Rebuilding tt-2 as a UM: Plan and Two Questions

Claude and MJC

2026-11-25

tt-2 is the small transformer of the previous paper: 115,008 learned numbers, trained on the first 10^7 bytes of enwik9, predicting each byte from those before it. Step 3 is to rebuild it as a UM whose events have names. This paper shows one of its predictions taken apart, proposes an order of work, and asks two questions (Section 5). It starts with which model to rebuild, since a model that sees fewer bytes would be easier to interpret.

1 A shorter context

How much of the past tt-2 uses. tt-2 was trained on windows of 128 bytes. To see what the early part of the window is worth, the same bytes were predicted several times, each time with only the last n bytes visible.

Bytes visible, n	8	16	24	32	40	64	128
training text	+0.184	+0.061	+0.029	+0.012	+0.005	−0.002	0
unseen text	+0.136	+0.050	+0.021	+0.010	+0.005	0.000	0

Entries are bits per byte (bpc) above the cost with all 128 visible, on 32,512 bytes of each text; the standard errors are 0.005 at $n = 8$ and 0.002 or less from $n = 24$. **With only the last 32 bytes visible tt-2 loses about 0.01 bpc**, and it was not trained for that. The cost with 128 visible is not quite the lowest: between 88 and 120 bytes it is 0.002 to 0.005 lower. Against the best length the loss at 32 is 0.014 (unseen) and 0.017 (training).

There is also a cost that a short context removes. Scored on whole windows of 128, as in the previous paper, the bytes at the start of each window have little to go on; that is about 0.055 bpc in the average. If every byte is predicted from its own last 32, no byte has a short start, except the first 32 of the file.

Two ways to make the limit exact.

Model	Change	Output at t depends on	What it means for the UM
tt-2-ctx32	windows of 32 bytes, in training and after	bytes $t-31..t$	each predicted byte needs its own window, and the same earlier byte gets different vectors in different windows
tt-2-span16	windows of 128, but each head may look only at the last 16 positions; there are two layers, so at most $15+15$ back	bytes $t-30..t$	each head's event space is the same 16 events, "k bytes back", $k = 0..15$, at every position

In either, nothing before $t-32$ can affect the output. The second also computes each position's vectors once, as the same function of the bytes before it everywhere past the first 30, so an event found at one place

means the same at another. I expect it to be the better one to interpret. Both are being trained now, in the same way as tt-2. The numbers follow in a short separate paper; the rest of this plan holds for either.

2 Where generalizing matters

The prize is won on 10^9 bytes and nothing outside them, so at full size a pattern only has to save more bits over those bytes than it costs to write down; whether it holds anywhere else is irrelevant, and a model that memorizes is fine: between two models we take the one better on 10^9, however badly it would do elsewhere. Small experiments are different: a pattern found in 10^7 bytes is useful to us if it keeps paying on the other 99%, so in them generalizing is a prediction about what pays at 10^9. An example from the text: the nine bytes `signed␣an` occur 13 times in the first 10^7 bytes; if that rate held they would occur 1,300 times in 10^9, enough to be worth a pattern. Whether the rate holds is a judgment of ours, and is the kind of judgment we want to make directly, pattern by pattern. The measured size of the question for tt-2: on ten 10^6-byte pieces of its training text it costs 2.108 bpc, on the next ten 2.243, a gap of 0.136 with a standard error of 0.021. That mixes what tt-2 fitted in particular with any difference between the texts.

3 One prediction taken apart

The sum. At each position tt-2 ends with a vector of 64 numbers, x. It is the sum of what was written into it: the vector of the current byte; what each of the four *heads* in each of two *layers* wrote (a head takes a weighted average over earlier positions of a linear function of them, the weights adding to one); and what each of the 512 *neurons* wrote, 256 per layer (neuron i is $m_i = \max(0, w_i \cdot v)$, where v is the vector at that point in the sum, scaled, and it adds m_i times a fixed vector). The score of byte b is $e_b \cdot x$ with a scale that depends on the position but not on b, and the probability of b is proportional to exp(score).

So for two bytes b, b', $\ln p(b)/p(b')$ is a sum, one term per writer, with nothing left over. In the UM's terms: every writer gives support to output events, and within the output event space only differences count. The example, `tt terms`, after the text ... `after␣17␣years␣and␣signed␣an`: tt-2 gives probability 0.815 to `d` and 0.132 to a space.

By group		The ten largest of the 521 terms, $\ln p(\texttt{d})/p(\text{space})$			
current byte	+0.557	L1N228	−2.872	L1N163	+1.056
layer 0 heads	−0.816	L1N234	−1.770	L1N155	+1.021
layer 0 neurons	+1.040	L1N76	−1.374	L1N139	+0.828
layer 1 heads	−0.749	L1N159	+1.268	L1N101	−0.794
layer 1 neurons	+1.791	L1N49	+1.247		
total	+1.823	L1N54	+1.098		

L1N228 is neuron 228 of layer 1. The total equals $\ln(0.815/0.132) = 1.82$ from the softmax to four decimals, the check that the sum is complete. There are 109 nonzero terms, and their absolute values add to 27.7, so most of it **cancels**: 27.7 of support for and against to leave 1.82. This was the first of six positions taken apart, spread over the training text: they have 100 to 167 nonzero terms, and in the five where the total is not near zero the absolute values add to 5 to 21 times the total (15 here). That is the main thing to plan around.

The same support, counted. In the first 10^7 bytes, `␣an` occurs 42,069 times, followed by `d` 32,358 times and by a space 4,509: $\ln(32358/4509) = 1.971$. One counted pattern gives nearly what the 521 terms leave (1.82). Longer contexts: `d␣an` 1.085, `ed␣an` 1.002 (678 against 249), and `signed␣an` is 6 against 6 in 13. In this one example tt-2 sits closest to the count of the last three bytes, and not to the counts for longer contexts.

What neurons look like. The strongest 1% of positions of two of them, over 192,000 positions of the training text:

Neuron	Where it is strongest	Next byte there
L1N40	after a comma mostly (`,` is the last byte in most of the commonest endings), some after ␣`to` and `).`	space 94.2%
L1N228	␣`an` 36%; `quo` 9%; ␣`to` 6%; line starts such as `>`, newline, space 6%; ␣`he` 5%; ...	space 48.9%, `d` 28.8%, `t` 10.6%

The first could have a name, the second has none: it is positive at 19% of positions, and in the example it argues for a space over `d` by 2.9 where other neurons argue back.

4 Proposed order of work

1. **Choose the model**: tt-2-ctx32 or tt-2-span16, whichever is cheaper in bpc, and tt-2-span16 if they are close. Everything below is for the chosen one.
2. **Ledger.** The sum of Section 3, at every position of the training text, checked against the model's own probabilities. This is the real-valued UM of tt-2. It must match; if it does not, that is a bug.
3. **What each piece is worth.** The bpc lost on the training text when each head and each layer's neurons are removed, and when groups of neurons are. Groups, since terms cancel and two terms may do one job.
4. **Look at the data.** For the pieces worth most, list the positions where they fire, the contexts and the next bytes. Name events, and write beside each whether it should hold over all of enwik9, only here, or we cannot say, and why.
5. **Replace by counting.** Build a UM with the named events, with strengths $\log_2$ of counts in the training text, and measure its bpc there, and its KL divergence from tt-2 (the average over positions of the extra bits paid for a byte drawn from tt-2's distribution when the UM's is used to code it).
6. **The max-min form**, strengths in $[0, 255]$, $f_p(t)_j = \max_i \min(t_i, p_{ij})$, measured last, pattern by pattern.

The earlier plan, in the literature review, had the max-min form straight after the exact real-valued UM. The example is why I would now do it last: the absolute values are 15 times the total in the example, and a rule that takes the largest of them keeps the largest term and loses the cancellation. Max-min over named, counted patterns has no such sum to lose, though it is not free: over counted patterns flat max-min has cost this project about 2.9 bpc against interpolation (March 2026), so step 6 needs a rule chosen for it.

5 Two questions

Question 1. Is it right to name and count before max-min?

Order	For	Against
Name and count first (proposed)	a loss can be traced to a pattern; counts are the support the UM means; the cancellation above should be gone before max-min meets it	the exact tt-2 in max-min form is never measured, so we do not learn what max-min alone costs tt-2
Max-min first	measures the cost of the rule with tt-2 otherwise unchanged; a number that stands alone	with 109 terms cancelling, the loss is likely large, and says little about the rule on a better factorization

I recommend the first, with the second as one measurement on the side if you want the number.

Question 2. When we replace a piece of tt-2 by a count, what do we count?

Event counted	For	Against
The neuron's own event (it is positive, or strong)	follows tt-2 exactly; no judgment of ours enters	L1N228 above: no name, so nobody can say whether it holds beyond 10^7 bytes
A named event, such as "the last three bytes are `␣an`" (proposed)	can be judged, and can be checked by reading the text; the pattern is the UM's own kind; the difference from tt-2 is a measured error	we choose the events, so tt-2 only suggests them; a piece we fail to name stays unexplained, and that part must be reported

I recommend the named event. Section 3 has two signs that it may go well: one count gave nearly what 521 terms leave (one example), and a neuron like L1N40 is close to a name.

The danger in that choice, and how we would see it. Events that are just "the last k bytes are ..." make an ordinary table of counts. If that is all we name, we have not learned anything from tt-2 that we could not have got by counting. What tt-2 should add is what it shares: contexts it treats alike (after `]]` or a comma), and distinctions in state a suffix does not show (inside a link, inside a tag). So each stage reports, next to the bpc, the bpc of a table of counts of the last k bytes, for the best k. A named set that does not beat that table per pattern, or that gets its bpc from a great many patterns, is not an explanation of tt-2.

6 Not decided here

Which of the two short-context models, until both are trained. Whether to redo the train/held-out gap on them.