Onward to the Hutter Prize

A token-level transformer compresses enwik9 to 117 MB on one CPU core

Aldric Pierrain
Cydral Technology, Paris, France
contact@cydral-technology.eu
15 November 2026

Abstract. This report describes a lossless compressor built on a small transformer language model and measured on enwik9, the gigabyte of Wikipedia text of the Hutter Prize. A network of 10 million parameters, trained in one day on the GPU of a laptop and stored on 6 bits per weight, predicts each next token among 8,192. Its distribution is split exactly into binary decisions, which 107 context models correct while they learn, and an arithmetic coder writes them. One CPU core compresses the file in 24 h 48 min with 6 GiB of memory, to 117,169,624 bytes, 0.937 bits per byte; archive and decompressor weigh 124.1 MB. The record of the prize is 100.4 MB. The report gives the design, the measurements and the tests behind them, the distance to the record and what each next step is worth.

1 Introduction

The Hutter Prize [1] rewards the programs that compress enwik9, the first 10^9 bytes of an English Wikipedia dump, into the smallest total of archive and program. A program runs on one CPU core, without a GPU, in less than 10 GB of memory and 70,000/T hours, T being the Geekbench 5 score of the machine: about 50 hours on the machines of the committee. The record is 100,424,672 bytes. It dates from July 2026, when fx2-cmix-transformer replaced the LSTM of the cmix model bank [7] by a transformer trained beforehand [2], and the entries now listed above it on the Large Text Compression Benchmark [5] follow the same pattern.

The program described here reaches that pattern from the other side. It is an example of Dlib-Transformer-extensions [3], a C++14 library that adds transformer layers to Dlib [4], and it was written to show what the building blocks of the library make together: a tokenizer, a network trained on a GPU, quantized weights and an inference engine for one core. The network is the main predictor, and the context models are added to it. A byte-level version dates from September 2026. The token-level program was rebuilt from 19 October, measured on the whole of enwik9 on 26 October (124.3 MB) and again from 2 November (117.2 MB).

Counted as the prize counts, the total of 2 November is 131.0 MB, 30.6 MB above the record. This report is therefore not a claim. Its title names a direction, and Section 7 measures the distance left. Sections 2 to 4 describe the design, Section 5 gives the measurements and the isolated tests behind them, Section 6 places the result among the published ones.

2 The chain

Figure 1 shows the chain from the corpus to the archive, and the way back.

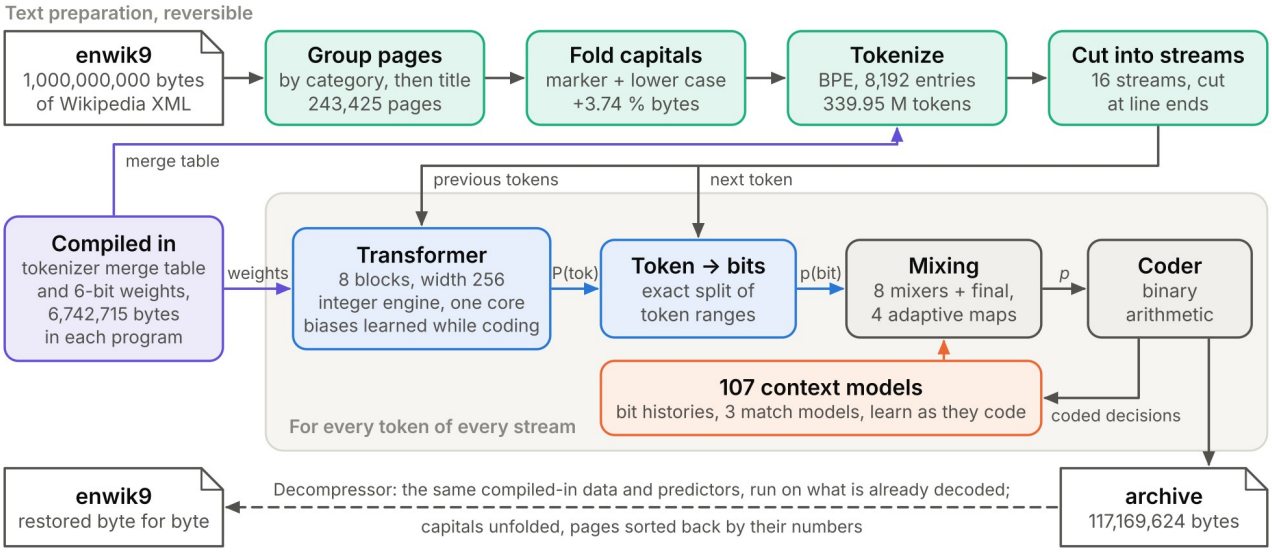


Figure 1. From the corpus to the archive and back. Green: reversible preparation of the text. Blue: the network and the split of its distribution. Orange: the context models. Violet: the data compiled into the programs.

Three reversible steps prepare the text. The 243,425 pages of the dump are grouped by their first category, then by title; the decoder sorts them back by their page numbers, so the order costs nothing in the archive. Capitals are written as a marker followed by the lower-case letter, which lets one token serve *The* and *the* and lengthens the text by 3.74 %. A byte-pair

tokenizer of 8,192 entries, cut where the class of the characters changes, turns the gigabyte into 340 million tokens of 2.94 bytes on average; the program learns its merges from the corpus in about 150 seconds. The tokens are cut at line ends into 16 streams that advance together.

For each token, the network gives a distribution over the vocabulary from the previous tokens of the stream, up to 256. Section 4 explains how this distribution becomes binary decisions, each corrected by context models and mixers before a binary arithmetic coder writes it. The decoder holds the same network and rebuilds the same predictions from what it has decoded.

Two things travel with the program, the merge table of the tokenizer and the quantized weights: 6,742,715 bytes compiled into the executable. The rest (context models, mixers, one bias per token and per stream) starts empty and learns while coding, identically on both sides. The archive holds the coded decisions and a CRC-32 of the input.

3 The token model

Network. Eight pre-norm blocks of width 256 (Figure 2, top). Each block applies an RMS normalisation, a causal attention with four query heads over two key-value heads and rotary positions, a second normalisation and a SwiGLU layer of 8/3 of the width. A final normalisation and a linear head score the 8,192 tokens. The network has 9,982,880 parameters, of which 2.1 million form the input table and 2.1 million the head. One token costs about 8.9 million multiply-adds, 2.1 million of them in the head. Predicting tokens in place of bytes divides the number of steps of the network by 2.94, which is what lets a network of this size run on one core within the time of the prize.

Training. The network is trained on enwik9 itself, since its weights are counted with the program: 48,114 steps of 64 windows of 256 tokens with AdamW, a peak rate of 0.0014 and a cosine schedule laid out for 60,000 steps. This makes 2.3 passes over the corpus in about 24 hours on the GPU of a laptop (RTX 5070 Ti). The loss at the stop was 2.15 to 2.19 nats per token, 1.05 to 1.07 bits per byte.

Quantization. Each weight is stored on 6 bits with one scale per output row, the clipping point of each row being the one of least squared error; the relative error on the weights is 3.0 %. The program codes the quantized weights with a context model of its own, at 5.4 bits per parameter.

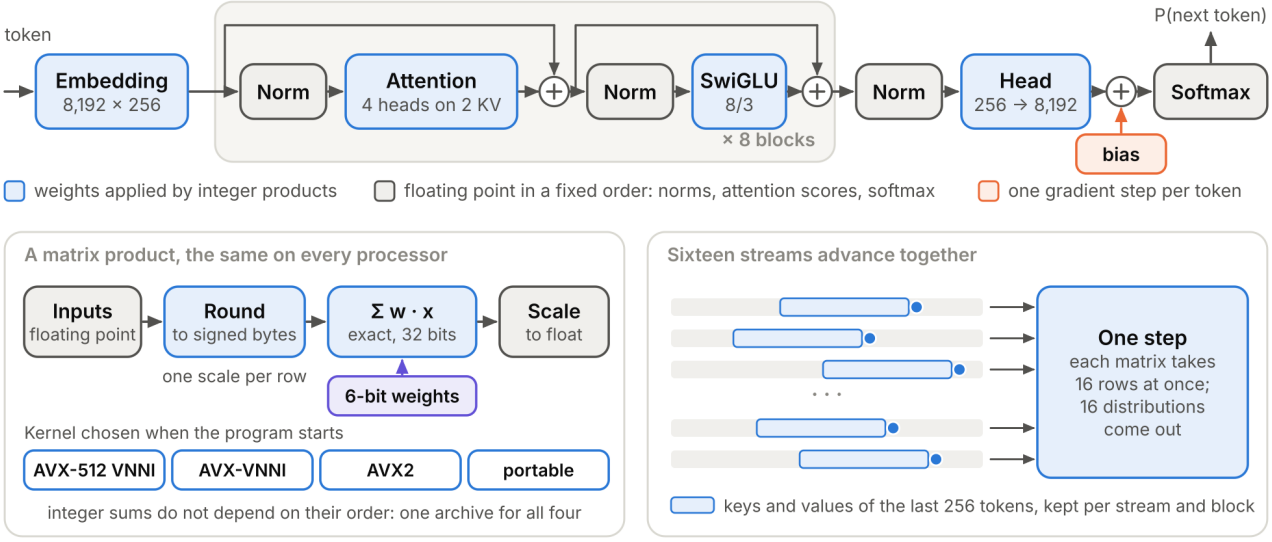


Figure 2. One step of the network for one token (top), a matrix product of the inference engine (bottom left) and the streams sharing each product (bottom right).

Inference. A decoder has to reproduce the predictions of the encoder bit for bit, on another processor. The matrix products therefore run on integers (Figure 2, bottom left): the inputs of a product are rounded to signed bytes with one scale per row, multiplied by the weights and summed exactly in 32 bits. An integer sum does not depend on the order of its terms, so the four kernels the program chooses from when it starts (AVX-512 VNNI, AVX-VNNI, AVX2, portable C++) write the same archive. Normalisations, attention scores and the softmax stay in floating point, written so that the vector code performs the operations of the scalar code one for one. Rounding the inputs to bytes costs 0.1 % of the archive. The 16 streams share every product (Figure 2, bottom right), and the keys and values of the last 256 tokens are kept per stream and per block.

4 From token probabilities to coded bits

An exact split. A token model and a binary coder meet through the order of the vocabulary. Its entries are sorted by their bytes, so the tokens that share a prefix form one contiguous range, whose probability is a difference of two cumulative sums. A token is then coded along its bytes (Figure 3, left): each bit of a byte splits the current range in two, and after a byte

that completes an entry, one more decision says whether the token ends there. The conditional probabilities multiply back to the probability the network gave to the token, so the split loses nothing, and a decision with a single possible outcome is not coded. Every decision is binary, which is the form context mixing works on [6].

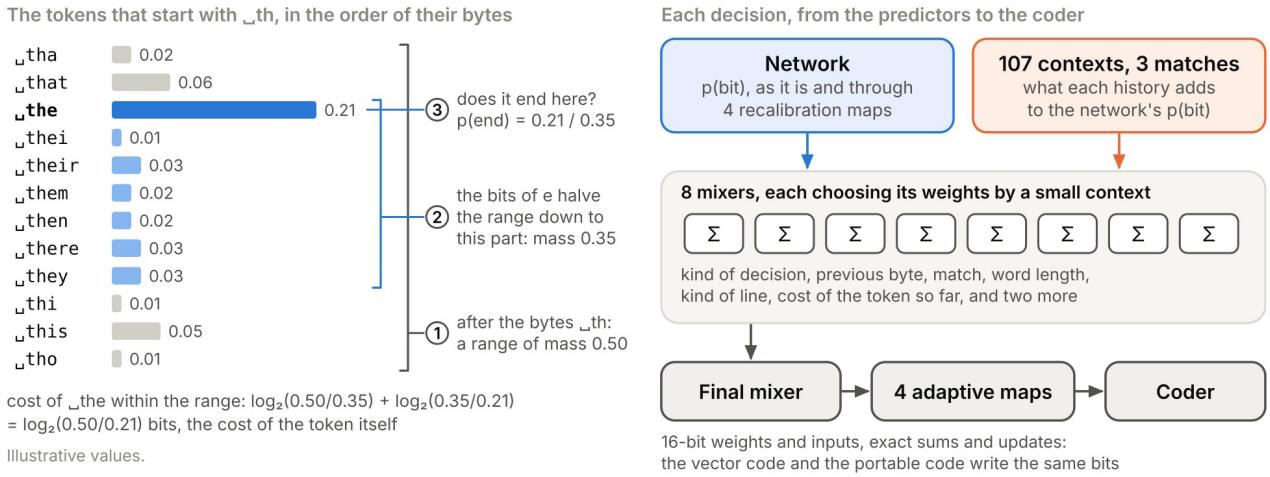


Figure 3. Left: a token coded as binary decisions on ranges of the sorted vocabulary. Right: the path of one decision from the predictors to the coder.

Context models. 107 contexts follow the text byte by byte: orders 1 to 32, sparse and word contexts, the line above, the state of the wiki markup (links, templates, tables, entities), numbers, and the previous tokens with the token the network holds most likely. Each context keeps a one-byte history per decision, drawn from 205 states. A learned map turns the history into what it adds to the probability of the network, and adds nothing for a history that has seen nothing: a context speaks where it knows something the network does not. Three match models follow the most recent repeats of the last 6, 12 and 24 bytes across all streams.

Mixing. The corrections enter eight mixers, each choosing its weights by a small context such as the previous byte or what the token has cost so far; a final mixer combines them and four adaptive probability maps refine the result (Figure 3, right). Weights and inputs are 16-bit integers with exact sums, for the reason given in Section 3.

Learning in the network. Each stream also holds one bias per token, added to the scores of the head and moved by one gradient step after each coded token, so that the distribution follows the vocabulary of the pages being coded. It saves about 1 %. Other additions were measured and left out, their gain being too small for their time: a cache over recent hidden states (2.0 % on the network alone, 0.08 % in the full coder), an update of the weights while coding (0.5 to 0.7 % for twice the time), a further match model (0.06 to 0.08 %) and 93 further contexts (no gain).

5 Measurements

Table 1 gives the run of 2 November: the program of the repository with its default options, on one core of an Intel Core Ultra 9 275HX under Linux (WSL2), with the AVX-VNNI kernels the program chose. The decompressed file is identical to enwik9. The Geekbench 5 single-core score published for this processor, about 2,200, remains to be measured on the machine of the run. It puts the limit of the prize at 31.8 hours, of which each direction takes 78 to 79 %.

Table 1. Compression and decompression of enwik9, run of 2 November 2026.

Quantity	Value	Detail
Archive	**117,169,624 bytes**	**0.937 bits per byte, 2.76 bits per token**
Decompressor	7,063,888 bytes	6,889,628 zipped, on a rebuild of the same source
Program that compresses and decompresses	7,129,424 bytes	6,921,345 zipped
Tokenizer and weights compiled into each program	6,742,715 bytes	5.4 bits per parameter
Compression	24 h 48 min 11 s	5.97 GiB at most, 89.3 µs per byte
Decompression	25 h 11 min 57 s	5.09 GiB at most, output identical to enwik9
Benchmark count: archive and zipped decompressor	124,059,252 bytes	26th among the 234 entries of [5]
Prize count: archive and twice the zipped program	131,012,314 bytes	record: 100,424,672 bytes

Isolated tests support these figures. Before the run, a projection from samples spread over the file gave 118.8 MB; the run gave 117.2 MB, 1.3 % lower. The run of the previous engine, 30 h 18 min on the same processor, had written the same archive. On a second processor, a 2.1 GHz Xeon, the AVX-512 VNNI and the AVX2 kernels wrote the same archive of a sample of 1 MiB, the AVX-512 VNNI kernels and the portable code wrote the same archive of a sample of 256 KiB, and the AVX-512 VNNI kernels decoded what the portable code had written. For this report, the published program also coded ten blocks of 4 MiB, one in the middle of each tenth of the file, each on its own and on that second processor (Table 2). Coded apart, each block starts with empty context models and four streams, which is why the ten blocks cost 0.995 bits per

byte together against 0.937 for the whole file. Block 6, pages on towns generated from census data, costs a third of the others; it was decompressed and compared with its source.

Table 2. Bits per byte on ten blocks of 4 MiB of enwik9, one in the middle of each tenth of the file, coded separately.

Block	1	2	3	4	5	6	7	8	9	10	All
Network alone	1.172	1.158	1.021	1.124	1.082	0.348	1.075	1.092	1.063	1.039	1.018
With the context models	1.152	1.140	0.992	1.105	1.057	0.334	1.053	1.064	1.036	1.016	0.995
Gain of the context models	1.7 %	1.6 %	2.9 %	1.7 %	2.2 %	3.9 %	2.1 %	2.6 %	2.6 %	2.2 %	2.2 %

The same ten blocks follow the program through its steps (Table 3). In ten days the cost of the blocks fell by 7.8 %, and the archive of the whole file by 7.1 MB between its two runs.

Table 3. The path of the program, on the ten blocks of Table 2 and on the whole file. Sizes in bytes.

2026	State of the program	Ten blocks	enwik9	With the decompressor	Compression
1 Sept.	Byte-level predictive compressor, an example of the library				
19 Oct.	Token level: tokenizer and network compiled in, exact split into binary decisions				
23 Oct.	Four blocks of width 256 on 6 bits, tokenizer cut at spaces	5,658,750			
26 Oct.	Tokenizer cut at character classes, folded capitals, bit histories, biases learned while coding, integer products	5,420,299	124,291,313	129,376,833	14 h 43
1 Nov.	Eight blocks	5,226,291			
2 Nov.	107 contexts as corrections of the network, pages grouped, kernels chosen at run time	5,216,107	117,169,624	124,233,512	24 h 48

Table 4 gathers what each part of the design is worth, each measured with the others in place on samples of encyclopaedic text. Two of its lines matter for what follows: the weights can lose a bit for 18 % of their size, and one token table can serve the input and the head.

Table 4. Isolated measurements of the parts. Negative values are bytes saved.

Part	Measured on	Archive	Program or time
107 context models on top of the network	first 300 MB	−4.3 %	28 µs per byte on the Xeon
107 contexts in place of the 20 of October	first 100 MB	−0.9 %	same memory
Bias per token learned while coding	encyclopaedic text	about −1 %	no weight added
Eight blocks in place of four	the ten blocks of Table 2	−3.6 %	weights +1.9 MB, time × 1.45
Cut at character classes, folded capitals	16 MB, two blocks of width 128	−1.8 %	15 % more tokens per byte
Pages grouped by category	first 100 MB	−0.11 %	nothing stored
Inputs of the products rounded to bytes	encyclopaedic text	+0.1 %	same archive on every processor
5 bits per weight in place of 6	two samples of 2 MB	+1.3 %	weights −18 %
4 bits per weight in place of 6	two samples of 2 MB	+7.9 %	weights −34 %
Vocabulary of 4,096 in place of 8,192	4 MB, four blocks of width 128	+2.6 %	weights −29 %, 9 % more tokens
One token table for the input and the head	4 MB, four blocks of width 128	−1.8 %	weights −36 %, −22 % at full size
AVX-512 VNNI kernels in place of AVX2	16 MB in 16 streams	same archive	2,950 against 2,550 tokens/s

6 Position

The Large Text Compression Benchmark [5] ranks programs by the size of enwik9 and of the zipped decompressor, without limits on time or memory. With 124,059,252 bytes, this program would stand 26th among its 234 entries (Figure 4, left). The 25 entries ahead are cmix and its descendants, phda9, gmix and five neural coders that rely on a GPU or on more than 40 GB of memory; the next one behind is paq8px, with 125.1 MB in 81 hours and 28 GB.

The same total would have led that benchmark from its start until 8 April 2016, when cmix v9 reached 124,036,309 bytes (Figure 4, right); the transformer architecture was published the following year [9]. Counted as the prize counts, with the program taken twice, the total has never been below the record, which started at 116.7 MB when the prize moved to enwik9. Entries awaiting validation now go down to 95.9 MB [5].

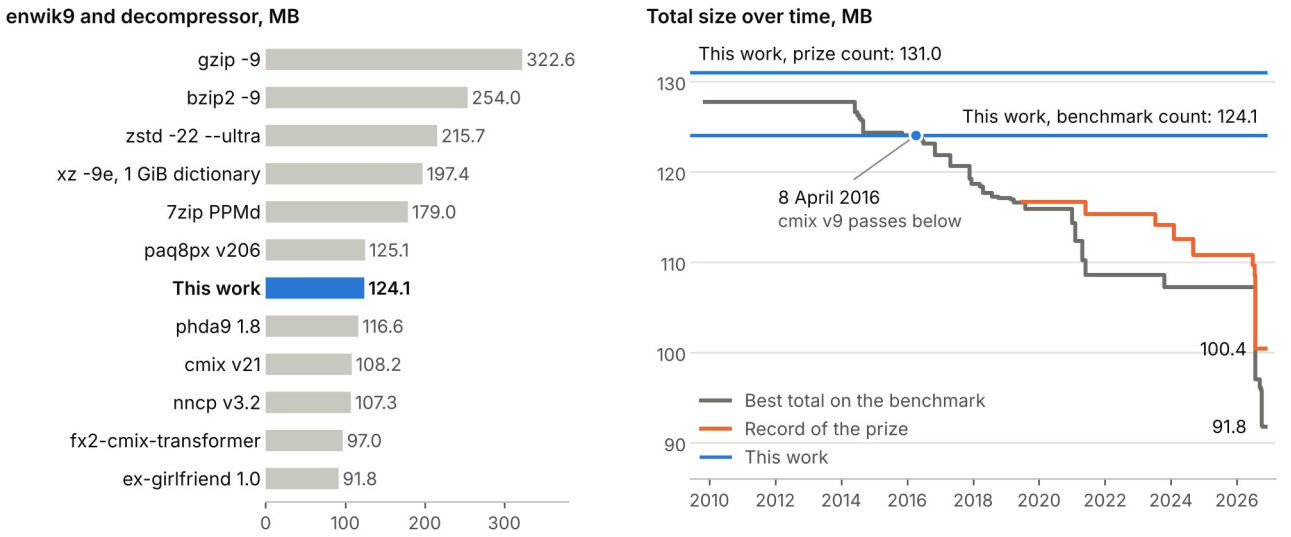


Figure 4. Left: enwik9 and decompressor for reference programs, as listed in [5]. Right: the best total of [5] and the record of the prize for enwik9 over time, with the two totals of this work.

Table 5 sets the program beside the record entry and two references. The record pairs the model bank of cmix with a byte-level transformer of 6 million parameters, trained on eight RTX 5090 GPUs for 26 hours and stored on 4 bits per weight [5]. Of the 30.6 MB between the two totals, 7.0 MB are program bytes (2 × 6.9 MB here, 2 × 3.4 MB there) and 23.6 MB are archive payload, 117.2 MB against about 93.6 MB. The two programs run at comparable speeds: by the ratio of the Geekbench scores, the 24.8 hours measured here become about 42 hours on the machine where the record takes 37.

Table 5. This work beside the record entry and two references. Sizes in bytes, times and memory as measured by each test on its own machine; data of [1] and [5].

	This work	**fx2-cmix-transformer [2]**	**cmix v21 [7]**	**nncp v3.2 [8]**
Date	2 November 2026	24 July 2026	17 September 2024	23 October 2023
Predictors	token transformer, 107 context models	cmix bank, byte transformer	cmix bank, LSTM	transformer trained while coding
Network shipped	10.0 M parameters, 6 bits, 6.7 MB	6 M parameters, 4 bits, 2.9 MB	none	none
Training beforehand	one laptop GPU, 24 h	eight RTX 5090, 26 h	none	none
Machine when coding	one core, 6.0 GiB	one core, 9.7 GB	one core, 31 GB	GPU, 7.6 GB
Compression time	24.8 h	37.2 h	173 h	67 h
enwik9 and decompressor	124,059,252	96,996,198	108,244,767	107,261,318
Prize count	131,012,314	100,424,672	outside the limits	outside the limits

What the measurement shows is on the left of Figure 4. The predictor is a language model in the ordinary sense: it was trained on the next-token objective by the code that trains the other models of the library, and nothing in it is specific to compression. With 10 million parameters, a day of training on a laptop and one core running integer products, it codes Wikipedia text in 0.94 bits per byte, where the general-purpose compressors need 1.4 to 2.6 bits and where the context mixers of 2015 needed four days and 30 GB for the same total. A better model of the text gives a shorter archive, which is the premise of the prize, and the measure holds for a small model as it does for a large one.

7 Onward

Between its two full runs the archive lost 7 MB in a week. At that pace the record would fall in early December, and Table 6 explains why this report does not announce it. Three facts order the next steps. The program weighs 6.9 MB and the prize counts it twice. The training stopped at 80 % of its schedule. The second stage holds 107 contexts where the record entries carry the whole bank of cmix. Table 6 prices the corresponding levers from the measurements of Table 4 and from the notes of the repository; each line assumes the previous ones. None of these levers is in the published code.

Table 6. Size levers and their projected effect, in MB. The last three lines are scenarios for the second stage.

Lever	Archive	Program, each copy	Prize count	Benchmark count	Basis
Measured, Table 1	**117.17**	**6.92**	**131.01**	**124.06**	**run of 2 November**
One token table for the input and the head, trained as long as the published network	−1.17	−1.48	−4.14	−2.66	Table 4: −1.8 % at reduced size, taken as −1 %; weights −22 %
5 bits per weight, the quantization placed inside the training	+0.70	−0.95	−1.20	−0.25	Table 4: +1.3 % without it, taken as +0.6 %; weights −18 %
Training to the end of its schedule, then twice the passes	−1.75	0	−1.75	−1.75	−1.2 to −2.0 % per doubling on 64 MB, taken as −1.5 %
After the three levers of the network	**114.94**	**4.49**	**123.93**	**119.40**	**sum of the lines above**
Second stage closing a quarter of the payload gap to the record entry	109.60	4.49	118.58	114.06	scenario; the payload of the record entry is 93.6 MB
Second stage closing half of the payload gap to the record entry	104.26	4.49	113.24	108.72	scenario
Second stage closing the whole of the payload gap to the record entry	93.57	4.49	102.55	98.03	scenario

The three levers of the network rest on measurements, and the record entries already train their weights for 4 bits [5]. Together they bring the prize count to 123.9 MB and the benchmark count to 119.4 MB. The second stage is the large lever and the least certain one. The record entries show what a full bank reaches around a network of 6 million parameters, and the split of Section 4 feeds any binary mixer; the gain of such a bank around this network is not measured, hence the scenarios. They also state the arithmetic of the prize: with the whole payload gap closed, the count would stay 2.1 MB above the record, since each copy of the program would weigh 4.5 MB against 3.4 MB. Passing the record takes a second stage of that class, a network near 3 MB, and the time to run both.

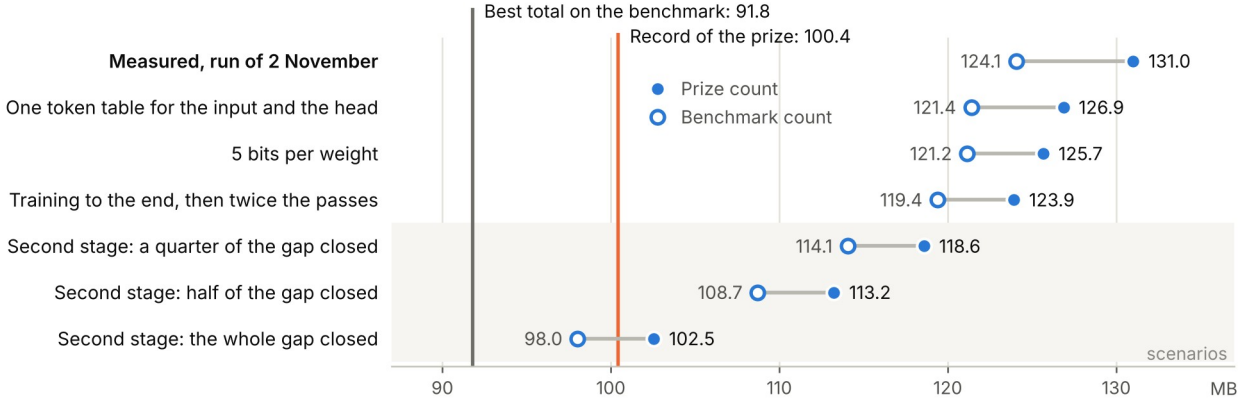


Figure 5. The two counts of the program along the lines of Table 6. Each line adds its lever to the previous ones; the shaded lines are the scenarios for the second stage.

Table 7. Speed levers and their projected effect. The shares of time come from a profile of the published program on 2 MiB in 16 streams (2.1 GHz Xeon, AVX-512 VNNI kernels).

Lever	Part of the time concerned	Projected time	Basis
Measured, Table 1	**whole run**	**24.8 h**	**78 % of the time allowed**
Attention on bytes: keys and values of the window kept as signed bytes, scores and sums by the integer kernels	attention in floating point: 20 %	−12 %	profile; four times as many terms per instruction
Head evaluated along the byte tree, in two levels, in place of 8,192 scores per token	products of the head: 12 %	−9 %	profile; 2.1 of 7.9 M products per token; needs a training
Context tables read one decision ahead	context models: 22 %	−5 %	profile; estimate of the waits on memory
After the three levers	**whole run**	**18.4 h**	**58 % of the time allowed**

The profile shows where the time goes: 45 % in the integer products, 20 % in the attention over the window, which still runs in floating point, and 22 % in the context models. The three levers of Table 7 would bring the run from 24.8 to about 18 hours, 58 % of the time the prize allows on the measured processor. The 13 hours then free are the budget of a broader second stage.

8 Availability

The program is one C++14 source of 5,147 lines, `examples/slm_tok_predictive_compressor_ex.cpp`, in the repository [3] under the Boost Software License 1.0, with the header that holds the tokenizer and the network. It needs no library other than the headers of the repository. From the root of the repository, the first command below builds the decompressor, and `-DSLM_CODEC_ONLY=1` in its place builds the program that does both; the next two reproduce Table 1. Builds with other compiler options write the same archives, provided that a build enabling FMA for the whole program also passes `-ffp-contract=off`. The full program also builds the tokenizer (`--build-tokenizer`) and trains the network (`--train`) with the options given in Section 3.

```
g++ -std=c++14 -O3 -s -DSLM_DECOMPRESSOR_ONLY=1 -I dlib examples/slm_tok_predictive_compressor_ex.cpp \
    -o decomp9
./codec --compress --input enwik9 --output archive9
./decomp9 --input archive9 --output data9
```

Acknowledgements. The second stage of this program applies the ideas of the PAQ family, and its position is measured on the Large Text Compression Benchmark, both by Matt Mahoney; the library builds on Dlib by Davis King.

References

[1] M. Hutter. 500’000€ Prize for Compressing Human Knowledge. http://prize.hutter1.net, rules at http://prize.hutter1.net/hrules.htm. Consulted on 15 November 2026.

[2] V. Ivanov. fx2-cmix-transformer, entry of 24 July 2026 to the prize of [1], as described in [5].

[3] A. Pierrain. Dlib-Transformer-extensions. https://github.com/Cydral/Dlib-Transformer-extensions.

[4] D. E. King. Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research 10, 2009, 1755–1758.

[5] M. Mahoney. Large Text Compression Benchmark. http://mattmahoney.net/dc/text.html. Copy of 14 October 2026.

[6] M. Mahoney. Adaptive Weighing of Context Models for Lossless Data Compression. Florida Institute of Technology, Technical Report CS-2005-16, 2005.

[7] B. Knoll. cmix. https://www.byronknoll.com/cmix.html.

[8] F. Bellard. NNCP v2: Lossless Data Compression with Transformer. 2021. https://bellard.org/nncp/.

[9] A. Vaswani et al. Attention Is All You Need. Advances in Neural Information Processing Systems 30, 2017.