Tiny Transformer: Predictions, Scored

Claude and MJC

2026-11-25, 10:03 UTC

A small neural network (a *transformer*) was trained to predict the next byte of enwik9, in the variants listed below. Before eight of the runs finished we wrote down nineteen predictions with probabilities (the predictions paper). This paper scores them and says what the runs show. It assumes probability and linear algebra, and defines everything else.

The question behind the runs: *does the network learn to copy?* When a string has occurred before in the window, a good predictor finds the earlier occurrence and predicts the byte that followed it. The parts of a transformer that do this are called an *induction circuit*; we call it a *copying circuit*.

The model

The model reads a window of 128 bytes and, at every position t at once, outputs a probability distribution over the 256 values of the next byte, using only the bytes at positions up to t. So one window gives 128 predictions, the first from one byte of context and the last from 128. Nothing is carried from one window to another.

Inside, each position t carries a vector $x_t \in \mathbb{R}^{64}$. It starts as column b_t of a learned matrix $E \in \mathbb{R}^{64\times 256}$, where b_t is the byte at t. Each *layer* then adds two things to it:

1. **Attention.** Four *heads*. A head has three learned matrices $W_Q, W_K, W_V \in \mathbb{R}^{16\times 64}$ and computes, at every position, $q_t = W_Q x_t$, $k_t = W_K x_t$, $v_t = W_V x_t$, from which the weights are

$$a_{ts} = \frac{\exp(q_t \cdot k_s/4)}{\sum_{r\le t}\exp(q_t \cdot k_r/4)} \qquad (s \le t).$$

   So $a_{t\cdot}$ is a probability distribution over the current and earlier positions: *where the head looks from* t. The head's output is $\sum_{s\le t} a_{ts} v_s$. The four outputs are stacked, multiplied by a learned 64×64 matrix, and added to x_t. (q and k are first rotated by angles proportional to t, so that $q_t \cdot k_s$ depends on the distance $t-s$ as well as on the contents.)

2. **A small network at each position.** x_t goes to $W_2 \max(0, W_1 x_t)$, with $W_1 \in \mathbb{R}^{256\times 64}$, $W_2 \in \mathbb{R}^{64\times 256}$, and the maximum taken entrywise. It is added to x_t. We call this the *MLP*.

Finally the output is $p_t = \mathrm{softmax}(E^{\mathsf{T}} x_t)$, where $\mathrm{softmax}(z)_i = e^{z_i}/\sum_j e^{z_j}$. Before attention, before the MLP, and before the output, the vector is divided by its root mean square and multiplied by learned numbers, one per coordinate (*RMSNorm*).

tt-2 has two layers, numbered 0 and 1, and 115,008 learned numbers. `L0H3` means head 3 of layer 0, counting from 0.

The learned numbers are found by minimising the average cost $-\log_2 p_t(b_{t+1})$ over training text, 32 windows at a time, each starting at a uniformly random place in the text. Each such *step* moves every number a little in the direction that lowers the cost (the method is AdamW). The size of the move, the *learning rate*, rises to a peak over the first 200 steps and then falls to a tenth of it; the peak is 0.003 unless stated. The training text is the first 10^7 bytes of enwik9. An *epoch* is 2441 steps, which make 10^7 predictions; since windows start at random places, some bytes are used more often than others. The random choices (initial numbers, window positions) come from a *seed*.

What is measured

bpc (bits per byte): the average of $-\log_2 p_t(b_{t+1})$ on 10^6 bytes of enwik9 the model has not been trained on (bytes 10^7 to $1.1 \cdot 10^7$), cut into non-overlapping windows. Lower is better. Restarting in each window costs something: tt-2 averages 4.06 bits at the first position, about 2.16 from position 32 on, and 2.2183 overall. A difference of 0.1 is large here.

The copying test. Take a string of R random printable characters (R from 24 to 64, drawn per test), and repeat it until it fills a window:

position	0	1	2	3	4	5	6	7	8	9	10	11
byte	k	7	?	Q	z	%	k	7	?	Q	z	%

(here $R = 6$, for illustration). Inside the first copy the bytes are random and cost about $\log_2 95 = 6.57$ bits each, since there are 95 printable characters. From the second copy on, each can be predicted almost free: look at the same place in the previous copy and predict what came next. At position 8 (the second `?`) the right place to look is position 3, the byte after the first `?`. With R drawn at random, a head that always looked the same distance back would not find it.

Mass. For one head, the average of $a_{t,\,t-R+1}$ over 1000 such windows and all positions $t \geq R$: the share of its attention it puts on the right place. (*Mass* as in probability mass: $a_{t\cdot}$ is a distribution, and this is the probability it gives to one place.) We report the largest value over all heads. A head that spread its attention evenly would score about 0.02. One that finds the earlier occurrence scores near 1, or somewhat less if the window holds several earlier copies and it divides its attention among them (a window holds up to five copies, and mass counts only the nearest). *All copies* adds up a_{ts} over every earlier copy. Mass says where a head looks, and not whether the prediction is right.

Copy loss. The average cost in bits, on the same windows, at positions after the first copy. A model that has learned the alphabet and nothing else costs about 6.57; perfect copying costs 0. The test strings are not English, so a low cost cannot come from knowledge of English; only copying can produce it. tt-2 and its variants were trained on no such strings. They cost above 9 bits: they give each random character a small probability, because it would be rare in English at that place, and 9 bits is a probability of $1/512$. The informative comparison is whether the cost falls after the first copy.

Removal. What a part contributes, measured by taking it out: the part's output at every position is replaced by its average over the window (or by zero), and the copy loss is measured again on 400 windows. A part is a head, or the MLP; we can also remove only what it passes to the q, k or v of layer 1.

Copy gain. The saving available to a copying rule on enwik9 text. At each position find the longest earlier match of the bytes just read, and let c be the byte that followed it. Replace p_t by $(1-\lambda)p_t + \lambda\,\mathbf{1}_c$, with λ chosen separately for each range of match length on half the windows, and measure the bits saved per byte on the other half. It is a lower bound on what copying would be worth to the model: tt-2 could save at least 0.057 bits per byte.

Scoring. A prediction with probability P, whose outcome is $o \in \{0, 1\}$, scores $(P - o)^2$. The total is the mean. Saying $\frac{1}{2}$ always scores 0.25; lower is better.

The runs

Run	What it is	bpc
tt-2, tt-2-s2	the two-layer model on enwik9, 8 epochs; two seeds	2.2183, 2.2148
tt-2-ep1	the same, 1 epoch	2.4352
tt-1	one layer instead of two, 8 epochs (65,728 numbers)	2.4789
tt-2-swiglu	a different MLP: entrywise product of two linear maps of x_t, one of them passed through a smooth threshold; width 176, 117,056 numbers; 8 epochs	2.1961
tt-2-nonorm	no division by the root mean square; 114,688 numbers; 8 epochs	2.3343
diag-synth100	the two-layer model trained *only* on copying tests, 2 epochs of steps	7.62
synth-s2	the same, another seed	7.49
synth-tt1	one layer, only copying tests, 2 epochs	6.38
diag-synth25	a quarter of the windows copying tests, rest enwik9, 1 epoch	not measured
restart-ctl	diag-synth100 trained a further $\frac{1}{4}$ epoch on copying tests	7.75
survive	diag-synth100 trained a further epoch on enwik9	2.4291
survive-s2	synth-s2 trained a further epoch on enwik9	2.4486
survive-lr3e4	as survive, peak learning rate 0.0003	3.088
warm-ep1, warm-ep8	1000 steps of copying tests, then enwik9; 1 or 8 epochs in all	2.7215, 2.2158
mix10-ep8	one window in ten a copying test throughout, 8 epochs	2.2293

The nineteen predictions, scored

	Prediction	Measured	P	Outcome	$(P-o)^2$
1	synth-s2: removing one layer-0 head raises the copy loss above 5	L0H1 6.09, L0H3 5.10	0.75	true	0.063
2	synth-s2: removing what the layer-0 MLP passes to layer 1 costs more than removing any one layer-0 head	MLP 4.45, L0H1 4.79*	0.60	false	0.360
3	survive: mass at least 0.3	0.092	0.65	false	0.423
4	survive: bpc below tt-2-ep1's 2.4352	2.4291	0.75	true	0.063
5	warm-ep1: mass at least 0.3	0.024	0.55	false	0.303
6	warm-ep1: bpc above 2.4352	2.7215	0.60	true	0.160
7	warm-ep8: mass at least 0.3	0.012	0.55	false	0.303
8	warm-ep8: bpc below tt-2's 2.2183	2.2158	0.60	true	0.160
9	mix10-ep8: mass at least 0.3	0.018	0.85	false	0.723
10	mix10-ep8: copy loss below 1	6.48	0.80	false	0.640
11	mix10-ep8: bpc below 2.2183	2.2293	0.45	false	0.203
12	mix10-ep8: copy gain below tt-2's (0.057)	0.0568 (tt-2 0.0567)	0.75	false	0.563
13	tt-1: bpc more than 0.05 above tt-2's	2.4789	0.90	true	0.010
14	tt-1: mass below 0.05 in every head	0.004	0.95	true	0.003
15	tt-2-swiglu: bpc at least 0.01 below tt-2's	2.1961	0.60	true	0.160
16	tt-2-swiglu: mass below 0.1	0.004	0.85	true	0.023
17	tt-2-nonorm: bpc above tt-2's	2.3343	0.65	true	0.123
18	tt-2-nonorm: bpc above 5 (training failed)	2.3343	0.10	false	0.010
19	tt-2-nonorm: mass below 0.1	0.005	0.88	true	0.014
	mean of nineteen				**0.226**

The total is slightly better than always saying one half. It hides a split. Lines 13–19, on the variants of tt-2, average 0.049. Lines 1–12, on the copying circuit, average 0.330, worse than one half. Lines 3, 5, 7, 9 and 10 gave 0.55 to 0.85 to a circuit forming or lasting; all five were false, and they average 0.478.

Two lines are close. Line 12: the prediction was written with tt-2's gain rounded to 0.057; the exact value is 0.0567 and the run's is 0.0568, so it is scored false against the exact value. Scored the other way

the total is 0.200. Line 8 passes by 0.0025. The two tt-2 seeds differ by 0.0035, so 0.0025 is not evidence of anything. *Line 2 is measured by window-average removal; with zero removal it would be true (MLP 5.22, L0H1 5.10).

What the runs show

No model trained on enwik9 text ended with a copying circuit of the strength seen in the models trained only on copying tests. The largest mass that was left on a text-trained model is 0.20 (survive-lr3e4), which started from a circuit; in runs that did not start from one it is 0.018 or less at the end, apart from warm-ep1 (0.024).

The code can learn to copy. On copying tests alone, two layers copy almost perfectly (copy loss 0.02) and one layer does not (5.44). The two seeds that were trained this way built different circuits. In diag-synth100, removing `L0H3` raises the copy loss to 6.93, no better than knowing the alphabet (6.57). In synth-s2, two heads are each essential (`L0H1` 6.09, `L0H3` 5.10). What the MLP passes to layer 1's q matters in the first (removing it costs 4.6) and not in the second (0.04). The mechanism as a whole is not established.

The match uses more than one byte. Consider a predictor that knows only this: take the last k bytes read, find every earlier place in the window where they occurred, and predict uniformly among the bytes that followed. Its copy loss is 0.471, 0.010, 0.005 and 0.005 for $k = 1, 2, 3, 4$. The trained models reach 0.02, so they use more than the current byte; how is untested.

Text takes the circuit apart.

Run	What was done to it	bpc	mass	all copies	copy loss
restart-ctl	copying tests, $\frac{1}{4}$ epoch	7.75	0.809	0.955	0.024
survive	enwik9, 1 epoch	2.429	0.092	0.116	9.18
survive-s2	enwik9, 1 epoch, from the other seed's circuit	2.449	0.101	0.127	9.10
survive-lr3e4	enwik9, 1 epoch, a tenth of the learning rate	3.088	0.201	0.253	8.03
warm-ep1	1000 steps of copying tests, then enwik9	2.722	0.024	0.029	9.91
warm-ep8	the same, 8 epochs in all	2.216	0.012	0.014	9.47
mix10-ep8	one window in ten a copying test, 8 epochs	2.229	0.018	0.020	6.48

The first row is a control: training further on copying tests, at the same learning rate, leaves the circuit intact (mass 0.81, from 0.82). One epoch of text takes it to 0.09. A tenth of the rate leaves more, 0.20, but at a much worse bpc. In warm-ep8 the first 1000 steps built a circuit (mass 0.80); it was 0.07 after 250 steps of text, 0.03 after 500, and no later value logged was above 0.03 in the 18,000 steps that followed.

One window in ten is not enough to build one. In mix10-ep8 no head passed 0.02 mass at any logged step. Its copy loss, 6.48, is just below $\log_2 95 = 6.57$: it learned which characters the tests are made of, and not to copy them. It saw about 62,000 copying tests; diag-synth100 had its circuit after 32,000. The cause is untested. A run with one window in four (diag-synth25, one epoch) also formed none.

One layer is worse at text and no test of copying. tt-1 is 0.26 bpc above tt-2. It has no copying circuit (mass 0.004), but neither has tt-2 (0.004), so the two do not contrast on text. On copying tests alone they do: 5.44 against 0.02.

Replacing the MLP by the swiglu form gains 0.022 **bpc; dropping the norm costs** 0.116**.** Neither changes the copying result. Since the two tt-2 seeds differ by 0.0035, a gain of 0.022 is unlikely to be seed noise, though swiglu itself was run once. The variant without the norm trains, where it might not have.

How far the numbers can be trusted

- The 0.0035 between tt-2 and tt-2-s2 is one difference from two seeds, a rough guide. On it, warm-ep8 (2.2158) lies between the two, and mix10-ep8 (2.2293) is 0.011 to 0.015 above both. Scored against tt-2-s2 instead, lines 8 and 12 would change (8 false, 12 true), the rest would not, and the mean would be 0.211. Lines 4 and 6 compare against tt-2-ep1, which has no second seed.
- Survive and survive-s2 differ by 0.020 bpc. They started from different circuits, so this is not the seed spread above. Line 4's margin (0.006) is inside it, and survive had also taken 4882 extra steps first.
- One binary with one seed does not give the same numbers twice: the weights differ by 0.4% after 122 steps, probably because two threads' gradients are summed in a varying order. Some of the 0.0035 is this.
- Survive-lr3e4 is the only run at another learning rate. A warm-up at a lower rate has not been run, so warm-ep1 and warm-ep8 do not separate the effect of text from the effect of the learning rate at the switch.
- Copying tests are random characters, unlike anything the text-trained models have seen. On repeated held-out enwik9 text instead, the largest mass is 0.078 (survive), 0.127 (survive-s2), 0.035 (warm-ep1), 0.016 (warm-ep8), 0.005 (mix10-ep8).
- Window-average removal uses later positions in its average, whose effect is unmeasured.
- Mass counts only the nearest earlier copy. In diag-synth100 a head scores 0.82 by it and 0.95 over all copies; no finished line's outcome depends on the difference.

The revised probabilities

Lines 5–12 were revised at 09:58 UTC on 2026-11-24, when warm-ep1 had done 250 of its 2441 steps and none of its measurements had been seen; lines 13–19 were left as they were. The scores above use the originals.

	Prediction	P	Revised	Reason
5	warm-ep1: mass ≥ 0.3	0.55	0.07	survive lost its circuit at this learning rate
6	warm-ep1: bpc above 2.4352	0.60	0.75	survive gained only 0.006 from 4882 extra steps
7	warm-ep8: mass ≥ 0.3	0.55	0.06	as 5
8	warm-ep8: bpc below 2.2183	0.60	0.45	loses 5% of text steps
9	mix10-ep8: mass ≥ 0.3	0.85	0.75	text erodes the circuit
10	mix10-ep8: copy loss below 1	0.80	0.72	as 9
11	mix10-ep8: bpc below 2.2183	0.45	0.35	10% fewer text windows, one seed
12	mix10-ep8: copy gain below 0.057	0.75	0.70	depends on 9

Over these eight lines the originals score 0.382 and the revised ones 0.258, both worse than one half.