turkish) score 85–97. HTML scores 23, UTF-8 mojibake 48, character repetition 35, shouting 37, a markdown table 0, a navigation list 36, a server log 9, a comma-separated keyword list 57, a list of URLs 47, U+FFFD soup 32, source code 17, a paragraph mixing five scripts 47, a 110-character string without spaces 41 and emoji spam 24.

Where it fails, by construction. On synthetic grammar pairs from the OELLM quality-gate data (a clean one-to-three-sentence snippet against the same snippet with one inflection change, adjacent-word swap, typo, duplicated word or deleted function word; Swedish, Norwegian, Georgian), the network is at chance, and most pairs are exact ties: the attribute representation carries no morphology or word order. Deleted function words are detected slightly above chance (62–86%) through length alone. The highest-scoring *removed* FineWeb-2 documents are fluent machine-translated hostel listings and site boilerplate, which have the surface shape of prose. Both failure modes are exactly the signal a trained judge provides, and the two are complementary: the designed network is ∼1000× cheaper than a 0.3B judge and can run first.

4.3 Development log

Each entry is a mistake found on external data and the mechanism that fixed it.

1. Hindi prose scored 61: Devanagari vowel signs are combining marks, so the letter rate was 45%. Fix: combining marks count as letters in the letter-rate terms (zalgo detection stays separate and is gated off for Indic, Arabic and Thai).

2. Japanese prose lost 12 points as "mixed scripts": Han and Kana were two scripts. Fix: one presence unit reads both rates.

3. Korean and Georgian earned no sentence reward: Korean puts a space after the full stop before a caseless letter; Georgian is Unicode-lowercase. Fix: a second sentence kernel, and Mkhedruli mapped to the caseless attribute.

4. An 80-character separator line cost 56 points through four detectors (repetition, punctuation runs, symbol streak, giant token). Fix: first-of-run counting for markup, symbol and dash rates; repetition and punctuation-run kernels exclude separator characters; the word-length counter increments only on letters and digits; the symbol-streak threshold raised.

5. One all-caps headline cost the full 15 shouting points. Fix: the caps-run rate needs about 15% of the text before the penalty saturates.

6. Short texts were hit by every rate-based penalty at once (a 17-character sentence-splitter fragment scored 3). Fix: two length gates as $\min(\text{gate}, \text{penalty})$; composition penalties switch on between 30 and 100 characters, structure penalties between 100 and 300.

7. The replacement character U+FFFD was classed as a symbol, not garbage. Fix: attribute order.

8. Chinese and Japanese lost 4–7 points for "comma-heavy" text (ideographic commas are dense). Fix: the detector is gated off for spaceless scripts.

No entry changed the FineWeb-2 AUC by more than ± 0.01; the fixes removed false positives on individual documents (e.g. a Georgian medical page from 66 to 96) without moving the aggregate.

4.4 Ablation by sub-score

Switching off one named sub-score at a time (zeroing its layer-2 unit; `examples/text_quality/ablate.py`) shows that on FineWeb-2 the *formatting* group—bullet-like lines, lines that break mid-sentence, spacing errors, ellipses, symbol streaks—carries most of the separation, which matches what FineWeb-2's own filters (Gopher repetition and quality rules, C4 line rules) remove. The remaining groups each contribute at most ~ 0.01 AUC on this data but are what separates the hand-written junk samples (HTML, mojibake, tables, logs) that FineWeb-2's sample rarely contains.

removed group	swe	deu	fin	tur	kat	arb	hin	jpn
none (full)	0.928	0.901	0.941	0.901	0.949	0.883	0.857	0.896
noise	0.925	0.903	0.943	0.897	0.948	0.880	0.856	0.896
repetition	0.928	0.901	0.941	0.899	0.946	0.873	0.849	0.893
composition	0.925	0.899	0.942	0.900	0.948	0.883	0.857	0.894
formatting	0.847	0.839	0.905	0.870	0.923	0.809	0.824	0.844
tokens	0.924	0.902	0.944	0.901	0.949	0.889	0.857	0.896
structure	0.928	0.901	0.940	0.899	0.946	0.882	0.851	0.896
length	0.928	0.900	0.940	0.898	0.947	0.879	0.855	0.891

Table 3: Leave-one-group-out ablation, AUC kept vs removed.

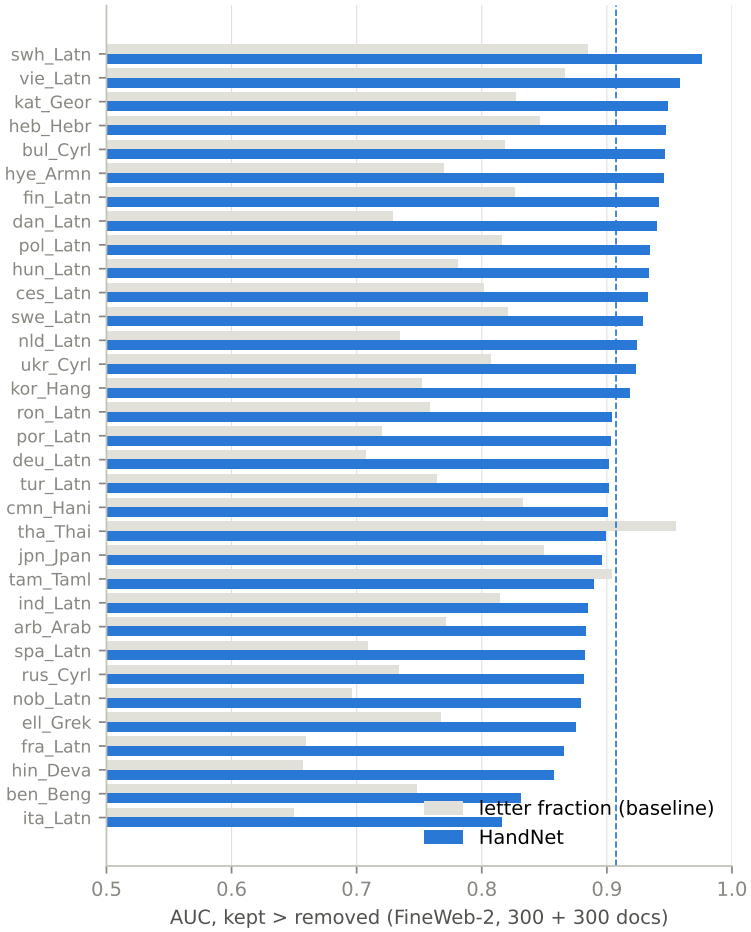


Figure 2: AUC(kept > removed) per language, the text-quality network versus the letter-fraction baseline. Dashed line: mean over languages (0.907).

4.5 Text quality across 33 languages, against single-statistic baselines

Section 4.2 evaluated eight languages. Here the same protocol (first 300 kept and 300 removed documents of the first FineWeb-2 shard, AUC for kept > removed) is run on 33 languages across eleven scripts (`examples/text_quality/eval_fineweb2.py`; Persian is not published in this layout and was skipped). As baselines, the same documents are scored with single statistics of the kind used by rule-based filters [8, 9]: log length, the fraction of characters that are letters, the fraction of tokens containing a letter, and the Gopher-style distance of the mean word length from 5.

Figure 2 and Table 4 give the results. The mean AUC is 0.907, with every language above 0.81; Swahili (0.976), Vietnamese (0.958), Georgian (0.949), Hebrew (0.947), Bulgarian (0.946) and Armenian (0.945) are highest, Italian (0.816), Bengali (0.831) and Hindi (0.857) lowest. Scores of kept documents are tightly concentrated near 95 while removed documents spread over the whole range (Figure 3); with a threshold of 50 essentially no kept document is rejected in any language, and between 5 % (Italian) and 59 % (Korean) of removed documents are.

Two observations. First, the letter fraction alone is a strong baseline (mean 0.78), and for Thai it beats the network (0.955 vs 0.899): Thai has no sentence punctuation and no word spaces, so the script gates switch off most of the structure detectors and little is left. Second, FineWeb-2's

Table 4: FineWeb-2 kept vs removed, 300 + 300 documents per language. AUC for the text-quality network and for three single-statistic baselines; mean network score of kept and removed documents.

language	network	length	letters	words	kept mean	removed mean
swh_Latn	0.976	0.736	0.885	0.902	92.2	44.8
vie_Latn	0.958	0.698	0.866	0.915	92.9	63.7
kat_Geor	0.949	0.697	0.827	0.858	95.2	58.4
heb_Hebr	0.947	0.643	0.846	0.886	93.0	58.6
bul_Cyrl	0.946	0.668	0.819	0.873	96.0	66.9
hye_Armn	0.945	0.534	0.769	0.763	93.5	57.0
fin_Latn	0.941	0.696	0.826	0.861	97.4	64.8
dan_Latn	0.940	0.657	0.729	0.803	95.3	66.5
pol_Latn	0.934	0.643	0.815	0.818	95.4	64.5
hun_Latn	0.933	0.705	0.780	0.854	96.3	64.4
ces_Latn	0.932	0.675	0.801	0.852	95.1	62.3
swe_Latn	0.928	0.671	0.821	0.882	95.2	68.8
nld_Latn	0.924	0.638	0.734	0.846	94.8	68.8
ukr_Cyrl	0.923	0.668	0.807	0.853	95.7	64.7
kor_Hang	0.918	0.492	0.752	0.850	80.4	41.5
ron_Latn	0.904	0.631	0.759	0.836	93.6	68.7
por_Latn	0.903	0.637	0.720	0.770	94.0	67.0
deu_Latn	0.901	0.628	0.707	0.849	95.6	74.1
tur_Latn	0.901	0.604	0.764	0.780	94.5	70.5
cmn_Hani	0.900	0.559	0.832	0.709	91.1	62.2
tha_Thai	0.899	0.571	0.955	0.862	85.6	64.3
jpn_Jpan	0.896	0.533	0.850	0.749	90.8	61.3
tam_Taml	0.889	0.682	0.904	0.881	96.0	73.6
ind_Latn	0.885	0.631	0.815	0.796	95.3	75.3
arb_Arab	0.883	0.612	0.771	0.832	88.6	64.4
spa_Latn	0.882	0.651	0.708	0.787	94.2	74.3
rus_Cyrl	0.881	0.607	0.733	0.805	95.9	75.8
nob_Latn	0.879	0.593	0.696	0.783	94.8	71.7
ell_Grek	0.875	0.646	0.767	0.854	94.1	69.6
fra_Latn	0.865	0.659	0.659	0.805	94.1	74.2
hin_Deva	0.857	0.604	0.656	0.733	91.9	71.3
ben_Beng	0.831	0.599	0.748	0.775	93.8	74.6
ita_Latn	0.816	0.650	0.650	0.722	93.1	76.4
mean (33)	**0.907**	0.634	0.781	0.823		

own removal decisions are rule-based, so part of what is measured is agreement between two rule systems rather than ground-truth quality; the comparison still tells us that a 627-weight network reproduces most of a production filtering pipeline's decisions in 33 languages.

In every language the most frequent penalties on removed documents are `broken_lines` (line breaks mid-sentence), `bullet_lines` and `fragments`, i.e. navigation lists and layout text, followed by `few_sentence_ends` and `markup`.

Ablation over 33 languages. Table 5 repeats the leave-one-group-out ablation of Section 4.2 on all 33 languages, now including the mixed-script penalty and the rewards (`examples/text_quality/ablate_groups`). Only the *formatting* group matters much: without it the mean AUC drops from 0.907 to 0.858 and Thai from 0.90 to 0.69. The other groups are nearly redundant on this benchmark, and removing the *rewards* or the *tokens* group slightly *improves* the AUC. This is a useful negative result for the design method: several detectors that are easy to justify (shouting, digit runs, giant tokens) contribute nothing measurable to agreement with FineWeb-2, and the reward units, which were

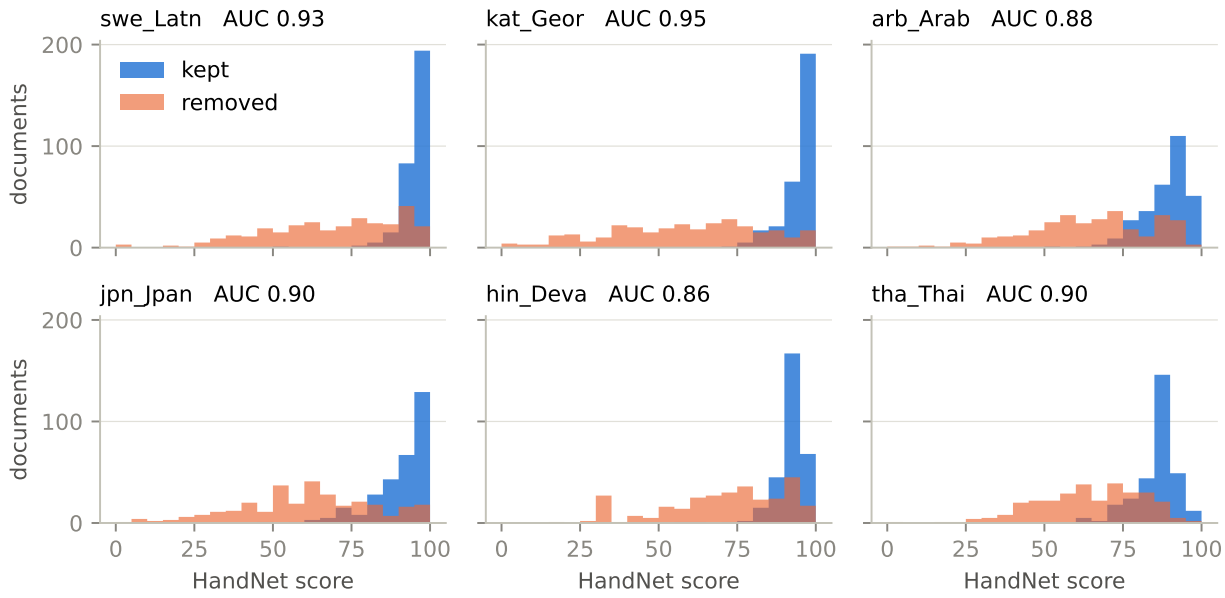


Figure 3: Score distributions of kept (blue) and removed (orange) documents for six scripts.

Table 5: Ablation over 33 languages: mean AUC with one group switched off.

variant	mean AUC	Δ	worst language
full	0.907	–	0.816 (ita_Latn)
without noise	0.905	-0.003	0.810 (ita_Latn)
without repetition	0.904	-0.003	0.809 (ita_Latn)
without composition	0.906	-0.001	0.816 (ita_Latn)
without formatting	0.858	-0.049	0.688 (tha_Thai)
without tokens	0.909	+0.001	0.815 (ita_Latn)
without structure	0.906	-0.002	0.820 (ita_Latn)
without length	0.905	-0.003	0.806 (ita_Latn)
without mixed_scripts	0.907	-0.001	0.817 (ita_Latn)
without reward	0.913	+0.006	0.830 (ita_Latn)

meant to make the scale interpretable, cost a little discrimination.

4.6 A 1.5k-parameter sentiment classifier for SST-2

Task. SST-2 is the binary Stanford Sentiment Treebank [10]: single movie-review sentences, lowercased and pre-tokenised (`does n't`), labelled positive or negative. We use the GLUE split (6,919 train / 872 dev / 1,821 test). It is hard for lexicon methods because of negation ("not a bad film at all"), contrast ("boring at first but ultimately moving") and intensified negatives used as praise ("painfully funny").

Architecture. Tokens → embedding $V \times 6$ → a gated recurrent cell with state [pos_acc, neg_acc, neg_gate, int_g → Linear($4 \to 6$), ReLU, Linear($6 \to 1$), sigmoid. The six embedding axes are *pos*, *neg* (strengths 1–3), *negator*, *intensifier*, *contrast* and *hedge*; the embedding is therefore a typed sentiment lexicon of about 1,100 words. The cell is ten scalars, each a statement about English: the negation gate

decays by 0.72 per token (reaching about three words) and is cancelled by a following hedge ("not *just* hilarious"); a contrast word multiplies the accumulated history by 0.35; a word under negation is routed to the opposite accumulator at 0.65 (pos→neg) or 0.75 (neg→pos) of its strength; intensifiers multiply by $1 + \text{int_gate}$, hedges by $1 - 0.5\,\text{hedge}$. All clipping is built from ReLUs. The readout's six units are net sentiment, net negativity, any positivity, any negativity, emotional mass and dangling negation, with output weights $[1.2, -1.2, 0.15, -0.35, -0.05, -0.4]$ and bias -0.15 (the prior, stated before evaluation, that non-evaluative review sentences lean negative). $\|\theta\|_0 = 1{,}463$ blind, 1,567 after revision.

Protocol. The weights were written blind, evaluated once on the sealed test set, then revised once after reading 45 development-set errors. Trained baselines (binary unigram+bigram counts; logistic regression and a 32-unit MLP, scikit-learn defaults) were fitted on random subsets of the training set.

Weights	Labels seen	$\|\theta\|_0$	Test acc.
Hand-written	0	1,463	**78.6%**
Hand-written, one revision	872 (dev)	1,567	**79.2%**
Trained logistic regression, uni+bigram	512	9,525	64.2%
Trained MLP (32 hidden)	512	304,832	65.3%
Trained logistic regression	2,048	30,884	74.0%
Trained MLP	2,048	988,320	75.2%
Trained logistic regression	6,919	81,032	80.0%
Trained MLP	6,919	2,593,056	81.9%

Table 6: SST-2 test accuracy (1,821 sentences). The blind network is worth about 3,000 labelled sentences (Figure 5).

The revision. Five changes, each a claim about language: (i) *never*, *nothing*, *hardly* had been listed both as negators and as negative words, so "never boring" counted negative twice; (ii) the negation decay was raised from 0.55 to 0.72; (iii) the neg→pos flip from 0.45 to 0.75 after "not a bad film at all" scored 0.22; (iv) the hedge-cancels-negation term was added; (v) about 100 lexicon entries were added (comparatives, horror-positive words such as *creepiest*, review negatives such as *familiar*, *tries*, and intensifier readings of *painfully*, *brutally*). Dev accuracy went 79.6 → 81.4%, test 78.6 → 79.2%; the gap between the two gains is the honest measure of dev-set overfitting.

Where it stops. The remaining errors are largely non-lexical: sentences with no evaluative word, topic words that look like sentiment ("a compelling anatomy of *grief*" is labelled negative), irony, and polarity carried by syntax ("what the director can't do is make either persona interesting"). A lexicon-and-gates network cannot represent the last two, which is why pretrained transformers reach 95%+ on this benchmark while the designed network stops near 80%.

4.7 MNIST with weights that are drawings

Task. MNIST digits, 10,000 test images. A trained linear classifier reaches ∼92%, a small MLP ∼97%, a CNN >99%.

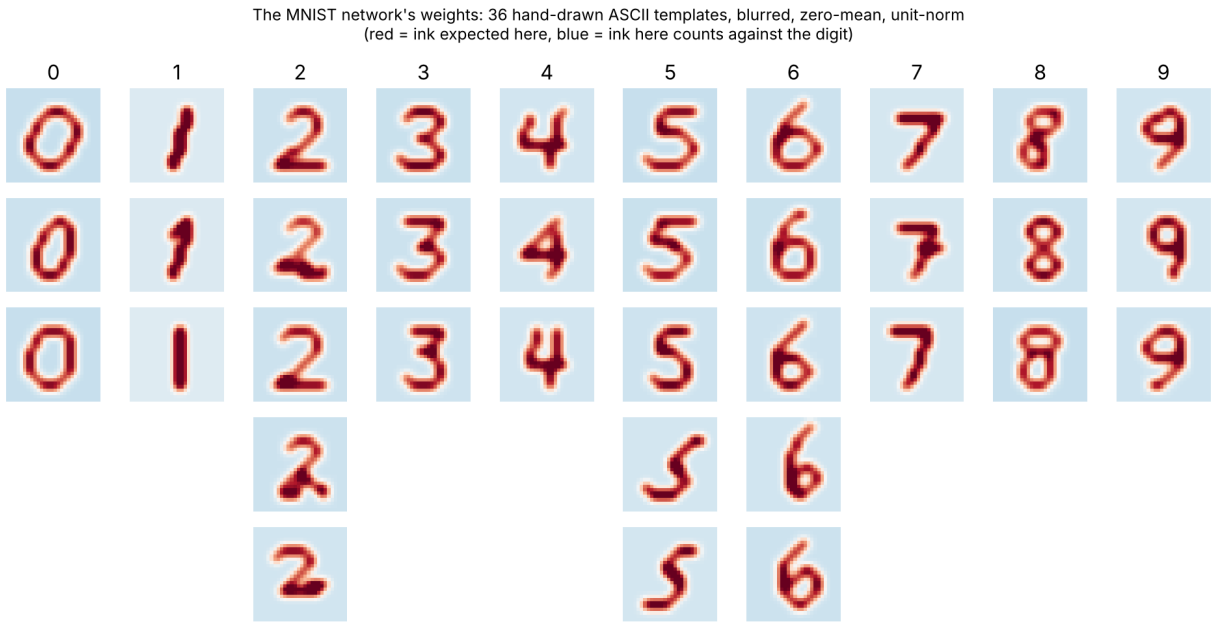


Figure 4: The MNIST network's weights: 36 hand-drawn templates after expansion to 28×28 kernels. Red: ink expected; blue: ink here is evidence against the digit.

Architecture. Image $\to$ fixed 5×5 Gaussian blur $\to$ zero-mean, unit-norm $\to$ Conv2d($1 \to 36$, 28×28, padding 3) $\to$ max-pool over the 7×7 offsets (± 3 px translation tolerance) $\to$ maxout $36 \to 10$ (best-matching style of each digit) $\to$ class bias $\to$ argmax. Each of the 36 kernels is a 14×14 ASCII drawing typed by the author (Figure 4), expanded by a parameter-free function: upsample $\times 2$, widen the stroke to ~ 3 px with a cross kernel, blur, subtract the mean (so ink *outside* the stroke counts against the match), normalise (so the activation is a cosine). Two to five styles were drawn per digit (a straight and a slanted 1, a 7 with and without a bar, an open and a closed 4, a 2 with a flat and with a looped base). We count the typed # characters as the non-zero parameters: 544 blind, 662 final.

Weights	Labels seen	Ink pixels / templates	Test acc.
Hand-drawn	0	544 / 28	**55.4%**
Hand-drawn, revision 1	500 (class means viewed)	553 / 30	**74.3%**
Hand-drawn, revision 2	500 (error images viewed)	662 / 36	**79.4%**
Trained logistic regression	50	–	61.4%
Trained logistic regression	100	–	71.5%
Trained MLP (64 hidden)	100	–	71.8%
Trained logistic regression	500	–	85.0%
Trained MLP (64 hidden)	500	–	86.2%
Trained logistic regression	10,000	–	90.9%
Trained MLP (64 hidden)	10,000	–	95.4%

Table 7: MNIST test accuracy. The best hand-drawn network is worth about 200 labelled images.

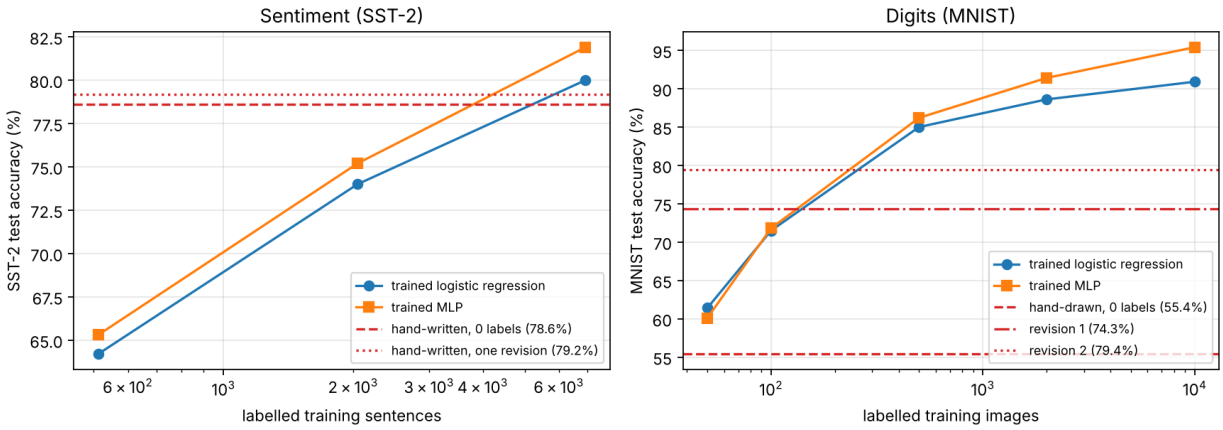


Figure 5: Designed versus trained, per label. Dashed lines: hand-written networks; curves: the same architectures trained on 2^k labels. Sentiment (left) and MNIST (right) sit at the two ends of the "nameable features" axis.

Revisions. The blind drawings were systematically wrong: one grid row too high, upright rather than right-leaning, too wide. Revision 1 redrew all ten digits after viewing the class-mean image of 500 labelled training digits next to the templates (+19 points). Revision 2 added two styles each for 2, 5 and 6 (the weakest classes) after viewing six misclassified training examples of each, widened the blur from 3×3 to 5×5 and the shift from ± 2 to ± 3 px (+5 points).

Why this one loses. The large confusions are $3 \to 5$ (197), $4 \to 9$ (158), $9 \to 4$ (129) and $5 \to 8$ (106): pairs that differ in the presence of one small stroke, which a whole-digit correlation barely registers against the shared bulk of the shape. A trained linear classifier puts large weight on exactly those few discriminative pixels; a person drawing a prototype cannot know which pixels those are without seeing thousands of examples. The features that matter on MNIST are not nameable, so the design step has nothing to work with. A part-based variant (hand-drawn stroke detectors feeding a hand-written 10×64 linear layer) was tried and abandoned: writing those 640 numbers was, on inspection, guessing.

4.8 MNIST, second design: a hole finder, sketches and evidence

Task and relation to Section 4.7. The same task, approached by a second thread independently and under a different protocol. Where the network of Section 4.7 was written blind and then revised twice, this one was designed with the training set in view from the start (see "How the numbers were set" below), so its number is not a blind-write result and is reported separately. Its design differs in three ways: a recurrent *hole finder* computes the topology that a feed-forward template cannot, the sketches are drawn per *stroke-orientation channel* rather than on raw ink, and a layer of named evidence units votes on the confusions that prototypes leave.

Design. The design question was "what does a person look at to tell digits apart?", and the network is that answer written as units (Figure 6).

Kernels. Five 5×5 kernels: a 3×3 box blur for *ink*, and oriented-bar detectors for horizontal, vertical, "/" and "\" strokes (+1 on the bar, +1.5 at the centre, −0.25 beside it, divided by 4, bias −0.15). A stroke lights its own orientation and, because of the negative surround, not the others.

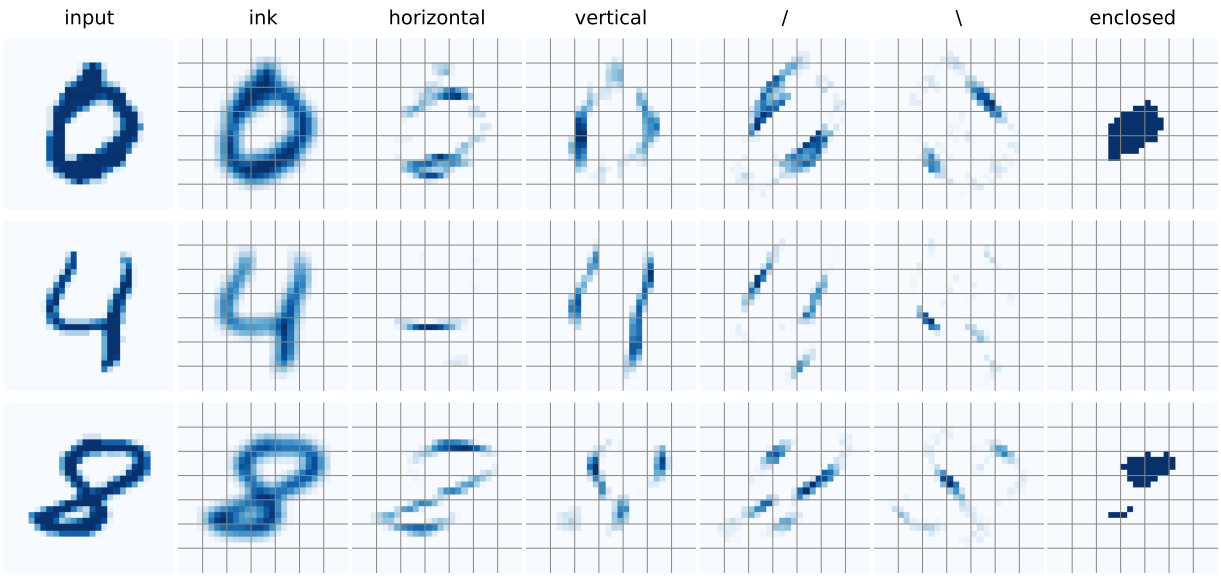


Figure 6: Three test digits through the network: the input, the five hand-drawn convolution channels, and the output of the recurrent hole finder. The grid is the 5×5 zone pooling. Note the 8 whose lower loop did not close: the single most common error.

Hole finder. Holes—the enclosed background of a 0, 6, 8 or 9—are the strongest single cue a person uses, and they are topological, which a feed-forward convolution cannot compute. We compute them with a ReLU recurrence over the image. Let b be the background (ink below 0.3). Seed r_0 with the border of b and iterate

$$r_{t+1} = \mathrm{relu}\big(\min(1, \textstyle\sum_{3\times3} r_t) + b - 1\big),$$

so a background pixel becomes *reached* as soon as it touches a reached pixel. After 56 steps everything reachable from the border has been reached and $b - r$ is the enclosed background. This is the same construction as the run-length counters of Section 4.2—a ReLU recurrence with identity self-connection—applied over an image instead of a string.

Zone pooling. All six maps (five conv channels and the hole map) are averaged over a 5×5 grid of 4×4-pixel zones covering the central 20×20 box in which MNIST digits sit: 150 zone features, plus seven globals (ink, hole area, hole area in the top and bottom halves, width, height, left–right ink balance).

Sketches. For each channel, each digit is drawn by hand on the 5×5 grid as ASCII art (Figure 7): `#` strong, `+` some, `.` little, `-` must be empty. The 60 sketches *are* the weights of 60 units: a sketch W gives the unit $W \cdot x - \frac{1}{2}\|W\|^2$, which up to a constant is minus the squared distance to the prototype, so the readout asks "which hand-drawn digit is this closest to", channel by channel (channel weights: ink 1, strokes 2, "\" 1, hole 4). The sketches were drawn after looking at what MNIST digits look like at this resolution—slanted 1s, 4s with a stem to the right, 7s whose roof sits on the second row—the way a person who has seen handwriting draws a digit, and are deliberately coarse.

Evidence units. Sketches alone reach about 80 %. The remaining confusions are topological and shape facts that distance to a prototype does not capture, so 29 named evidence units vote for and against digits, each a capped detector $\min(\mathrm{relu}(w \cdot f + b), 1)$ over a few zone features (the construction of Section 3.3): `hole:none` (votes for 1 3 4 5 7, against 0 8 9), `hole:big` (0), `hole:top_only` (9), `hole:bottom_only` (6), `hole:top_and_bottom` (8), `narrow` (1), `four:open_top` (two posts

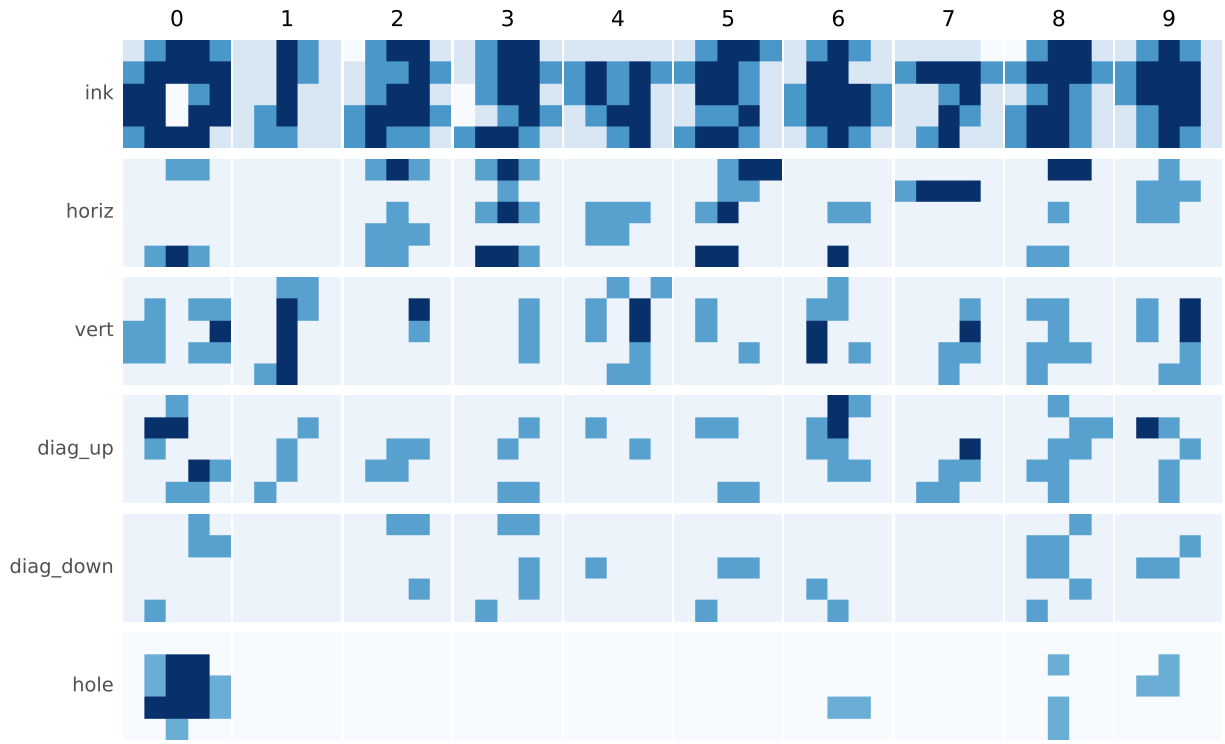


Figure 7: The sixty hand-drawn sketches, one per digit and channel. These are the prototype weights.

and an empty top centre), `seven:roof`, `two:foot`, `three:bumps`, `five:corner`, `six:left_sweep`, `nine:stem`, `zero:ring`, `eight:waist`, `bottom_bar` (0 2 3 5 6 8 have one; 1 4 7 9 do not), `roof_row1` (a 7, never a 4), `top_centre_filled` (against 4), `has_horizontals` and `has_backslashes` (against 1), and a few more. In total $|\theta| = 21{,}772$ and $\|\theta\|_0 = 2{,}411$.

How the numbers were set. The kernels, the hole finder, the sketches and the unit definitions were written from how digits look. Thresholds were placed by reading class-conditional means of the zone features on the first 5,000 *training* images and putting the knee between a digit and its neighbours ("a 4's top-centre ink is 0.6, a 9's is 1.8, so threshold at 1.0"). Vote weights were then adjusted by hand over about a dozen rounds of looking at the confusion matrix on those same 5,000 images and asking which unit was to blame for each confusion. No gradient, optimiser or search over weights was used. A held-out training slice (images 5,000–10,000) gives 82.8 %, so the hand tuning did not overfit; the test set was scored once, at the end.

Results. On the 10,000 MNIST test images the network is **83.3 % accurate** (Figure 8), at 0.9 ms per image in NumPy. The majority class gives 11.4 %. A *trained* nearest-class-mean classifier on the raw 784 pixels (ten prototypes fit on 20,000 training images) gives 81.6 %, so ten digits drawn by hand on a 5×5 grid, plus the hole finder and a page of evidence, do slightly better than ten prototypes fit to data at full resolution. A trained convolutional network reaches over 99 %. Per-class accuracy ranges from 95 % (1) and 91 % (2, 3) down to 77 % (5, 9) and 65 % (8).

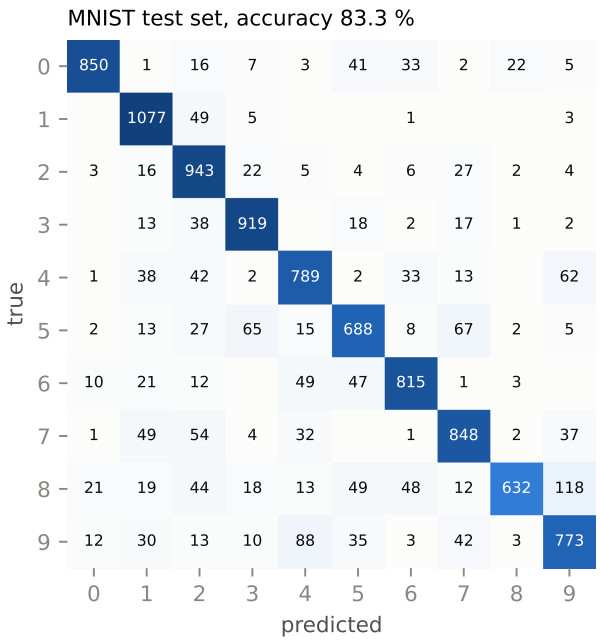


Figure 8: Confusion matrix on the 10,000 MNIST test images (rows normalised for colour).

Where it fails, by construction. The errors are the ones a 5×5 sketch would be expected to make, and the hole finder explains most of them: 8→9 (118 cases) when the lower loop did not close, 9→4 (88) when the upper loop did not close, 4→9 (62) when a 4 is written with a closed top; and the shape pairs 5→7 (67) and 5↔3 (65), where a 5 with a short roof is a 3 with a corner. The unit firing rates on the test set confirm the picture: `hole:none` fires on 99 % of 1s, 3s, 5s and 7s but also on 26 % of 9s and 37 % of 6s, which is where the accuracy goes. A 5×5 sketch cannot say "the loop must close", and the hole finder cannot close a loop the writer left open; a trained network learns the fine stroke-topology statistics that fill that gap, and that is exactly the part of the task a designer cannot write down.

5 Discussion and limits

5.1 When does writing beat training?

The three networks line up on one axis: how *nameable* the discriminative features are (Table 8). Where an expert could write the rules as a list —this word is positive, negation reaches three words, mojibake looks like `Ã¶`—the rules map one-to-one onto units and the designed network is competitive with thousands of labels. Where the features are statistical regularities in pixel space that nobody can enumerate, a few hundred labels beat the best drawings anyone can make. Lauritzen's legal-citation task [3] sits at the far nameable end (reporter names, volume and page formats), which is why designed weights beat 8,192 labels there.

5.2 The revision step is where the labels go

A revision after inspecting 500 labelled examples is not zero-label learning; it is hypothesis-driven learning with very few degrees of freedom. The right comparison for a revised designed network

	Sentiment	Text quality	Digits
Input unit	word	character attribute	pixel
Nameable features?	yes: lexicon, negation, contrast	yes: markup, repeats, mojibake, lengths	no
Designed vs trained	wins up to ∼3,000 labels	AUC 0.86–0.95, no labels	loses beyond ∼200 labels
Ceiling	syntax, irony	grammar	fine stroke differences

Table 8: Where hand-written weights are competitive.

is a trained one at the same label count, which is why every table lists labels seen. On MNIST the revised network (500 seen, 79.4%) loses to a trained one on 500 labels (85–86%); on SST-2 the revised network (872 seen, 79.2%) beats a trained one on 2,048.

5.3 A capability measurement

"Can model M write the weights of a working classifier for task T?" has a numeric answer; it differs across tasks in an interpretable way; and it will presumably rise across model generations. A model that writes a 79% MNIST classifier from an ASCII-art prior and a 79% SST-2 classifier from its own lexicon is telling us something about what it knows that a multiple-choice benchmark does not.

5.4 Things that went wrong

Dual-role words (*never* as negator and as negative) need an explicit decision. The blind MNIST drawings were one row too high; a single *unlabelled* look at the input distribution would have prevented this and a future protocol should allow it. "Painfully funny" was scored as *painful*+*funny*; intensifier readings of negative adverbs are real in review English. A document-length penalty appropriate for pretraining filtering dominates single sentences. And when a weight cannot be justified in a sentence—as in the abandoned part-based MNIST net—that is the signal to stop.

5.5 What reasoning buys, and where it runs out

What reasoning buys. The text-quality and hole-finder networks were written in a few hours of conversation and have no training cost, and both land at the level of the classical hand-engineered baselines for their tasks: the text gate reproduces most of a production filtering pipeline's decisions across 33 languages, and the second digit classifier beats a trained prototype classifier. Both are entirely legible: every score decomposes into named contributions, which is what a data engineer wants from a filter ("why was this document dropped?") and what makes the error analyses above possible without any probing.

Where reasoning runs out. The failures are informative. The text network cannot see grammar, because that needs a representation of words and their order, which no amount of character-level reasoning produces; and the 33-language ablation shows that several detectors that are easy to justify contribute nothing measurable. The digit network's ceiling is set by the resolution of what a person can specify: a 5×5 sketch cannot say "the loop must close", and the hole finder cannot close a loop the writer left open. A trained network learns the fine stroke-topology statistics that fill that gap, and that is exactly the part of the task a designer cannot write down.

Fine-tuning from a designed initialisation. Because these are ordinary networks with PyTorch exports (`to_torch()`, checked against NumPy in the tests), the obvious next experiment is to use them as initialisations: does a network that starts from named, correct detectors train faster, need fewer labels, or stay legible? The repository deliberately does not run that experiment yet, so that the reported numbers are those of the hand-written weights alone.

On process. The weights were written by a language model in conversation with the first author, who set the tasks, the rule (reason and check, never fit) and the standard of evidence. We think this is worth stating because it bears on what the result means: the design knowledge in these networks is the kind that can be articulated—what mojibake looks like, where a 4 is empty—and a system that can articulate it can also write it into weights. The repository documents the process, including the development log of Section 4.3 and the iteration on training-set confusion matrices for MNIST, so that the line between "reasoned" and "fit" is visible to the reader.

6 Conclusion

Seven ordinary networks, no optimiser, every non-zero weight decided and named: three transformers whose attention heads add, match brackets and copy exactly, a text-quality gate that agrees with FineWeb-2's filters at AUC 0.907 across 33 languages with 627 weights, a sentiment classifier worth a few thousand labels, and two digit classifiers built from drawings, one of which beats a trained nearest-class-mean baseline. Writing weights gets a network to the level of the classical hand-engineered baselines, makes each decision legible and testable, and stops where the task knowledge stops being articulable in the input features. That boundary moves with the author's knowledge, not with data; the protocol makes it measurable, and the networks are released as initialisations for anyone who wants to measure what training adds on top of design.

References

[1] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[2] Jacob Lauritzen. Got fable to hand-write weights of a 100k param model to recognize real versus fake legal citations, 2026. X post, @jacsebl.

[3] Jacob Lauritzen. Got fable to hand-write weights of a 100k param model to recognize real versus fake legal citations. X (Twitter) post, @jacsebl, November 2026.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020.

[6] Guilherme Penedo et al. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. arXiv:2406.17557.

[7] Guilherme Penedo et al. Fineweb2: One pipeline to scale them all, 2025. arXiv:2506.20920.

[8] Jack W. Rae et al. Scaling language models: Methods, analysis & insights from training gopher. 2021. arXiv:2112.11446.

[9] Colin Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

[10] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[11] Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision RNNs for language recognition. In *ACL*, 2018.