Episode Book

EPISODE #6502

What a Cross-Entropy Loss of 1.8 Actually Means

PUBLISHED	8 November 2026 at 14:43 UTC
DURATION	25:22
LISTEN	myweirdprompts.com/episode/interpreting-cross-entropy-loss-perplexity-50396/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	25 cited by the research (listed at the end)
BOOK GENERATED	9 November 2026 at 08:09 UTC

SYNOPSIS

Training runs live and die by cross-entropy loss curves, yet few can translate a single loss number into plain terms. This breakdown walks through how a loss of 1.8 nats translates directly into an average perplexity of six branching choices, why language modeling mathematically equates to compression, and how differing tokenizers produce deceptive per-token numbers. It also explores why tools like EleutherAI's evaluation harness and Karpathy's nanochat bypass token perplexity entirely in favor of bits per byte.

TRANSCRIPT



Corn

One point eight. That's the number Daniel admits he's been staring at on loss curves for months, and his confession is that he couldn't tell you what it means. His prompt is refreshingly direct: every training run is watched through a single number, the cross-entropy loss, and its exponentiated cousin perplexity gets quoted constantly, but "few people who stare at loss curves could say what a loss of 1.8 actually means." He wants the plain-terms reading, how it connects to perplexity and bits per byte, why losses aren't comparable across models with different tokenizers, and why a lower loss doesn't automatically get you a more useful model.



Herman

And the beautiful part is that the arithmetic is trivial. Everything interesting is in the interpretation.



Corn

You've been waiting all week to say that.



Herman

I have. Start with what the loss on a single token is. The model puts a probability on every possible next token. Exactly one of them is correct. The loss for that token is minus the natural log of the probability it gave the correct one. If the model gave the right token sixty-six percent probability, the loss is about zero point four nats. That's it. The training loss is just the average of that quantity over millions of tokens.



Corn

So when someone shows me a loss curve, I'm looking at average surprise.



Herman

Average surprise at the correct answer. If the model was certain and right, surprise near zero. If it gave the right answer one chance in a hundred, that token costs about four point six nats. Everything else — the summation signs, the one-hot vectors, the KL divergence you'll drag in later — is bookkeeping around that one operation.



Corn

You said nats. That's the unit people skip.



Herman

Because PyTorch computes cross-entropy with the natural logarithm, the loss comes out in nats, not bits. One nat is one over ln two bits, roughly one point four four. A loss of one point eight nats is about two point six bits per token. If you want bits, divide by point six nine three. People quote loss numbers across papers without ever saying which unit they're in, and since everyone uses PyTorch, everyone's in nats, and nobody says so.



Corn

So the number is simple, the interpretation is where it earns its keep. Take the one point eight. What does it actually tell me about the model?



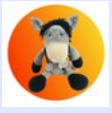
Herman

Exponentiate it. e to the one point eight is about six. That's the perplexity. And the plain reading of a perplexity of six is: on average, the model is about as uncertain over the next token as if it were choosing uniformly among six equally likely options.



Corn

Six equally likely next tokens. Every single step.



Herman

Every step, on average. Some tokens it's near-certain on — the closing quote, the space after a period — and some it's lost on. The perplexity is the branching factor of a uniform distribution that would produce the same average surprise. Same number, different clothing.



Corn

Six feels almost insultingly small. There are fifty thousand tokens in a typical vocabulary.



Herman

That's what makes it a decent model! A uniform guess over a fifty-thousand-token vocabulary is a perplexity of fifty thousand. Six means the model has concentrated its probability mass onto a handful of plausible continuations almost all the time. Roy has a nice worked example — a five-token sequence with joint probability about one point one times ten to the minus five, which comes out to a cross-entropy of one point nine nats and a perplexity of six point seven. So "a loss around one point nine" and "picking among about six or seven options" are the same statement in two units.



Corn

The one thing I'd add, because it took me embarrassingly long to internalize: perplexity is not a probability. People say "perplexity of six" like it's a score out of a hundred. It's the exponent of the loss. A model going from perplexity six to perplexity three is halving its average surprise, not improving by three points.



Herman

Right — and the geometric-mean framing makes that exact. Perplexity is the inverse of the geometric mean of the token probabilities. Which is why one catastrophic token drags it far more than one confident token helps it.



Corn

You promised me compression earlier. Off the kitchen counter, so to speak.



Herman

The compression reading is my favorite, and Dive into Deep Learning states it about as cleanly as it gets stated: a loss of one point eight nats per token is a file size, a perplexity is a branching factor, and choosing between two models becomes a question of which complete description of the text is shorter. Shannon's source coding theorem says the expected code length for data following distribution p, when you encode it using a model q, is the cross-entropy. So the training loss is literally the length, in nats per token, of the compressed file you'd get if you used the model as the compressor.



Corn

So a model with lower loss is a better zip file.



Herman

For text drawn from the same distribution, yes. That's the cleanest one-line identity in the whole field: language modeling is compression. And it hands you the bits-per-byte conversion for free, but I want to earn that properly, because there's a subtlety about tokenizers coming, and if I set it up now, it'll land harder.



Corn

Wait. Before the subtlety. The decomposition — you said cross-entropy isn't the whole story on its own.



Herman

Cross-entropy decomposes into two parts: the entropy of the data plus the KL divergence between the model and the truth. Entropy measures one distribution against itself; cross-entropy measures the cost of using the wrong one. And here's the kicker for training: the entropy of natural language is fixed. You can't lower it. Whatever the true distribution of text is, it has whatever entropy it has. So when you minimize cross-entropy during training, the only thing you can actually be reducing is the KL term — the gap between your model's distribution and the data's. That's why this loss, out of all possible losses, is the one everybody uses: gradient descent on it is provably pushing your distribution toward the data's.



Corn

That's the honest defense of the number, then. It's not arbitrary, it's a distance.



Herman

A distance in whatever units the tokenizer chose — hold that thought, it's the next act. First, the units fix. Bits per byte.



Corn

Give me the formula before the intuition.



Herman

Bits per byte is cross-entropy renormalized per byte instead of per token. You take the mean loss in nats, multiply by the number of tokens your tokenizer produced, and divide by the number of bytes in the text times ln two. Tokens on top, bytes on the bottom. The intuition: how many bits, on average, would the model need to encode each byte of this text? Lower is better, and the denominator — the byte count of the text — is identical for every model looking at the same text.



Corn

And this is where you've got a real chain, not a toy example.



Herman

Regnant published a Kiswahili evaluation in September where they ran a hundred-and-forty-nine-million-parameter base model over held-out text. Mean loss, three point one seven nats, over just under sixty-five thousand tokens. The text itself was about three hundred and twenty-one thousand bytes. Multiply the loss by the tokens, divide by bytes times ln two, and you get zero point nine two six bits per byte. Every step of that is auditable — you can check the arithmetic yourself from the four numbers.



Corn

Under one bit per byte. Which, for anyone who's compressed text, is...



Herman

Respectable. Plain gzip on English gets you somewhere around two and a half bits per byte. A model that knows Kiswahili getting under one is doing real work.



Corn

And the reason they went to bytes rather than just quoting the three point one seven is the part I suspect Daniel was really poking at. Why can't you put two models' losses side by side?



Herman

Because the denominator changed. Here's the cleanest demonstration I know, from Dip Kumar's write-up. Two models predicting the phrase " is Delhi" — eight bytes of UTF-8. Model A has a coarse tokenizer and splits it into two tokens, with losses of one point five and four point five nats. Model B has a finer tokenizer, splits it into three tokens, with losses of one point five, two point zero, and two point five.



Corn

Both total six nats.



Herman

Both total exactly six point zero nats. Identical models, as far as the text is concerned. But Model A's per-token loss is three point zero, perplexity about twenty. Model B's per-token loss is two point zero, perplexity about seven point four. Model B looks nearly three times better, and it's the same six nats spread over more tokens. Compute bits per byte — six over eight bytes times ln two — and both come out at one point zero eight. Identical, because they should be.



Corn

The perplexity gap was pure tokenizer.



Herman

Pure tokenizer. And the bias runs in whichever direction the tokenizer moved. This bit people in public: there's a long llama.cpp thread from when Llama 3 came out, because Llama 2 had a thirty-two-thousand-token vocabulary and Llama 3 went to a hundred and twenty-eight thousand. Llama 3's per-token perplexity scores came out much higher, and people panicked — new model, worse number? No. Each Llama 3 token predicts more underlying bytes, so each prediction is harder, so the per-token number is bigger. Finer tokenizer, higher per-token loss, same or better model.



Corn

So the folk rule "lower loss is better" has a hole in it before you even ask whether the model is any good. A coarser tokenizer flatters you for free.



Herman

It's worse than a hole, it's a systematic bias that people run leaderboards on. EleutherAI's evaluation harness straight-up refuses to report token-level perplexity. There's a GitHub issue from twenty twenty-three where one of the contributors says it plainly: this is a tokenizer-dependent metric, and it's not possible to compare token-level perplexities between models. The harness reports word perplexity, byte perplexity, and bits per byte instead.



Corn

The harness has a stronger position than most people who quote the number.



Herman

The harness is ahead of the discourse, yes.



Corn

And Karpathy made the same call with nanochat.



Herman

Made it the headline metric. Validation bits per byte is the number nanochat reports during training, and his stated reason is exactly this — it normalizes each token's loss by the number of bytes that token represents, so it's tokenizer-independent. The code comment calls it apples to apples even if you change the vocab size. Which, in a repo where the whole point is that a hobbyist can train the model, is the right call — you don't want people comparing their run to someone else's run and drawing conclusions from the vocab.



Corn

Herman, we should take a listener call before you get to the deeper problem, because you're about to build a tower and I want a witness.



Herman

Fair. Let's hear from a listener. Doug, you're on the air.
Doug: Hi Corn and Herman, this is Doug from Kitchener, Ontario, I'm calling from the cab of my truck while I wait for a load at the yard, and I've been thinking about today's episode on lost functions and perplexing city, I think you called it perplexity. My question is, when you say the model is as uncertain as picking among six equally likely tokens, does that mean if I asked it to finish a sentence it's basically rolling a six sided die, because I once watched a model finish a sentence about my dog and it felt like it had way more than six choices in mind, and I guess I'm just wondering how that number squares with how creative it feels.



Corn

Doug, thanks for the call. The die is a good picture with one fix: the six sides are an average, not a limit. The model is looking at every token in the vocabulary every time — fifty thousand sides on the die, some huge, some slivers. Six is just the effective count.



Herman

And the creativity you saw is exactly the slivers. Most steps the model is confidently narrowing to a few obvious words, and occasionally it's spreading mass across dozens of different continuations. Average those together and you get six, the same way a road with one hill and a hundred flat kilometers is, on average, flat.



Corn

Which is also, helpfully, the exact flaw in the number. Carry on, Herman.



Herman

There's one deeper layer on comparability, and it's Mielke's point from back in twenty nineteen, which I think is the sharpest thing ever written on this. You can only compare distributions on likelihood metrics if they have support on the exact same set of strings. Corollary: closed-vocabulary and open-vocabulary perplexities are not comparable at all. A word-level model can't assign probability to a word it's never seen — wolpertinger, say —



Herman (continued)

so it dumps all its mass on in-vocabulary strings. On a test set with no novel words, it gets a much lower perplexity for free. Meanwhile the character-level model has to spread its mass thin over every possible spelling. Her word for it is that the closed-vocab model gets to cheat.



Corn

Any metric that rewards concentration of probability mass can be gamed by a model that declines to cover the space.



Herman

That's the general law, yes. And the Kiswahili table is the demonstration in the wild. Regnant ran nine models on the same Kiswahili text with vocabularies running from thirty-two thousand tokens up to two hundred and fifty-six thousand, and their line is that that one fact decides which numbers can be put side by side. On bits per byte: the Kiswahili-trained model at zero point nine three, XGLM at one point one, BLOOM at one point seven, GPT-2 at two point one, and Qwen two point five half-billion at two point three nine.



Corn

The Qwen is the biggest model in that table.



Herman

The largest model in the table scores the worst, because it never trained on Kiswahili. All that capacity, and it compresses the language worse than a hundred-and-twenty-four-million-parameter GPT-2. Size buys you nothing the training data didn't already give you.



Corn

Same text, same denominator, and the ranking actually means something. Okay — so bits per byte fixes the tokenizer problem. It does not fix the question of whether the model is any good.



Herman

That's the second half of Daniel's prompt, and it's where the number goes from "tricky" to "quietly misleading." Perplexity averages over all tokens. Most tokens are easy. In a long document, the overwhelming majority of next tokens are the, is, of, a comma. The model would get them right almost regardless of quality. The information-carrying tokens — the ones that resolve a long-range dependency, the answer the question is actually asking about — are a thin slice. Averaging buries them.



Corn

There's a paper on exactly this, isn't there.



Herman

LongPPL, twenty twenty-four. Their finding: standard perplexity shows almost no correlation with long-context benchmark scores. Almost none. Recompute perplexity only on the answer tokens — the few that matter — and the correlation with long-context accuracy hits minus zero point nine six.



Corn

From nothing to point nine six by changing which tokens you average.



Herman

Which tells you the mean was hiding the entire signal. And it's not just long context. There's a NAACL twenty twenty-five paper on kNN-LMs, the models that bolt a retrieval store onto a language model, and the finding is that they can reduce perplexity yet fail reasoning tasks even with perfect retrieval. Great memory, shallow reasoning. The retrieval hands the model the right fact, the loss drops because the next token became predictable, and nothing about the model's ability to reason improved.



Corn

That's the cleanest separation of the two axes I've heard. Lower loss from better memory, no change in reasoning.



Herman

And there's a community-side acknowledgment of the same problem. In the autoresearch discussions, people training agents on pure validation loss have flagged that isolated cross-entropy can disguise models that have collapsed in specific linguistic capabilities or are cheating through weird formatting. The loss keeps going down while the model is quietly getting worse at something you care about.



Corn

Herman, we did a whole episode once on benchmarking speech-to-text tools beyond the word error rate, and this is the same animal — one averaged number that's smooth and easy to watch, hiding everything you actually wanted to know. WER averages over easy words and buries the ones that matter.



Herman

Same disease, different metric. And the reason the disease persists is that the cure is expensive. Someone running a homelab nanochat reproduction put it well: validation bits per byte is the primary training signal because it's smooth and low-noise compared to task-based evaluations. You get a meaningful number every few minutes of training, versus a benchmark suite that costs a fortune and jitters.



Corn

So the loss is a steering wheel, not a map.



Herman

Nice, but incomplete — there's a twist in TokEval worth holding onto.



Corn

Before the twist, the question I keep circling: if everyone in the field knows this, why does the number stay on the dashboard?



Herman

Because it's cheap, dense and low-variance. You compute it for free as a byproduct of training. Every batch gives you another reading. Benchmarks are sparse, expensive and noisy. So you steer on the loss because you must — you need a dense signal to run gradient descent — and you check the benchmarks occasionally to find out whether the steering went anywhere. The mistake isn't using the loss, it's quoting it as a capability.



Corn

The twist?



Herman

TokEval, this year. They ran controlled pretraining sweeps varying only the tokenizer, and found the information-theoretic tokenizer metrics predict language-modeling ability with Spearman correlation up to zero point eight — but the structure-sensitive metrics, things like digit alignment and line-break handling, correlate with task accuracy instead. Compression and capability are not the same axis. You can be an excellent compressor and a mediocre assistant.



Corn

Which is almost reassuring. If compression predicted everything, the field would be even more loss-obsessed than it is.



Hilbert

It would, and I've seen it happen with a till roll.



Corn

A till roll.



Hilbert

The shop on Rehov Argaman, electrical supplier, I've had an account there since the owner's father ran it. Two brothers, and the older one prints every counter sale on the roll and keeps the average per customer on a card behind the register. One number per week. He was proud of it, Corn, and he had reason to be, the man could tell you his week to a shekel. But the average hid the returns. Anything came back, it never touched the card, because the refund prints on the other roll, the one under the counter, the roll you change once a month whether it needs it or not.



Herman

The returns were on a separate tape.



Hilbert

Separate tape, separate average, one card. I only found out because I returned a cooker switch in March and watched where the slip went. Under the counter, into a shoebox. He knew exactly where. He'd shown me the card for years, never mentioned the box.



Corn

So his weekly number was true the whole time.



Hilbert

True and useless, same as yours. He retired, the younger brother took the register, first thing he did was move the refund roll next to the sales roll so they'd tear off together. I still get my terminal strips there. He rounds down the shekel on those, always has. He can't do that with the card, of course, the card stays.



Herman

The card stays but the box moved — that's the whole evaluation-harness argument in one register. Corn, that's the refunds being moved off the average. The younger brother is bits per byte.



Hilbert

Herman, he's a man with a screwdriver, not a metric. But the arithmetic's the same, I'll give you that. The claw situation on the packaging for those strips, by the way — heat-sealed plastic, a human invention nobody at that shop ever had to open with pincers.



Corn

He did audits for a living, Herman. Shop audits, some of them. You can hear it.



Herman

You can hear the shoebox. What I want to keep is the two rolls ending up side by side — the fix wasn't a better average, it was putting the hidden tape next to the counted one. That's exactly what LongPPL does with answer tokens.



Corn

And the field keeps the card on the dashboard because the card is free. Which raises the question Daniel's really sitting on: if the metric you watch isn't the metric you care about, what would a better training signal even look like? Nobody's built it, and the reason is boring — the cheap, dense number comes out of training whether you look at it or not, and the signal you actually want costs a benchmark run every time you ask for it. TokEval says compression and capability aren't even the same axis, so the honest version of the dashboard is probably two numbers forever: the loss to steer with, and something expensive to check against.



Herman

And the one number is real. A loss of one point eight is a true fact about the model — it's about six equally likely next tokens, about two point six bits a token, a file compressed to its size. It's just answering a narrower question than the one we keep asking of it.



Corn

That's the show, everyone. Thanks to Hilbert Flumingtop for producing, as ever. If this was your kind of episode, go back for episode eleven, How Does Fine Tuning Work Anyway; episode three, Safetensors or something else; and episode fourteen, AGI's Crossroads. This has been My Weird Prompts, the human-AI collaboration podcast. You can find us at my weird prompts dot com, and if the show's earned it, a review goes a long way.



Herman

Send your own strange ones somewhere warm.



Corn

We'll be back soon. See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Karpathy, Introducing nanochat: The best ChatGPT that $100 can buy, 2025-10-13, with training report dated 2025-11-11, retrieved 2026-11-08. [primary]**
   https://github.com/karpathy/nanochat/discussions/1

2. **nanochat `loss_eval.py` source, retrieved 2026-11-08. [primary]**
   https://github.com/karpathy/nanochat/blob/master/nanochat/loss_eval.py

3. **Wikipedia, Cross-entropy, last edited 2026-09-15, retrieved 2026-11-08.**
   https://en.wikipedia.org/wiki/Cross-entropy

4. **(information_theory)**
   Wikipedia, Entropy (information theory), last edited 2026-10-15, retrieved 2026-11-08.
   https://en.wikipedia.org/wiki/Entropy

5. **Wikipedia, Perplexity, last edited 2026-11-02, retrieved 2026-11-08.**
   https://en.wikipedia.org/wiki/Perplexity

6. **What is Wrong with Perplexity for Long-context Language Modeling? (LongPPL), full text read, retrieved 2026-11-08.**
   https://arxiv.org/abs/2410.23771

7. **Meister, TokEval: A Tokenizer Evaluation Suite, COLM 2026, revised 2026-09-09, retrieved 2026-11-08.**
   https://arxiv.org/abs/2608.18062v3

8. **ATMA: Long-Context Language Modeling via Polar Attention and Gated-Delta Compression Memory, revised 2026-08-12, retrieved 2026-11-08.**
   https://arxiv.org/abs/2606.25156v4

9. **Tacconelli, Nacrith: Neural Lossless Compression via Ensemble Context Modeling and High-Precision CDF Coding, revised 2026-02-24, retrieved 2026-11-08.**
   https://arxiv.org/abs/2602.19626v2

10. **Rethinking Perplexity: Revealing the Impact of Input Length on Perplexity Evaluation in LLMs, retrieved 2026-11-08.**
    https://arxiv.org/abs/2602.04099

11. **Great Memory, Shallow Reasoning: Limits of kNN-LMs, NAACL 2025 Short, retrieved 2026-11-08.**
https://doi.org/10.18653/v1/2025.naacl-short.40

12. **Dip Kumar, Bits-per-Byte (BPB): a tokenizer-agnostic way to measure LLMs, full text read, retrieved 2026-11-08.**
https://dipkumar.dev/posts/llm/bits-per-byte/

13. **Regnant Engineering, Why we report bits per byte, full text read, retrieved 2026-11-08.**
https://www.regnant.io/blog/why-bits-per-byte

14. **Sabrina J. Mielke, Can you compare perplexity across different segmentations?, 2019-04-23, full text read, retrieved 2026-11-08.**
https://sjmielke.com/comparing-perplexities.htm

15. **EleutherAI lm-evaluation-harness issue #806, comment 2023-08-27, retrieved 2026-11-08.**
https://github.com/EleutherAI/lm-evaluation-harness/issues/806

16. **EleutherAI lm-evaluation-harness issue #1544, comment 2024-03-08, retrieved 2026-11-08.**
https://github.com/EleutherAI/lm-evaluation-harness/issues/1544

17. **matt-langston, Reproducing nanochat in a homelab, 2026-04-16, retrieved 2026-11-08.**
https://github.com/karpathy/nanochat/discussions/710

18. **aniruddhaadak80, Feat: Lightweight downstream Zero-Shot Evaluation (Tiny-Eval), 2026-03-10, retrieved 2026-11-08.**
https://github.com/karpathy/autoresearch/discussions/294

19. **llama.cpp issue #7111, Making perplexity 'comparable' between models with different vocab: per-byte perplexity, retrieved 2026-11-08.**
https://github.com/ggml-org/llama.cpp/issues/7111

20. **Are Some Words Worth More than Others?, Eval4NLP 2020, retrieved 2026-11-08.**
https://aclanthology.org/2020.eval4nlp-1.13.pdf

21. **Multigrid, Cross-Entropy: What Training Actually Minimises, retrieved 2026-11-08.**
https://multigrid.ai/learn/cross-entropy-explained

22. **Dive into Deep Learning, §28.2 Codes, Compression, and Perplexity, retrieved 2026-11-08.**
https://d2l.smola.org/chapter_mdl-information-theory/mdl-coding-compression.html

23. **S. Roy, Probability for LLMs: Distributions, Entropy, and KL Divergence, retrieved 2026-11-08.**
https://swastikroy.me/blog/math-llm-probability

24. **MerceMay, Why Perplexity Is Not Comparable Across Tokenizers, retrieved 2026-11-08.**
https://mercemay.com/articles/inference-systems/evaluation-of-outputs/perplexity-tokenization-likelihood/

25. **Perplexity Is Not a Proxy, retrieved 2026-11-08.**
https://aimlds.org/d/2026.0061