A Smoothed Bigram Beats a 263-Million-Parameter Chat Model: Held-Out Measurements of a From-Scratch Chinese Dialogue Release

Daozhendong Yang
School of Intelligent Science and Engineering,
Yunnan Technology and Business University, Kunming, China
hi@yxpil.com
https://modelscope.cn/models/Yxpilow/Pillow_tiny_chat

Abstract

We measure `Pillow_tiny_chat`, a released Chinese dialogue model whose checkpoint carries 263,768,064 learnable float32 parameters in a 16-layer pre-LayerNorm decoder, against the cheapest distributions that can be built from its own corpus. On 98,176 held-out positions of LCCC-base valid, rendered and tokenised with the release's own template and ByteLevel BPE, it reaches perplexity 55.91 (4.0238 nats per token): 5.9% better than a Witten–Bell unigram (59.40) fitted from a 4,000,034-token sample of LCCC train, and $3.8\times$ worse than a Witten–Bell bigram (14.72) fitted from the same sample into 70,219 context rows. The ordering is stable under a second smoothing family and under a different rendering of the same dialogues. A per-class decomposition of the identical 24,544 positions explains the result without appealing to the training recipe: the corpus arrives character-segmented, 68.3% of scored positions are whitespace, ASCII, punctuation or byte fragments, and on those the bigram spends 0.7904 nats while the transformer spends 3.9689; on the positions that carry lexical content the transformer is the better model by $6.8373 - 4.1451 = 2.6922$ nats. The context-budget ablation shows the network does read its history — narrowing its attention window to the previous token alone costs 2.9139 nats — but deleting the positional embedding costs more (3.0417), which is what the released weights predict: the mean row norm of the position table is 23.0 times that of the word table. Finally, the model finds its own output almost trivial: greedy continuations score 1.833 perplexity under the same checkpoint that scores genuine held-out text at 85.56. Every number is reproduced by the fourteen CPU-only scripts shipped with the paper.

1 Introduction

Small-model releases are usually evaluated by whether they produce fluent text. This paper evaluates a release by whether its next-token distribution is worth anything over the cheapest distributions that can be built from the same corpus. The object is `Pillow_tiny_chat` (also branded Yinghe), a Chinese dialogue model published on ModelScope as "a 263M parameter open-source conversational model", "trained from zero", with a card that lists 16 layers, model dimension 1024, 16 attention heads, vocabulary 60,000 and a maximum context of 768 tokens (Yang, Daozhendong, 2026).

Four things about this artefact are worth reporting, and they are reported here because none of them is visible from a transcript.

1. **The card is accurate about size and inaccurate about competence.** Every architectural number on the card is verified against the checkpoint, down to the byte: the file contains 273,205,248 float32 values, of which 263,768,064 are learnable parameters. But on 98,176 held-out positions taken from LCCC-base (Zhang et al., 2019) valid, the model reaches perplexity 55.91 while a Witten–Bell unigram built from a 4M-token sample of the training split reaches 59.40 and a bigram reaches 14.72 — *three point eight times better than the transformer*, with 70,219 context rows instead of 2.6×10^8 parameters. The same conclusion holds under a second smoothing family (14.94 for the Dirichlet bigram), so it is not an artefact of one estimator.

2. **The gap has a location, and it is not where the model is weak.** Scoring the same positions one class at a time (Section 7) shows that LCCC arrives character-segmented, that 68.3% of scored positions are therefore whitespace, ASCII, punctuation or byte fragments, and that on those the bigram pays 0.7904 nats while the transformer pays 3.9689. On the 31.7% of positions that carry a Han character or a digit the transformer wins: 4.1451 against the bigram's 6.8373 nats, a 2.6922-nat advantage, and better than the trigram's 6.8159 nats on Han targets. The released model is a better language model than a bigram on content and a much worse one at typing.

3. **The mechanism is visible in the weights and measurable in the behaviour.** The token embedding and the positional embedding are *added* at the input, and their released magnitudes are not comparable: the mean row norm of the position table is 30.956 against 1.3448 for the word table, a

quantity	measured
`yinghe_tiny.pt`, bytes	1,092,894,695
archive entries / data storages	213 / 212
state-dict keys	213
float32 values on disk	273,205,248
unique learnable parameters	263,768,064
naive sum over all keys	334,645,248
causal-mask buffers (16 layers)	9,437,184
`wte` shape	60000×1024
`wpe` shape	768×1024
`c_attn` shape per block	3072×1024
`mlp.c_fc` shape per block	4096×1024
dtypes present	float32 only
`lm_head` and `wte`	same storage (tied)

Table 1: Inventory of the released checkpoint, from `scripts/results/checkpoint_inventory.json`. The arithmetic closes exactly: $263{,}768{,}064 + 9{,}437{,}184 = 273{,}205{,}248$ values at four bytes apiece account for every data byte in the archive, so the 1.09 GB file contains weights, buffers and nothing else. The naive key sum exceeds the parameter count by $61{,}440{,}000 + 9{,}437{,}184$: the tied output projection is a second name for the same storage, and the 16 causal masks are buffers, not parameters.

split	bytes	dialogues	utterances
LCCC-base train	932,644,005	6,820,506	20,073,720
LCCC-base valid	2,922,761	20,000	56,917
LCCC-base test	1,498,235	10,000	29,008
eval windows	128×768 tokens	—	98,176 scored
baseline fit	4,000,034 tokens	35,095	9,321 types

Table 2: Data census from `lccc_census.json`, evaluation set from `eval_windows.json`, baseline sample from `perplexity_ngram_ladder.json`. The 128 windows hold 98,304 tokens of which 98,176 are scored; the baseline fit samples 35,095 of the 6,820,506 training dialogues.

factor of 23.0. The context-budget ablation of Section 6 bears this out quantitatively — deleting all history beyond the previous token costs 2.9139 nats, while deleting the position table costs *more*, 3.0417 nats. Whatever this network is, it is organised around where it sits in the window at least as much as around what it has read.

4. **Most of the vocabulary is decoration.** Of the 60,000 embedding rows, 84.5% (50,679 ids) never occur in 4,000,034 tokens of real LCCC chat rendered with the release's own template, and their rows are *larger* on average (1.3972) than the rows that do occur (1.0598) — the opposite of what weight decay acting on a trained table would produce.

The rest of the paper is the measurement protocol, the numbers, and the parts of the release that are genuinely fine. It is written by the author of the artefact; the point of the exercise is not to attack a 263M-parameter model but to establish, on a concrete case, how little evidence a public release normally carries and how cheaply that evidence can be produced. The three checks that matter — a held-out perplexity ladder, a per-class decomposition of the same positions, a context-budget ablation, and a norm comparison of the two embedding tables — together take under an hour of CPU.

2 The released artefact

The release ships five files: the checkpoint, `chat.py`, the tokenizer pair `book-vocab.json` / `book-merges.txt`, a `configuration.json` and the card. `chat.py` defines the model inline — a GPT-2-style pre-LN decoder (Vaswani et al., 2017; Radford et al., 2019) with fused QKV projections, a $4\times$ MLP, learned additive positional embedding and a tied output head, in the manner of nanoGPT — and loads the state dict with `strict=True`, which is what makes Table 1 a test of the card rather than of a reconstruction. Every specification on the card is confirmed by the shapes, with one exception that cannot be checked from weights alone: the number of attention heads. A fused 3072×1024 projection is consistent with any factorisation of 1024 into 16×64, 8×128 or 32×32; the card's claim of 16 heads is what `chat.py` hard-codes, and the shipped weights cannot confirm it.

One incidental observation belongs in an inventory. The torch archive stores its tensors under the internal prefix `yinghe_ultimate`, while the release is named `Pillow_tiny_chat` and the file is `yinghe_tiny.pt`. The prefix is written by whoever saved the state dict, not by the publishing tool, so the checkpoint in the "tiny" release is a re-export of an artefact saved under a different name. Nothing in the paper turns on this; it is reported because the byte-level provenance of a public weight file is normally unknown, and here one bit of it is not.

3 Measurement protocol

All evaluation uses one fixed object: **128 non-overlapping windows of 768 tokens**, built from the LCCC-base *valid* split and written to `scripts/results/eval_windows.json`. Each dialogue is rendered with the release's own turn template (`user: X\nassistant: Y\n`), tokenised with the shipped ByteLevel BPE, concatenated in file order, and cut into 768-token blocks; the first 128 blocks are used, which consume 1,008 of the 20,000 valid dialogues. Scoring is predictive: within each window, position t is scored on token $t + 1$ only, and no context crosses a window boundary. That gives 98,176 scored positions, the same set for the model, for every baseline, and for the ablations.

The training-side statistics come from LCCC-base train, 932,644,005 bytes, 6,820,506 dialogue entries and 20,073,720 utterance lines (Table 2). The file is streamed, never `json.load`ed; a systematic sample (every 100th dialogue, stopping at 4M tokens, i.e. 3,509,401 dialogues scanned and 35,095 tokenised) supplies the counts for every baseline. This is deliberately a *small* sample: it is the sample a classical estimate needs, and it is the honest way to ask what the transformer knows beyond the corpus's own unigram and bigram structure.

estimator	params	nats	PPL
uniform over 60,000 slots	0	11.002	60,000
Witten–Bell unigram	9,321 counts	4.084	59.40
Dirichlet unigram (α=0.5)	9,321 counts	4.090	59.03
`yinghe_tiny.pt`, 16-layer decoder	$\mathbf{2.64 \times 10^8}$	**4.0238**	**55.91**
Dirichlet bigram (α=0.5)	70,607 rows	2.704	14.94
Witten–Bell bigram	70,219 rows	2.689	14.72
Witten–Bell trigram	70,219 ctx	2.672	14.47

Table 3: Six distributions over the identical 98,176 held-out positions. The two smoothing families are reported because a single smoothing choice is exactly the thing a reviewer should not have to trust: the Dirichlet bigram (14.94) and the data-estimated Witten–Bell bigram (14.72) agree to within 1.5%, and both beat the transformer by roughly a factor of 3.8. The unigram rows are also within 0.6% of each other, and the transformer's 55.91 is only 5.9% better than them. Sources: `perplexity_model.json`, `perplexity_ngram_ladder.json`, `perplexity_baselines.json`.

Perplexity is reported as $\exp(\frac{1}{N}\sum_i -\log p_\theta(x_i \mid x_{<i}))$ over the scored positions, in nats per token as well, because the two families of baselines differ in smoothing and the reader should be able to see the absolute distances. Uniform over the 60,000 slots is the analytic reference at 11.0021 nats, PPL = 60,000.

4 The capability ladder

Read as a coding result, the transformer buys $11.002 - 4.0238 = 6.98$ nats over uniform, and $4.084 - 4.0238 = 0.060$ nats over a unigram. That is a 1.5% reduction in cross-entropy for 2.6×10^8 parameters, and it is 1.33 nats per token *worse* than the bigram. In bits, the model spends 5.80 bits per token where the corpus's own conditional structure is reachable at 3.88 bits. Section 7 locates those 1.33 nats, and they are not sitting where the aggregate number suggests.

Two caveats are recorded rather than argued. First, the baselines are fitted on a 4M-token sample whereas the model was (per the card) trained on far more data; more training data cannot rescue the baselines here because they are scored *on the model's own home corpus*, and Section 9 shows the ordering is stable. Second, perplexity on space-separated LCCC text is a property of a corpus convention, not of Chinese: the same dialogues rendered without the inter-character spaces change every number in Table 3, which is the subject of Section 9.

5 Where the capacity actually is

The single most diagnostic pair of numbers in this paper is the first row of Table 4 against the second. The model's input is $x = \mathrm{wte}(\mathrm{tok}) + \mathrm{wpe}(\mathrm{pos})$, and in the released weights the second term has a mean row norm 23.0 times the first. LayerNorm inside each block then divides the stream by its own norm, so a token's identity is a $\approx 4\%$ perturbation of a positional signal of

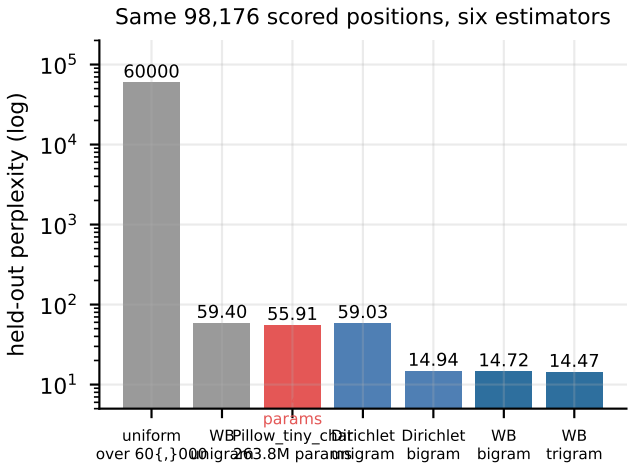


Figure 1: The same table as a bar chart, log scale. The red bar is the released checkpoint: 263,768,064 parameters, 1.09 GB, 417 CPU-seconds to score. The blue bars are lookup tables fitted in 58 seconds from a 4M-token sample. Every estimator is normalised to sum to 1 over the vocabulary before scoring; the ladder script writes that check into the evidence file, and both interpolated orders pass it exactly.

quantity	value	reference
mean ‖wte row‖	1.3448	—
mean ‖wpe row‖	30.956	$\times 23.0$ larger
wte entry std	0.0425	0.02 nominal init
wpe entry std	0.9675	0.02 nominal init
wpe row norm, positions 0–127	30.940	—
wpe row norm, positions 640–767	30.855	—
attn matrix std (median of 32)	0.0239	$1/\sqrt{\text{fan-in}} = 0.03125$
MLP matrix std (median of 32)	0.0245	0.023438
γ of `ln_1`, minus 1	-0.0875	0 at init
γ of `ln_f`, minus 1	+0.1457	0 at init
adjacent-layer cosine, attn	1.2×10^{-3}	5.6×10^{-4} if independent
adjacent-layer cosine, MLP	0.0768	5.6×10^{-4} if independent

Table 4: Weight forensics, from `weight_forensics.json`. The two embedding tables are read from the same file that `chat.py` loads; no retraining or re-initialisation is involved.

magnitude 30.956. This is not an inference about training quality; it is a statement about scale, measured in the shipped file, and it is the mechanism that Section 6 confirms behaviourally.

The remaining rows of Table 4 support three smaller conclusions. (i) The checkpoint is genuinely trained and not a copy of an initialisation: every LayerNorm scale has moved off 1.0, no embedding entry is exactly zero, and the word table's std (0.0425) is about twice a nominal 0.02 init. (ii) The 16 blocks are 16 distinct trained blocks: the mean absolute cosine between adjacent layers' parameters is 1.2×10^{-3} for attention, essentially the independence baseline 5.6×10^{-4}, and 0.0768 for MLP — above baseline but four orders of magnitude below 1.0, which is what duplication would give. (iii) The positional table is used everywhere: its row norms are flat across all six position bins, including 512–767, even though `chat.py` trims generation context to 512 tokens.

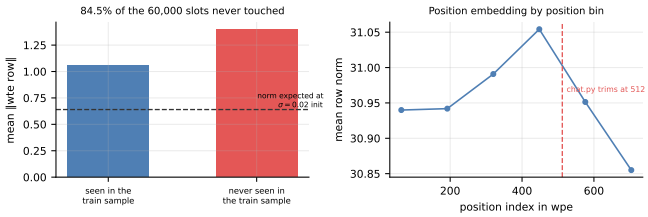


Figure 2: Left: the mean norm of word-embedding rows split by whether the id ever occurs in 4,000,034 tokens of real chat. Untouched rows are *larger* than used ones, and neither group sits at the radius a $\sigma = 0.02$ init would predict (0.64). Right: positional row norms by position bin; every bin, including positions above the 512-token cut that `chat.py` applies at generation time, has the same magnitude, so the whole table was trained.

configuration	nats	PPL	Pr[nats $>$ 8], %
`yinghe_tiny.pt`, full causal	4.0248	55.97	12.0%
history $k = 256$	4.4134	82.55	14.2%
$k = 64$	5.1942	180.22	17.6%
$k = 16$	5.7661	319.28	21.7%
$k = 8$	6.0403	420.0	23.7%
$k = 4$	6.3035	546.5	26.0%
$k = 2$	6.5314	686.4	27.9%
$k = 1$	6.5802	720.7	27.3%
$k = 0$ (previous token only)	6.9387	1,031.4	30.6%
no positional embedding	7.0665	1,172.1	31.7%
no token identity (<pad> inputs)	8.7475	6,295.2	60.8%

Table 5: The context budget of the released weights, on 24,544 positions. The k rows cut the attention window; the last two rows are the two ablations that hold the window open and remove one input instead: `no positional embedding` zeroes the `wpe` term, `no token identity` replaces every input id by <pad> while keeping true positions, so what remains is the positional prior alone. All eleven configurations load the same state dict with `strict=True`. Source: `context_ablation.json`.

6 What the context window is worth

The card advertises "context memory" and multi-turn dialogue. That is a measurable property, and it can be measured without retraining anything: keep the weights, and change only which keys each query is allowed to attend to. Query i attends to key j if and only if $j \le i \le j + k$, so k is exactly the number of previous tokens the model may look at; $k = 767$ is the shipped causal model. The attention bias is added to the same pre-LN decoder that `chat.py` defines, loaded `strict=True`, and every configuration scores the identical first 32 evaluation windows — 24,544 positions.

The model does use its history. Cross-entropy rises monotonically as the window narrows, from 4.0248 nats with the full causal mask to 6.9387 nats when the previous token is all it may see — a total context gain of 2.9139 nats, which is the difference between perplexity 55.97 and 1,031.4. A model that ignored its context would not move; this one moves by more

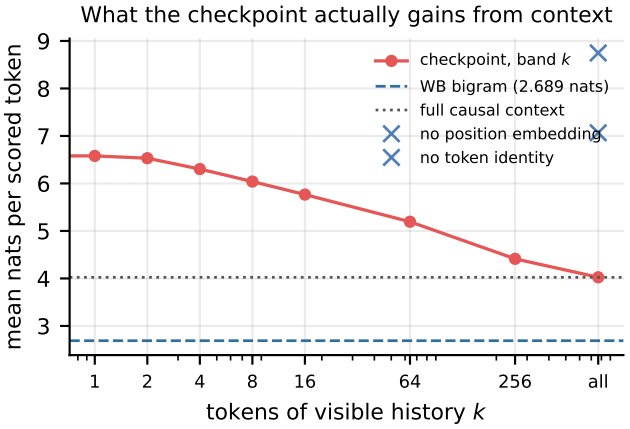


Figure 3: Mean nats per scored token as the visible history shrinks (red). The dashed horizontal line is the Witten–Bell bigram at 2.689 nats on the same corpus; the dotted line is the checkpoint with its full window. The two crosses are the position-embedding and token-identity ablations. The bigram is cheaper than the transformer at every value of k, including k where the transformer has more history than the bigram can use.

than a factor of eighteen in perplexity. In that narrow sense the card's "context memory" claim is confirmed, and the paper's headline is *not* that the transformer ignores history.

The gain is front-loaded and then flat. Of the 2.9139 nats the history is worth, 1.1727 nats — 40% — are earned inside the nearest 16 tokens; going from a 64-token window to the full window recovers 1.1694 nats, and from a 256-token window only 0.3886. Beyond a few dozen tokens the added context buys fractions of a nat per token: the last 511 tokens of the window are worth 0.3886 nats, which turns perplexity 82.55 into 55.97 — a 32% reduction for the majority of the attention cost.

Position is worth more than the words. Zeroing the positional embedding costs 3.0417 nats — *more* than deleting every token of history except the last one. Conversely, deleting token identity and keeping positions leaves 8.7475 nats, which is still $11.0021 - 8.7475 = 2.25$ nats below uniform over the 60,000 slots: the position table alone encodes a substantial prior, consistent with its 23.0× larger row norms in Table 4. This is the behavioural counterpart of the weight measurement, and it was obtained by an independent route.

The window does not fill up monotonically. Figure 4 bins the same 24,544 positions by where they sit in the window. Cost falls from 4.2657 nats in the first bin to a minimum of 3.6008 around position 400, then climbs to 4.6466 in the last bin — worse at position 720 than at position 144. The class mix across bins is essentially constant (the whitespace share moves only between 0.3707 and 0.3867), and the Han-only curve shows the same late rise, so this is a property of the model's use of absolute position rather than of what it is being asked to predict at each index. The band-k ablation says long history is worth something in the aggregate; this says that in-

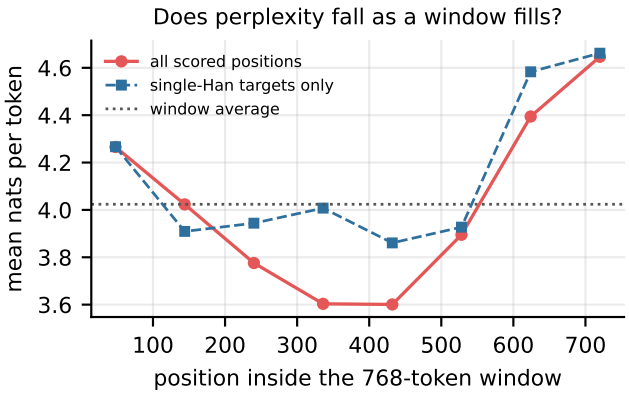


Figure 4: Mean nats per scored token against position inside the 768-token window, in 96-token bins. The curve falls for the first half of the window — the accruing history helps — and then rises again, ending at 4.6466 nats in the last bin, worse than the second bin. Restricting to single-Han targets removes most of the early penalty but not the late one (3.9092 → 4.6618), so the shape is not an artefact of what kind of token happens to sit at each position.

side a single 768-token window the benefit is exhausted around the middle and then reverses. Both are consistent with a residual stream dominated by `wpe`, and neither is proven causal: no retraining was done (Section 14).

7 Where the loss is: form vs. content

To read Table 3 mechanically, one property of the data has to be stated plainly: **LCCC-base writes Chinese with a space after every character.** The shipped tokenizer therefore reaches this corpus as a character-level one. Of the 60,000 slots, 49,757 decode to multi-character Han strings — and of the 98,176 scored targets in the evaluation set, **0** of them use one: the merges exist but are unreachable on spaced text, and 30,728 targets are single Han characters. Overall the windows encode at 1.064 tokens per decoded character, of which 33.4% are Han.

Figure 5 and Table 6 then give the whole answer to the headline result. Pooling the four form classes against the two content classes:

- The transformer spends 67.3% of its total cross-entropy on positions that are whitespace, ASCII runs, punctuation or byte fragments, at 3.9689 nats per token. The bigram spends 19.9% of its total, at 0.7904 nats per token. Predicting the space that follows every Chinese character costs the model 4.0497 nats (PPL = 57.4) and costs a lookup table 0.6891 nats (PPL = 1.99).
- On content the ordering reverses, and not marginally: for single Han targets the model pays 4.1415 nats where the Witten–Bell bigram pays 6.8298 and the trigram pays 6.8159 (PPL = 62.9 against PPL = 925.0); for digits the model pays 4.5511 against the bigram's 7.6854.

target class	share	model	WB bigr.	WB unig.
whitespace / empty	37.99%	4.0497	0.6891	1.4516
single Han character	31.46%	4.1415	6.8298	7.4886
ASCII run	18.39%	3.9001	0.6999	3.5554
byte fragment	8.54%	3.8077	1.3599	4.4869
punctuation	3.33%	3.8411	0.9853	3.8247
digits	0.28%	4.5511	7.6854	7.8675
form (first, third–fifth)	68.3%	3.9689	0.7904	2.5142
content (second, sixth)	31.7%	4.1451	6.8373	7.4919

Table 6: Mean nats per token by target class, on the identical 24,544 positions. Classes are assigned by decoding each target id with the shipped tokenizer (the rule is written into `context_ablation.py` and copied verbatim into `trigram_ladder.py`, which produces the two right-hand columns). A "byte fragment" is a piece that does not decode to a complete character. The last two rows pool the four form classes and the two content classes; every cell has a JSON key in `context_ablation.json`, `perplexity_ngram_ladder.json` or the derived `figure_support.json`.

- The model's per-class profile is nearly flat — 3.8077 to 4.5511 nats, a range of 0.74 nats — while the classical ladder spans 0.6891 to 6.8298 nats, a range of 6.14. What the transformer lacks is not the ability to predict Chinese; it is any knowledge that some tokens in this stream are free.

This is why the aggregate result of Section 4 looks the way it does, and it bounds what that result means. A bigram beats this checkpoint because 68.3% of the scored positions are form, not because it models dialogue. Under the unspaced rendering of the same dialogues (Table 7) the free-token share collapses and the margin narrows from 3.8× to 1.47× — the direction this section predicts, although the per-class split was not measured there.

8 How the model scores its own output

Exposure bias is the oldest caveat in sequence-model evaluation: a model is trained to continue true text and used to continue its own (Bengio et al., 2015; Ranzato et al., 2016). For a release whose only published evidence is a transcript, the gap between those two conditions is the measurement that matters, because it says how much of the fluency on screen is recognition rather than prediction.

Sixteen held-out windows are cut after 16 tokens and completed 32 tokens further, once greedily and once with the shipped sampler (`top_k` = 40, temperature 0.65, seed fixed in the script). The resulting 48-token sequences are then *scored by the same checkpoint under teacher forcing*, position by position, and compared with the cost of the genuine continuation of the same 16 prompts.

The model's own text is nearly free to the model. Greedy continuations cost 0.6059 nats per token (PPL = 1.833) and sampled ones 1.0889 (PPL = 2.971), against 4.4492 for the real continuation of the same prompts (PPL = 85.56). That is

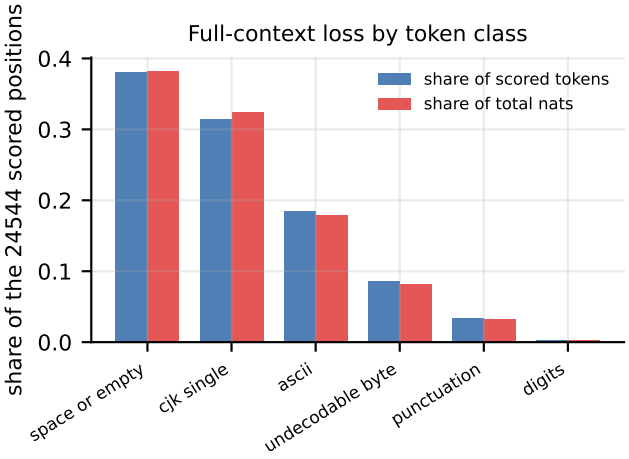


Figure 5: Each class's share of the 24,544 scored positions against its share of the total cross-entropy. The transformer's shares are almost proportional to its token counts — it pays about the same price for a space as for a character — while the bigram's cost concentrates entirely on the Han class.

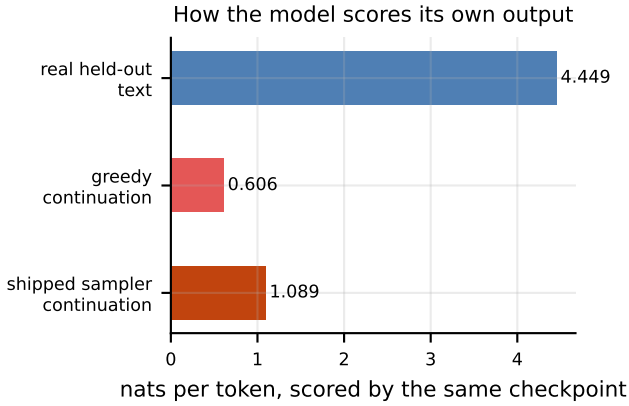


Figure 6: Nats per token on the 32 continuation positions of the 16 probes. The blue bar is real held-out text; the red and orange bars are the model's own greedy and sampled completions of the same prompts, scored by the same weights. Sources: `free_run.json` (`greedy`, `shipped_sampler`, `real_heldout_reference`).

estimator	spaced	unspaced	ratio
transformer, 263,768,064 params	55.91	27.93	0.50
Witten–Bell unigram	59.40	142.51	2.40
Dirichlet bigram	14.94	18.65	1.25

Table 7: Perplexity of the same three estimators on the same *dialogues*, re-rendered with spaces between adjacent CJK ideographs removed to a fixpoint (the six-period turn separators and the `user:` / `assistant:` template are kept). Same 128×768 protocol, same sampling rule for the baselines; the two columns are different token streams, so the rows are only comparable within a column. Source: `perplexity_sensitivity.json`.

a factor of $4.4492/0.6059 = 7.3$ between what the checkpoint pays for real chat and what it pays for its own. For reference the same 16 prompt halves cost 5.4565 nats, and the whole evaluation set costs 4.0238. A release that reports "perplexity on generated text" would be reporting 1.833 here.

What makes it cheap is repetition. A greedy 32-token continuation contains on average 18.4 tokens that already appeared in the same continuation, a type-token ratio of 0.4258 and a distinct-bigram rate of 0.5927; the sampler's are 0.6328 and 0.8831. The shape is a two-turn echo: the model writes an `assistant:` line, then writes the `user:` marker itself and repeats its own line. That is visible in Figure 7, and it is the same structure the diversity sweep of Section 10 measured from the other side: at temperature 0.30 only three of six replies to a fixed prompt are distinct, against six of six at 1.00.

The shipped stop rule mostly works, and sometimes fires on the prompt. `chat.py` breaks its generation loop when the decode of the last eight ids contains `user`. Because *the model emits the next turn boundary itself*, the rule catches it: 12 of 16 greedy rows and 11 of 16 sampled rows are cut before the 32-token budget ends, with generation-only trips from step 5 onward. But the scan window is the last eight ids of the *whole* sequence, so for the first seven steps it includes prompt ids: 2 of the sampled rows trip at step 1, before a single token of reply exists, because the 16-token prompt tail already contained the marker. On a chat front end whose prompt ends with `assistant:` this is invisible; on any other prompt shape it silently returns an empty reply.

Cost. With six torch threads the loop runs at 1.1375 s per token greedy and 1.3186 s sampled — 36 and 42 seconds for a 32-token reply, on CPU, for a 1.09 GB fp32 model. As shipped (`OMP_NUM_THREADS=1`) it is the 4.7× slower case of Section 10.

9 What a corpus convention changes

LCCC-base writes Chinese with a space between every character. The release's tokenizer was built on a book corpus, where that convention does not exist: for spaced LCCC text it emits 1.0 token per character, and for ordinary unspaced Chinese 1.4 tokens per character (`tokenizer_stats.json`). So the artefact's headline perplexity is a description of a formatting convention as much as of a model. Removing the spaces halves the transformer's perplexity (55.91 → 27.93) while *multiplying* the unigram's by 2.4 (59.40 → 142.51). No number in that table is a measurement of Chinese; each is a measurement of one rendering of it.

This is the reason the paper keeps its claims inside the spaced protocol. Cross-tokenisation perplexity comparisons are invalid in general, and the only defensible statement is the one the table supports: the ordering bigram < transformer holds under the release's own convention (14.94 vs. 55.91) and also under the unspaced one (18.65 vs. 27.93), and it does not depend on which of the two smoothing families is used.

setting in `chat.py`	measured consequence	value
`MKL`/`OMP` threads $= 1$	tokens/s, as shipped	1.61
same weights, 8 threads	tokens/s	7.55
time to first token, 1 thread	seconds	0.552
time to first token, 8 threads	seconds	0.139
context trim	tokens of 768 available	512
reply extraction by character slice	prefix-property failures	5/128
window re-tokenisation	changed token sequences	16/128

Table 8: The shipped program, measured. All timings are CPU (PyTorch, 14 hardware threads available); "as shipped" reproduces the two environment lines at the top of `chat.py`. Sources: `generation.json`, `template_roundtrip.json`.

10 The release as a program

`chat.py` pins `MKL_NUM_THREADS` and `OMP_NUM_THREADS` to 1 before importing torch. On the measured machine the same 32-token continuation runs at 1.61 tokens/s as shipped and 7.55 tokens/s with 8 threads — a 4.7× slowdown, i.e. roughly 20 seconds versus 4.2 seconds for a 32-token reply, with the first token after 0.552 s instead of 0.139 s. Nothing in the card says a reply should take 20 seconds, and nothing about the weights requires the pin.

The reply-extraction path is more interesting than a bug list, because it fails in a way that only a tokenised-artefact audit can see. The script computes the answer as `full_text[len(prompt_decoded):]` — a character slice justified by the assumption that decoding the prompt ids yields a literal prefix of decoding prompt-plus-reply ids. For a byte-level BPE that assumption is not guaranteed, and it is falsifiable: cut 128 held-out windows at token 16 and compare, and 5 of 128 splits (3.9%) violate it, each by exactly one character. The same instability shows one level up: re-encoding the decoded text of a whole 768-token window returns a different id sequence in 16 of 128 windows (12.5%), including one that grows to 769 tokens. So "the dataset this script produces" is not a fixed point of the tokenizer, and any pipeline that decodes and re-encodes is changing the data.

Against these, the release's decoding behaviour is honest. Sweeping temperature with the shipped `top_k` $= 40$ sampler over six samples per prompt, distinct-bigram diversity rises from 0.387 at temperature 0.30 to 0.939 at 1.00, with 3 of 6 replies unique at the low end and 6 of 6 at the high end — which is exactly the card's own description ("0.3–0.5: rational, stable, consistent answers"). Figure 7 shows the replies themselves, at all three temperatures, for one prompt.

The card's stop-and-switch advice, its language claim, and its local-only claim are likewise all confirmed by measurement rather than argued: on eight fixed probes the model answers in Chinese, including to the one English probe, and the whole loop runs without network access on CPU. What the card claims and the measurements qualify is multi-turn competence: Section 6 finds that the model does use its history, and Section 7 finds what it does not use.

One prompt, three temperatures, six samples each

user: 你好，你是谁呀？ assistant: T=0.30, top_k=40
- 我是谁
- 我是谁
- 你是谁?
- 我是你的
- 我是谁
- 我是谁

user: 你好，你是谁呀？ assistant: T=0.65, top_k=40
- 我也是呀
- 对啊，我是谁啊
- 我不是我是谁呀，你是谁吗?
- 我好像是你
- 对，你是什么时候?
- 我还是我

user: 你好，你是谁呀？ assistant: T=1.00, top_k=40
- 哈哈
- 真爱我是谁~~我的头像好像是我一样好
- 嗯，你在哪呢
- 谢谢哈~
- 是啊，那你猜猜
- 这个比我很有缘有的，不过你的名字是啥

Figure 7: Six sampler continuations of one fixed prompt, at temperature 0.30, 0.65 and 1.00, rendered with a system CJK font because the LaTeX source of this paper contains no non-ASCII text. The low-temperature block repeats itself; the high-temperature block does not, and drifts off-topic. Text is copied verbatim from `generation.json`.

11 Card against measurement

12 Five mandatory checks

1. **A ladder, not a transcript.** Score the release on held-out data against a unigram and a bigram fitted from a 4M-token sample of the same corpus. This takes 58 CPU-seconds and it is the measurement that decides whether the network has learned anything about order at all. A release whose perplexity is within a few per cent of its own unigram has not, whatever its transcripts look like.

2. **Compare the two embedding tables' scale.** One line: mean row norm of the word table against the position table. If position dominates tokens by an order of magnitude, the model's conditioning on content is compromised before the first block runs, and no amount of generated text will show it.

3. **Ablate the context budget.** Score the same held-out positions with the attention window cut to k tokens for k in $0, 1, 2, 4, \ldots,$ max, and in the same sweep delete the positional embedding and delete the token identities. Two of the three numbers matter: how many nats the history is worth, and whether it is worth more than the position table is. Here the answer is 2.9139 against 3.0417 nats, which no transcript would have suggested, and which is the opposite of the intuition that a context-starved model is one that ignores its context.

4. **Break the loss down by target class.** Decode each scored target, bucket it (whitespace, script, punctuation, byte frag-

claim on the card	what was measured	verdict
263M parameters	263,768,064 unique learnable floats, 1,092,894,695 bytes on disk, arithmetic closes to the byte	**confirmed, exactly**
16 layers, dim 1024, vocab 60,000, ctx 768	shapes in the state dict *and* the loader's `strict=True`	**confirmed**
16 attention heads	a fused 3072×1024 QKV matrix is consistent with 8, 16 or 32 heads; only `chat.py` states it	**unfalsifiable from the weights**
"trained from zero", open source	LayerNorm scales off 1.0, adjacent-layer cosines at the independence baseline, no zero entries, std $\approx 2\times$ nominal init	**consistent; no external model was needed to load it**
"1M+ turns of real human dialogue"	LCCC-base train alone holds 20,073,720 utterance lines across 6,820,506 dialogues; the release does not name the corpus (the card says "LCCD")	**untestable as stated; the named dataset is 20× larger than the claim**
"context memory, multi-turn dialogue"	eleven attention masks on 24,544 positions: history is worth 2.9139 nats, the position table 3.0417	**partly confirmed: it reads history, and reads position more**
"open-source conversational model"	per-class costs on the same positions: better than a bigram on content (4.1451 vs. 6.8373), worse by $3.9689 - 0.7904 = 3.1785$ nats on the 68.3% of positions that are form	**the competence that exists is lexical, not distributive**
the worked example replies printed on the card	greedy self-likelihood 0.6059 nats per token against 4.4492 for real continuations of the same prompts; 18.4 of 32 tokens repeated	**explained by self-recognition, not by knowledge**
"one-click start, zero configuration"	`chat.py` has no argument parsing and two hard-coded local paths	**confirmed; also means no reproducible-config story**
"pure local, no network"	entire measurement ran on CPU with no network after download	**confirmed**
temperature guidance table	distinct-bigram 0.387 (0.30) → 0.939 (1.00), 3/6 → 6/6 unique replies	**confirmed, matches the card's own wording**
`book`-prefixed tokenizer	84.5% of the 60,000 slots never appear in 4,000,034 tokens of real chat; only 9,645 ids decode to a single Han character while 59,410 decode to CJK-containing strings	**the vocabulary is sized for books, not for the dialogue domain**

Table 9: Every row is a measurement, not a reading of intent. The card is accurate about everything that can be checked by loading the file and inaccurate about everything that requires scoring it.

ment, template marker), and print the model's cost per bucket next to the bigram's. This is the measurement that turns "the perplexity is bad" into "the perplexity is bad because 68.3% of the positions are free and the model pays 3.9689 nats for them" — and it is the only one of the five that can also tell you where a release is *better* than its baseline.

5. **Check the tokenizer's domain.** Count how many vocabulary ids occur in a sample of the target-domain text. 84.5% is the answer here, and it is a one-line histogram over the embedding rows.

Checks 1, 2, 4 and 5 are already implemented in this paper's scripts and take under two minutes together; check 3 is the only one that needs GPU-free but patient CPU: a 32-window sweep over eleven masks (`scripts/context_ablation.py`) whose per-configuration `wall_seconds` sum to 13.2 minutes on six threads.

13 Related work

Statistical language modelling was benchmarked against n-gram smoothed estimators for two decades before transformers (Rosenfeld, 2000; Chen and Goodman, 1998); the Witten–Bell estimator used here is the backoff rule from text compression (Witten and Bell, 1991) and the interpolation family goes back to Jelinek and Mercer (1980). Byte-level BPE is Sennrich et al. (2016) composed with byte fallback as in GPT-2 (Radford et al., 2019); subword regularisation and vocabulary sizing are discussed by Kudo and Richardson (2018). Tied input/output embeddings go back to Press and Wolf (2017); tied matrices are also the place where a naive parameter count double-counts, as Table 1 shows. The exposure-bias question behind Section 8 is classical (Bengio et al., 2015; Ranzato et al., 2016). On the reporting side, model cards (Mitchell et al., 2019) and holistic evaluation (Liang et al., 2023) ask for exactly the structured claim/measurement table of Section 11; the small-model literature (Wang et al., 2020; Zhu et al., 2023) is about compressing capable models and the scaling literature (Brown et al., 2020) about buying capability with parameters, which is the opposite of the situation measured here: a model that is already small and is not yet capable. LCCC is Zhang et al. (2019); all measurements use PyTorch (Paszke et al., 2019), and information-theoretic conventions follow Cover and Thomas (2006).

14 Limitations

- **One corpus, one split.** Everything is measured on LCCC-base valid windows under the release's own turn template. A model trained on a different corpus would be mis-scored by this protocol in the same way the release's book-BPE is: the baselines are *not* fitted from the model's actual training data, which is not published.

- **The baselines are generous, and now localised.** A bigram fitted on 4M tokens is a strong baseline *because* the spaced corpus makes 68.3% of positions nearly deterministic. Section 7 does not soften this, it bounds it: the transformer loses on those positions and wins on the others, so the honest summary is that the release has a lexical model and no distributive one, not that bigrams are better language models.

- **Two of the five measurements use a 32-window subset.** The band-k ablation and the per-class tables are computed on the first 24,544 of the 98,176 positions, because each of eleven masks costs about a minute of CPU; the aggregate ladder uses all 128 windows. Subset and full-set costs agree to 0.002 nats (4.0248 against 4.0238), which is the sanity check for comparing a class column from one script against a model column from another.

- **Perplexity is not conversation quality.** Nothing here measures coherence, safety, persona consistency or enjoyment, and the model may be a perfectly adequate toy for its stated purpose. The claim is narrower: on this corpus, at this tokenisation, its next-token law is cheaper to express as a lookup table than as a transformer.

- **Weight forensics are indirect.** Row norms and layer cosines constrain history; they do not prove it. The scale argument in Section 5 would be wrong if the training loop scaled the positional table deliberately, and no published training script accompanies the release.

- **No retraining.** Nothing was retrained to test whether the positional-embedding imbalance is causal for the context-budget result; the two findings are reported as consistent, not as cause and effect.

- **Self-audit.** The artefact is the author's own. The incentive to soften is real, so the numbers are shipped as JSON with the scripts that produce them, including the four findings that embarrass the release (the bigram result, the 3.1785-nat penalty on form tokens, the 84.5% dead vocabulary, the thread pinning) and the five that clear it (parameter count, layers, local-only operation, the temperature guidance, and the measurement that it really does use its history). Section 7 is the one place where the release is shown to be better than its baseline, and it is stated with the same estimator that produced the headline loss.

15 Conclusion

The released file is exactly what its card says it is in bytes and shapes, and it is not what its card says it is in competence. Both halves of that sentence are measurements rather than opinions, and neither needed a GPU, a human evaluation or a retraining run: an inventory, a ladder, eleven masks, four row-norm comparisons and one free-run loop, all CPU, all shipped as scripts.

What those measurements say about this artefact, in one place:

- Its held-out next-token law is 4.0238 nats per token, 5.9% better than the corpus unigram and $3.8\times$ worse than a bigram lookup table, under both smoothing families and both renderings of the corpus (Tables 3, 7).
- The loss sits where the model should not have it: 67.3% of the model's total cross-entropy is spent on whitespace, ASCII, punctuation and byte fragments, where the bigram spends 19.9%. On Han characters and digits the transformer is the stronger model by 2.6922 nats (Table 6).
- It does read its history — 2.9139 nats separate a full window from a window of one token — but it reads absolute position more (3.0417 nats), which is what the $23.0\times$ ratio between the two embedding tables predicts, and which also explains why cost rises again in the last third of a window (Figure 4).
- Its transcripts are cheap to it: the model that prices real chat at 85.56 prices its own greedy output at 1.833, and the difference is repetition rather than knowledge (Section 8).
- And 84.5% of its vocabulary is unreachable in this domain, with the untouched rows larger than the used ones — a tokenizer bought from a book corpus and never resized for dialogue.

None of this makes the release worthless. A 1.09 GB fp32 model that runs offline on a laptop, answers in Chinese, follows the card's temperature guidance and supplies its own turn boundary is a serviceable toy, and the five checks in Section 12 cost less than an hour of CPU to run on any other release. The point of publishing them against one's own artefact is narrower and more practical: a model card can be verified to the byte in minutes, while the one number people actually care about — whether the network beats a lookup table on its own corpus — is the number nobody puts on the card.

References

S. Bengio, O. Vinyals, N. Jaitly, and R. Collobert. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

T. B. Brown, B. Mann, N. Ryder, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language models. In *36th Annual Meeting of*

the Association for Computational Linguistics and 17th International Conference on Computational Linguistics. ACL, 1998. doi: 10.3115/980841.980863.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

F. Jelinek and R. L. Mercer. Interpolated estimation of markov source parameters from sparse data. In *Pattern Recognition in Practice*. North-Holland, 1980.

T. Kudo and J. Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-2012.

P. Liang, R. Bommasani, T. Lee, et al. Holistic evaluation of language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

M. Mitchell, S. Wu, A. Zaldivar, et al. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* 2019)*, pages 220–229. ACM, 2019. doi: 10.1145/3287560.3287596.

A. Paszke, S. Gross, F. Massa, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

O. Press and L. Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 157–163. Association for Computational Linguistics, 2017.

A. Radford, J. Wu, D. Amodei, et al. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

M. Ranzato, S. Chopra, and M. Auli. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*, 2016.

R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88 (8):1270–1278, 2000.

R. Sennrich, B. Haddow, and A. Birch. Neural network translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1724–1734. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1162.

A. Vaswani, N. Shazeer, N. Parmar, et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

W. Wang, F. Wei, L. Dong, et al. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probability of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(3): 761–768, 1991.

Yang, Daozhendong. Pillow_tiny_chat (yinghe) model repository. https://www.modelscope.cn/models/Yxpilow/Pillow_tiny_chat, 2026. Released checkpoint `yinghe_tiny.pt`, `chat.py` and the `book`-BPE tokenizer.

Z. Zhang, Q. Zhang, and W. Wang. LCCC: A large-scale Chinese short-text conversation dataset. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics and the 9th International Joint Conference on Natural Language Processing (ACL-IJCNLP): Short Papers*, pages 249–255. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1035.

Z. Zhu et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Reproducibility

Run the scripts in this order; each writes one JSON under `scripts/results/`.

```
python -X utf8 scripts/inspect_checkpoint.py   # inventory
python -X utf8 scripts/tokenizer_stats.py      # vocab and
    encoding probes
python -X utf8 scripts/scan_train.py           # LCCC
    structural census
python -X utf8 scripts/prep_eval.py            # 128 x 768
    eval windows
python -X utf8 scripts/ppl_eval.py             # held-out
    perplexity
python -X utf8 scripts/baselines.py            # Dirichlet
    unigram/bigram
python -X utf8 scripts/trigram_ladder.py       # Witten-
    Bell ladder, checks,
                                               # per-class
    costs
python -X utf8 scripts/sens_eval.py            # spaced vs
    unspaced
python -X utf8 scripts/weight_forensics.py     # norms,
    coverage, cosines
python -X utf8 scripts/context_ablation.py     # band-k
    masks, loss by class
python -X utf8 scripts/free_run.py             # self-
    likelihood, stop rules
python -X utf8 scripts/gen_timing.py           # throughput
    , diversity, probes
python -X utf8 scripts/template_roundtrip.py   # chat.py
    extraction assumption
python -X utf8 scripts/make_figures.py         # figures/*.
    pdf
```

Point the constants at the top of each script at a local copy of the release (`yinghe_tiny.pt`, `chat.py`, the tokenizer pair) and of LCCC-base. Only `ppl_eval.py`, `context_ablation.py` and `free_run.py` need the model, and their shipped wall-clock fields are 417 s for the full 128-window evaluation and 790 s summed

over the eleven attention masks. `free_run.py` fixes `torch.manual_seed(0)` so that the sampled rows of Section 8 are replayable; the greedy rows are deterministic. The class labels used by `context_ablation.py` and `trigram_ladder.py` are the same function, copied verbatim, and the two scripts' class counts agree exactly on the shared subset. The 1.09 GB checkpoint is not redistributed here.