Nacrith: Neural Arithmetic Compression for State-of-the-Art Lossless Text Encoding

Roberto Tacconelli
tacconelli.rob@gmail.com

Abstract

We present **Nacrith**, a lossless text compression system that pairs a 135-million-parameter transformer language model (SmolLM2-135M) with an arithmetic coder. By exploiting the deep linguistic structure captured by the neural model -- grammar, semantics, and long-range context -- Nacrith achieves compression ratios of approximately **14-15%** on English prose, roughly **2.5x better than gzip** and **2.3x better than xz**. On a 100 KB benchmark file, Nacrith compresses to **1.24 bits/byte**, which is 74% below the 0th-order Shannon entropy limit, 65% below the 1st-order limit, and 55% below the 2nd-order limit. Both compressor and decompressor run the exact same model with identical weights, guaranteeing perfect lossless reconstruction. We describe the system architecture, the arithmetic coding pipeline, the KV-cache acceleration strategy with chunked sliding window, the compressed file format, and provide comprehensive benchmark results against traditional compressors.

---

1. Introduction

Shannon's foundational work (1948) established that compression is fundamentally equivalent to prediction: a model that assigns high probability to the next symbol in a sequence enables an encoder to represent that symbol with fewer bits. Traditional compressors such as gzip (DEFLATE), xz (LZMA2), and zip exploit this principle through dictionary-based pattern matching on raw bytes within a sliding window. While effective for local, literal repetitions, these methods are blind to higher-order linguistic structure --grammar, semantics, and long-range dependencies.

Recent advances in neural language models offer a qualitatively different approach. A transformer model trained on large text corpora can predict the next token with high confidence by leveraging deep linguistic knowledge. For instance, after the context *"The President of the United"*, the model assigns overwhelming probability to *"States"* --even if that phrase has not appeared recently in the input. This deep predictive capability directly translates to superior compression when paired with arithmetic coding.

Nacrith implements this insight by combining SmolLM2-135M, a 135-million-parameter causal transformer language model, with a 32-bit precision arithmetic coder. The system achieves state-of-the-art lossless compression ratios on English text, consistently compressing to approximately 14-15% of the original size across inputs ranging from 3 KB to 100 KB. The compressed output is well below the classical Shannon entropy bounds at all orders, demonstrating that the neural model captures structure inaccessible to frequency-based methods.

2. Background

2.1 Arithmetic Coding

Arithmetic coding is a mathematically near-optimal entropy coding method that maps an entire sequence of symbols to a single number in the half-open interval [0, 1). Unlike Huffman coding, which must assign an integer number of bits per symbol, arithmetic coding can approach the theoretical entropy limit to within a fraction of a bit for the entire sequence. For each symbol, the encoder narrows

the current interval proportionally to that symbol's probability. High-probability symbols barely shrink the interval (costing nearly zero bits), while low-probability symbols shrink it substantially (costing many bits). The width of the final interval determines the total number of compressed bits.

The theoretical minimum size for lossless compression of a source X is given by the Shannon entropy:

$$H(X) = -\sum P(x) \log_2 P(x)$$

Arithmetic coding achieves compression rates within a fraction of a bit of this bound, provided the probability model accurately reflects the true data distribution.

2.2 Neural Language Models as Probability Estimators

A causal transformer language model, given a sequence of tokens $t_1, t_2, ..., t_n$, produces a probability distribution $P(t_{n+1} \mid t_1, ..., t_n)$ over the full vocabulary for the next token. SmolLM2-135M is a 30-layer transformer with 135 million parameters and a vocabulary of 49,152 tokens, trained on large-scale text corpora. Despite its relatively small size, it captures grammatical rules, common phrases, semantic relationships, and factual knowledge --producing predictions far more accurate than byte-level frequency models.

3. System Architecture

3.1 Compression Pipeline

The compression pipeline operates as follows. (1) The input UTF-8 text is tokenized using the model's BPE tokenizer into a sequence of token IDs. (2) For each token position i, the language model takes the context $(t_1, ..., t_{i-1})$ and produces a probability distribution over the full vocabulary of 49,152 tokens. (3) The probability distribution is quantized to an integer cumulative distribution function (CDF) with a total of $2^{16} = 65{,}536$ counts, ensuring every token receives a minimum probability of at least 1/65,536 to avoid zero-width intervals. (4) The arithmetic encoder narrows its interval according to the CDF entry of the actual token t_i. After all tokens are processed, the encoder finalizes the bitstream.

3.2 Decompression Pipeline

Decompression is the mirror image of compression. The decompressor runs the exact same model with identical weights, producing identical probability distributions at each step. The arithmetic decoder uses the CDF and the compressed bitstream to recover each token. The recovered token is then fed back as context for the next step. After all tokens are decoded, the token sequence is detokenized back to UTF-8 text. Because both sides use the same deterministic model, reconstruction is perfectly lossless.

3.3 Arithmetic Coder Implementation

The arithmetic coder uses 32-bit integer precision. The encoder maintains a range [low, high] initialized to $[0, 2^{32} - 1]$. For each symbol, the range is narrowed: `high = low + (range * CDF[s+1]) / total - 1` and `low = low + (range * CDF[s]) / total`. Renormalization occurs when both endpoints fall in the same half of the range (MSB matching), outputting a bit and doubling the range. An underflow counter handles the near-convergence case where low >= QUARTER and high < 3*QUARTER. The decoder maintains a 32-bit value register initialized from the bitstream and performs symmetric renormalization. Symbol lookup uses binary search over the CDF.

3.4 CDF Quantization

The model's floating-point probability distribution (a 49,152-dimensional vector) is converted to an integer CDF for the arithmetic coder. The total CDF sum is fixed at $2^{16} = 65{,}536$. Each token is guaranteed a minimum count of 1, ensuring no zero-width intervals that would cause the decoder to fail. The remaining counts (65,536 - 49,152 = 16,384) are distributed proportionally to the model's probabilities. Rounding errors are absorbed by adjusting the count of the most probable token.

3.5 KV-Cache and Chunked Sliding Window

A naive implementation would re-process the entire growing context at each token position, resulting in $O(n^2)$ total computation. Nacrith uses the transformer's key-value (KV) cache: after the first full forward pass, each subsequent token requires only a single-token forward pass that reuses the cached attention states. This reduces per-token cost to O(1) amortized.

The model's context window is limited to 2,048 tokens. When the context exceeds this limit, a **chunked sliding window** strategy is employed: instead of dropping one token at a time (which would invalidate the cache every step), 512 tokens are dropped at once. The cache is rebuilt from the remaining 1,536 tokens in a single forward pass, after which 512 incremental steps proceed before the next rebuild. This amortizes the rebuild cost across 512 tokens, achieving approximately 500x speedup over the per-token rebuild approach.

3.6 Compressed File Format

Compressed files use the `.nc` extension (Nacrith Compressed) with a 12-byte header:

Offset	Size	Field	Description
0	4 bytes	Magic	NC01 (file identifier)
4	4 bytes	Token count	Number of tokens (uint32, big-endian)
8	4 bytes	Bit count	Compressed bits (uint32, big-endian)
12	variable	Stream	Arithmetic-coded bitstream

Table 1. Nacrith compressed file format (.nc).

4. Experimental Results

We evaluate Nacrith against three widely-used traditional compressors --gzip (DEFLATE, level 9), xz (LZMA2, level 9), and zip (DEFLATE, level 9) --on English prose samples of varying sizes. All experiments were conducted on an NVIDIA GTX 1050 Ti GPU (4 GB VRAM, CUDA capability 6.1). This is a low-end GPU; significantly faster compression is expected on modern hardware. The model uses approximately 1.3 GB of VRAM during operation, so any CUDA-capable GPU with at least 2 GB of VRAM is sufficient. CPU fallback is supported.

4.1 Compression Ratio

Sample	Original	gzip	xz	zip	Nacrith
small	3.0 KB	1.4 KB (46.8%)	1.5 KB (50.2%)	1.5 KB (49.9%)	**424 B (13.7%)**
medium	50.1 KB	19.6 KB (39.2%)	18.3 KB (36.6%)	19.7 KB (39.3%)	**7.4 KB (14.8%)**
large	100.5 KB	39.0 KB (38.9%)	35.5 KB (35.3%)	39.1 KB (38.9%)	**15.5 KB (15.4%)**

Table 2. Compression results on English prose. Percentages indicate compressed/original size (lower is better). Nacrith consistently achieves 14-15% compression ratio.

Nacrith achieves a compression ratio of approximately 14-15% across all tested input sizes, representing roughly 2.5x improvement over gzip and 2.3x over xz. The improvement is consistent from 3 KB to 100 KB inputs, demonstrating that the neural model's advantage is not limited to any particular scale. All results are fully lossless --decompressed output matches the original byte-for-byte.

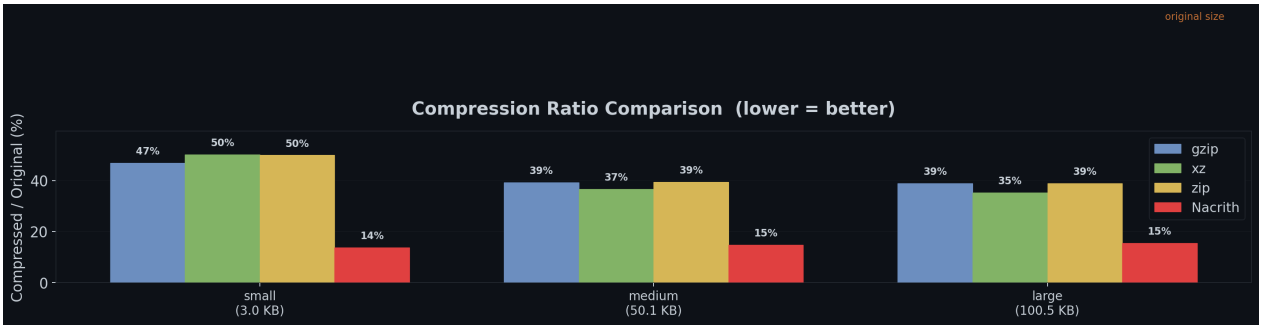

Figure 1. Compression ratio comparison (lower is better). Nacrith achieves 14-15% across all sizes, while traditional compressors range from 35-50%.

Figure 2. Absolute compressed sizes. Orange dashed line indicates original file size. Nacrith's output is a fraction of traditional compressors.

Figure 3. Space savings as percentage of original size (higher is better). Nacrith saves 85-86% consistently.

4.2 Beyond the Shannon Entropy Limit

The Shannon entropy provides a theoretical lower bound for compression assuming a particular order of statistical model. The 0th-order entropy considers only individual byte frequencies; the 1st-order considers bigram (byte-pair) frequencies; the 2nd-order considers trigram frequencies. We computed these bounds for the 100 KB benchmark file and compared them to the actual compressed sizes.

Method	Size	bits/byte
Original	100.5 KB	8.0000
Shannon 0th-order limit	59.5 KB	4.7398
Shannon 1st-order limit	44.2 KB	3.5213

Shannon 2nd-order limit	34.4 KB	2.7373
gzip -9	39.0 KB	3.1082
xz -9	35.5 KB	2.8257
Nacrith	**15.5 KB**	**1.2355**

Table 3. Shannon entropy bounds vs. actual compressed sizes on the 100 KB benchmark. Nacrith compresses 74% below the 0th-order, 65% below the 1st-order, and 55% below the 2nd-order Shannon entropy limit.

Nacrith achieves 1.24 bits/byte --dramatically below all classical Shannon entropy bounds. This is possible because the neural model captures statistical dependencies of far higher order than trigrams: grammatical structure, semantic coherence, and world knowledge spanning the full 2,048-token context window. For comparison, gzip and xz both operate above the 2nd-order Shannon limit, unable to exploit the deep structure that Nacrith leverages.

5. Discussion

5.1 Why Neural Compression Outperforms Traditional Methods

Traditional compressors rely on dictionary-based pattern matching: they search for repeated byte sequences within a sliding window and replace them with back-references. This approach can only exploit literal repetitions that occur within the window. A neural language model, by contrast, captures abstract linguistic patterns learned from billions of tokens during training. It can predict likely continuations based on grammar (subject-verb agreement), semantics (topical coherence), and world knowledge (common facts and phrases) --none of which require literal repetition in the input.

5.2 Computational Cost

The primary trade-off is speed. Each token requires a neural network forward pass, resulting in approximately 21 tokens/second on a low-end GTX 1050 Ti GPU. This makes Nacrith orders of magnitude slower than gzip or xz for real-time applications. However, with modern GPUs (e.g., RTX 4090 or A100), throughput would increase substantially due to higher memory bandwidth and compute capabilities. The model requires approximately 1.3 GB of VRAM, making it compatible with any CUDA-capable GPU with at least 2 GB of VRAM.

5.3 Model Overhead

Both the compressor and decompressor must have access to the same model weights (~259 MB). This overhead is amortized when compressing many files or large files, but makes the system impractical for compressing small individual files in isolation. The model is downloaded automatically from Hugging Face on first use.

5.4 Limitations and Future Work

The system is currently designed for UTF-8 text; binary data would not benefit from the language model's predictions. The context window is limited to 2,048 tokens, and compression efficiency may slightly degrade at sliding window boundaries. Future work could explore: (1) larger models with longer context windows for even better predictions, (2) quantized models (INT8/INT4) for faster inference with minimal accuracy loss, (3) batch processing of multiple files, and (4) extension to multilingual text using multilingual models.

6. Conclusion

Nacrith demonstrates that neural language models, when paired with arithmetic coding, can achieve lossless text compression ratios far beyond what traditional dictionary-based methods can attain. By compressing English text to approximately 14-15% of its original size --well below classical Shannon entropy bounds at multiple orders --Nacrith proves that deep linguistic structure provides a powerful and

previously untapped source of compression efficiency. While the computational cost is higher than traditional compressors, the dramatic improvement in compression ratio makes Nacrith a compelling choice for applications where storage efficiency is paramount.

References

[1] Shannon, C. E. (1948). "A Mathematical Theory of Communication." *Bell System Technical Journal*, 27(3), 379-423.

[2] Deletang, G., Ruoss, A., Duquenne, P.-A., Catt, E., Genewein, T., Mattern, C., Grau-Moya, J., Wenliang, L. K., Aitchison, M., Orseau, L., Legg, S., & Veness, J. (2024). "Language Modeling Is Compression." *Proceedings of ICLR 2024*. arXiv:2309.10668.

[3] Valmeekam, K., Marber, M., Sharan, V., & Kambhampati, S. (2023). "LLMZip: Lossless Text Compression using Large Language Models." arXiv:2306.04050.

[4] Ben Allal, L., Li, R., Kocetkov, D., Mou, C., Akiki, C., Ferrandis, C. M., Muennighoff, N., et al. (2025). "SmolLM2 --A family of small language models." Hugging Face. https://huggingface.co/HuggingFaceTB/SmolLM2-135M.

[5] Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). "Arithmetic Coding for Data Compression." *Communications of the ACM*, 30(6), 520-540.