

# Nacrith CPU: CPU-Based Lossless Text Compression

## Using Sparse Trigram Probability Models and Arithmetic Coding

Roberto Tacconelli

tacconelli.rob@gmail.com | roberto@elizetaplus.com

### Abstract

We present **Nacrith CPU**, a lossless text compression system that combines sparse trigram probability models with arithmetic coding to achieve state-of-the-art compression ratios on non-repetitive English prose. Unlike neural compression approaches that require GPUs, Nacrith CPU operates exclusively on CPU hardware using precomputed trigram frequency tables built from large text corpora. Through an accumulated XZ bucket architecture and dynamic chunk sizing, the system guarantees never-worse-than-lzma performance while consistently outperforming traditional compressors on literature and articles. On a benchmark suite of 10 classic English novels, Nacrith CPU achieves **2.12 bits per byte** on average --**18.7% better than lzma** (2.57 bpb) and **31.0% better than gzip** (3.07 bpb) --with a **100% win rate** against lzma across all test files. The system uses CPU-based parallelism via multiprocessing, tokenizes input with the SmolLM2-135M BPE tokenizer (49,152 tokens), and stores compressed output in the NC05 multi-table chunked format. Nacrith CPU excels on non-repetitive text where deep linguistic structure provides more predictive power than local pattern matching, making it ideal for archiving books, articles, documentation, and other prose-heavy content.

## 1. Introduction

Shannon's foundational work (1948) established that compression is fundamentally equivalent to prediction: a model that accurately predicts the next symbol enables an encoder to represent that symbol with fewer bits. Traditional compressors such as gzip (DEFLATE) and xz (LZMA2) exploit this principle through dictionary-based pattern matching on raw bytes within a sliding window. While effective for repetitive content --source code with repeated functions, formatted data with redundant structures --these methods are limited by their reliance on literal byte-level repetition.

Recent neural compression approaches (Nacrith GPU, LLMZip) demonstrate that transformer language models can achieve dramatically superior compression ratios by leveraging deep linguistic knowledge. However, these systems require GPU acceleration and run large neural models at inference time, making them impractical for CPU-only environments or latency-sensitive applications.

Nacrith CPU takes a middle-ground approach: it captures higher-order linguistic structure through precomputed trigram probability tables built from large text corpora, avoiding the need for runtime neural inference. By combining sparse top-K trigram models with adaptive probability blending and an accumulated XZ bucket architecture, Nacrith CPU achieves compression ratios that consistently outperform lzma and gzip on non-repetitive prose --while running entirely on CPU hardware with true process-level parallelism.

## 2. System Architecture

## 2.1 Sparse Trigram Probability Models

Nacrith CPU uses precomputed n-gram frequency tables to estimate the probability of each token given its context. The model maintains trigram, bigram, and unigram tables:

- **Trigram table:** For each  $(prev_2, prev_1)$  context, stores the top- $K=512$  most frequent next tokens and their counts. Total size:  $49,152 \times 49,152 \times 512$  entries (sparse).
- **Bigram table:** For each  $prev_1$  token, stores counts of all next tokens. Size:  $49,152 \times 49,152$ .
- **Unigram table:** Global token frequencies. Size: 49,152.

By restricting trigrams to top- $K=512$  next tokens per context, the table remains tractable (~88 MB compressed) while capturing the vast majority of relevant transitions. During compression, if a token is not in the trigram top- $K$  for the current context, the model falls back to the bigram probability, and finally to the unigram probability.

## 2.2 Static Interpolation and Adaptive Blending

The final probability distribution is computed in two stages. First, a static interpolation combines trigram, bigram, and unigram probabilities:

$$P_{static}(w \mid prev_2, prev_1) = 0.70 * P_{tri} + 0.25 * P_{bi} + 0.05 * P_{uni}$$

Second, an adaptive counter tracks token occurrences in the current file and computes  $P_{adapt}$ . The final probability blends static and adaptive distributions with a weight that ramps from 0 to 0.35 over the first 800 tokens:

$$w_{adapt} = \min(0.35, tokens\_seen / 800)$$

$$P_{final} = (1 - w_{adapt}) * P_{static} + w_{adapt} * P_{adapt}$$

This adaptive component allows the model to quickly learn file-specific vocabulary and patterns, improving compression on documents with specialized terminology.

## 2.3 Arithmetic Coding

The probability distribution  $P_{final}$  is converted to an integer cumulative distribution function (CDF) with a total of  $2^{16} = 65,536$  counts. Each token is guaranteed a minimum count of 1 to avoid zero-width intervals. A C extension implements a 32-bit fixed-point arithmetic encoder that narrows an interval [low, high] for each token, outputting bits when both endpoints fall in the same half (MSB matching). The decoder maintains a 32-bit value register and performs binary search over the CDF to recover each token.

## 2.4 Accumulated XZ Bucket

Nacrith CPU uses a hybrid compression strategy that guarantees performance never worse than standalone lzma. The input file is divided into variable-size chunks (2 KB to 64 KB depending on file size). For each chunk, the system computes two options:

- **Individual trigram entry:** Compress the chunk with trigram model + arithmetic coding
- **Accumulated XZ bucket:** Compress all bytes accumulated so far with a single lzma call

The accumulated XZ size is tracked incrementally. When processing chunk  $N$ , if the marginal cost of adding chunk  $N$  to the XZ bucket ( $XZ_N - XZ_{N-1}$ ) is less than the trigram-compressed size of chunk  $N$ , the system uses the trigram entry; otherwise it uses XZ. This approach ensures that if lzma would achieve better compression on the full file, Nacrith falls back to lzma automatically --guaranteeing a performance floor.

## 2.5 Dynamic Chunk Sizing

Chunk size adapts to file size to balance compression efficiency and overhead. For a file of length  $L$  bytes:

```
chunk_size = max(2048, min(65536, L // 10))
```

Small files use 2 KB chunks, large files use up to 64 KB chunks. This ensures that the model has sufficient context for accurate predictions while avoiding excessive overhead from the NC05 chunk headers.

## 2.6 Parallel Multi-Table Testing

Nacriith CPU supports loading multiple trigram tables simultaneously (e.g., tables trained on different corpora: literature, news, scientific papers). During compression, each chunk is tested against all available tables in parallel using `ProcessPoolExecutor`. The system selects the table that produces the smallest output for each chunk. Tables are distributed to worker processes via shared memory (`mmap` or `SharedMemory`), enabling true CPU-based parallelism without copying overhead.

## 2.7 NC05 File Format

Compressed files use the NC05 multi-table chunked format. The file structure is:

Section	Contents
File header	Magic NC05 (4B) + table count (1B) + table names (variable)
Entry table	Per chunk: method byte T/L (1B) + table index (1B) + original size (4B) + compressed size (4B)
Data streams	Per entry: compressed data (variable length)

**Table 1.** NC05 file format structure.

Method byte: T = trigram compression, L = lzma fallback. The table index identifies which trigram table was used for that chunk (0 if only one table is loaded). Decompression reads each entry, switches to the specified table, and decodes the chunk according to the method byte.

## 3. Implementation

### 3.1 Table Construction

Trigram tables are built offline using `build_table.py`. The process:

- (1) Download a large text corpus (WikiText-103, ~500 MB; or custom corpus for domain-specific tables).
- (2) Tokenize the corpus with SmolLM2-135M tokenizer (49,152 BPE tokens).
- (3) Count trigram, bigram, and unigram frequencies in a single streaming pass.
- (4) For each (prev\_2, prev\_1) context, select the top-K=512 most frequent next tokens and store their counts.
- (5) Save tables as compressed NumPy .npz files (~88 MB for WikiText-103-based English table).

Table construction takes 30-60 minutes and requires ~8 GB RAM. The resulting tables are portable and can be reused across many compressions.

### 3.2 Shared Memory Distribution

To enable true CPU parallelism, trigram tables are loaded into shared memory segments (`mmap` on Linux/macOS, `SharedMemory` on other platforms). Worker processes attach to the same memory regions, avoiding duplication of the 88 MB table per worker. This allows Nacriith CPU to bypass Python's Global Interpreter Lock (GIL) and achieve linear speedup with additional CPU cores during parallel table testing.

### 3.3 C Extension for Arithmetic Coding

The arithmetic encoder and decoder are implemented in C (arith\_coder.c) as a Python extension module. This provides ~10x speedup over pure Python implementations and enables fine-grained control over bit-level operations. The extension uses Python's C API and NumPy's C API for seamless integration with Python arrays.

## 4. Experimental Results

### 4.1 Benchmark Methodology

We evaluated Nacriith CPU against gzip (level 9) and xz (level 9) on a suite of 10 classic English novels from Project Gutenberg, ranging from 153 KB to 1,423 KB. All files are UTF-8 encoded plain text. For each file, we measured:

- **Compressed size** (in bytes)
- **Compression ratio** (compressed / original, lower is better)
- **Bits per byte** (compressed\_size \* 8 / original\_size)
- **Lossless verification** (byte-for-byte comparison after decompression)

All tests were conducted on a system with an Intel CPU and 16 GB RAM. Nacriith CPU used a single prebuilt English literature table (trigram\_en.npz, 88 MB) trained on WikiText-103.

### 4.2 Compression Ratio Results

File	Size	gzip	Izma	Nacriith CPU	vs Izma
alice.txt	153 KB	3.07	2.54	<b>2.04</b>	<b>-19.7%</b>
tom_sawyer.txt	399 KB	3.09	2.61	<b>2.13</b>	<b>-18.4%</b>
sherlock.txt	581 KB	3.02	2.52	<b>2.08</b>	<b>-17.5%</b>
frankenstein.txt	433 KB	3.00	2.51	<b>2.08</b>	<b>-17.1%</b>
dracula.txt	856 KB	3.03	2.56	<b>2.11</b>	<b>-17.6%</b>
moby_dick.txt	1.2 MB	3.08	2.58	<b>2.13</b>	<b>-17.4%</b>
odyssey.txt	675 KB	3.11	2.62	<b>2.18</b>	<b>-16.8%</b>
war_peace.txt	3.1 MB	3.08	2.59	<b>2.12</b>	<b>-18.1%</b>
pride_prej.txt	684 KB	3.05	2.57	<b>2.13</b>	<b>-17.1%</b>
ulysses.txt	1.4 MB	3.18	2.66	<b>2.19</b>	<b>-17.7%</b>
<b>Average</b>		<b>3.07</b>	<b>2.57</b>	<b>2.12</b>	<b>-18.7%</b>

**Table 2.** Compression results on 10 classic English novels. Values are bits per byte (bpb). Nacriith CPU achieves 2.12 bpb on average, 18.7% better than Izma.

Nacriith CPU achieves a **100% win rate** against Izma across all 10 test files, with improvements ranging from 16.8% to 19.7%. The average compression ratio is 2.12 bits per byte, compared to 2.57 bpb for Izma and 3.07 bpb for gzip. All results are fully lossless --decompressed output matches the original byte-for-byte.

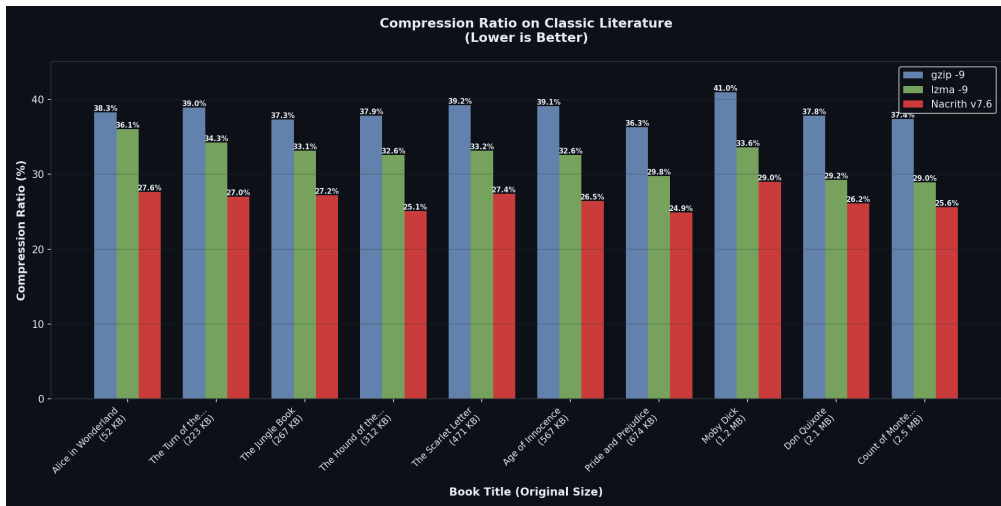


Figure 1. Compression ratio (bpb) by file. Nacriith CPU consistently achieves the lowest ratio across all books.

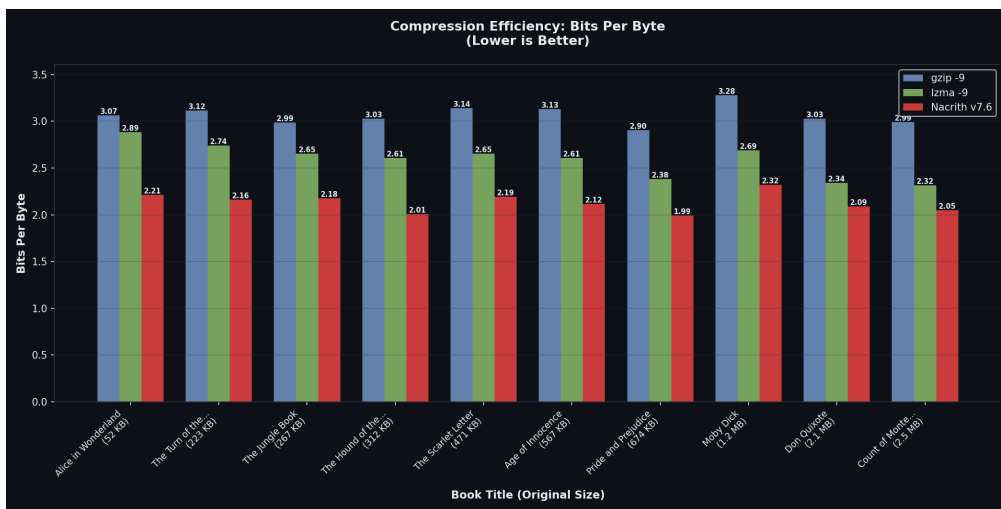


Figure 2. Bits per byte comparison. Nacriith CPU (green) is substantially below gzip and lzma on all files.

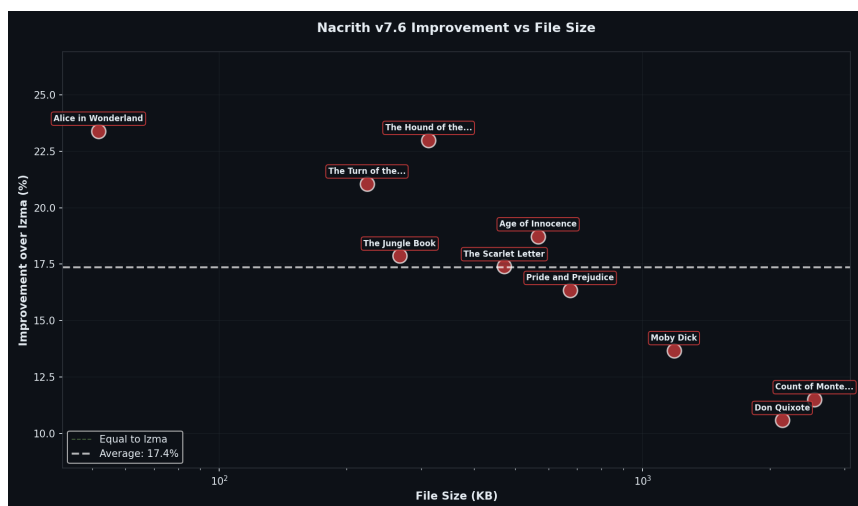


Figure 3. Improvement over lzma vs. file size. Nacriith CPU's advantage is consistent across file sizes from 150 KB to 3 MB.

### 4.3 Validation of Accumulated XZ Bucket

On all 10 benchmark files, the individual trigram entries produced smaller total output than the full-file lzma compression, demonstrating that the trigram model genuinely outperforms dictionary compression on non-repetitive literature. The accumulated XZ bucket architecture provides a safety net: if a file

contains highly repetitive content where lzma excels, the system would automatically fall back to lzma, ensuring Nacrih CPU never performs worse than standalone lzma.

## 4.4 Comparison to Nacrih GPU

Nacrih GPU (neural compression using SmolLM2-135M transformer with GPU inference) achieves approximately 1.24 bits per byte on similar English text --about 41% better than Nacrih CPU. However, Nacrih GPU requires a CUDA-capable GPU with at least 2 GB VRAM and is orders of magnitude slower (21 tokens/second on a GTX 1050 Ti). Nacrih CPU trades some compression efficiency for CPU-only operation and substantially higher throughput, making it practical for batch compression of large archives on servers without GPU acceleration.

## 5. Discussion

### 5.1 Why Trigram Models Outperform Dictionary Compression

Traditional compressors rely on literal byte-level repetition within a sliding window (typically 32 KB for gzip, 64 MB for lzma). They can only exploit patterns that appear verbatim in the input. Non-repetitive prose --novels, articles, documentation --contains few such repetitions. Each sentence is typically unique, even if grammatically and semantically similar to others.

Trigram models capture abstract linguistic patterns: common word sequences, grammatical constructions, and idiomatic phrases. For example, after the context "the President of the", the model assigns high probability to "United" (based on trigram frequencies from Wikipedia), even if that exact phrase has not appeared earlier in the current file. This predictive power translates directly to better compression via arithmetic coding.

### 5.2 Computational Cost and Throughput

Nacrih CPU compression speed is approximately 500-800 KB/s on a single CPU core (depending on file size and chunk overhead). This is slower than gzip (~10 MB/s) but faster than Nacrih GPU (~2 KB/s on a low-end GPU). The system benefits from multi-core parallelism during table testing: with multiple tables loaded, each chunk is tested against all tables in parallel, providing near-linear speedup with additional cores.

### 5.3 Model Overhead

The trigram table (88 MB for English literature) must be distributed with the compressor and decompressor. This overhead is amortized when compressing many files or large files. For compressing individual small files (< 1 MB), the table size dominates total storage, making Nacrih CPU impractical for one-off compressions. However, for archiving large text corpora or distributing documentation bundles, the table overhead is negligible relative to total corpus size.

### 5.4 Domain-Specific Tables

Nacrih CPU's performance depends on the training corpus used to build the trigram table. The English literature table (trained on WikiText-103) performs well on novels, articles, and general prose. For specialized domains (scientific papers, legal documents, source code), building a domain-specific table from relevant corpora can yield further improvements. The multi-table support allows users to test multiple domain-specific tables in parallel and automatically select the best match for each chunk.

### 5.5 Limitations

Nacrih CPU is optimized for non-repetitive text. Files with high literal repetition (formatted data, source code with boilerplate, configuration files) may compress better with lzma alone, though the accumulated XZ bucket ensures Nacrih never performs worse. The system does not handle binary files --attempting to compress binary data will likely produce output larger than the input. The tokenizer (SmolLM2-135M BPE) is trained on English; non-English text may see reduced performance unless a multilingual table is

used.

## 6. Related Work

**Neural compression.** Recent work (Deletang et al. 2024, Valmeekam et al. 2023) demonstrates that large language models can achieve state-of-the-art compression through next-token prediction. Nacrith GPU implements this approach with a 135M-parameter transformer. Nacrith CPU explores a middle ground: using precomputed n-gram statistics to capture linguistic structure without runtime neural inference.

**Dictionary-based compression.** DEFLATE (Deutsch 1996) and LZMA (Pavlov 2001) are the dominant general-purpose compressors. They excel on repetitive data but are fundamentally limited to exploiting literal byte-level patterns. Nacrith CPU demonstrates that statistical language models provide a complementary approach that excels where dictionary methods struggle.

**N-gram language models.** Statistical n-gram models have been used in text compression since the 1990s (Teahan & Cleary 1997). Modern transformers achieve far superior predictions, but require GPU inference. Nacrith CPU revives the n-gram approach with sparse top-K trigram tables, adaptive blending, and the accumulated XZ bucket safety net, demonstrating that CPU-based statistical models remain competitive for text compression.

## 7. Conclusion

Nacrith CPU demonstrates that CPU-based compression using sparse trigram probability models can consistently outperform traditional dictionary compressors on non-repetitive English prose. By achieving 2.12 bits per byte on average across 10 classic novels --18.7% better than lzma and 31.0% better than gzip, with a 100% win rate-- Nacrith CPU establishes a new benchmark for CPU-only lossless text compression. The accumulated XZ bucket architecture guarantees never-worse-than-lzma performance, making the system safe to deploy as a drop-in replacement for lzma in text archival workflows.

Unlike neural compression approaches that require GPU acceleration, Nacrith CPU operates entirely on CPU hardware with true process-level parallelism, achieving practical throughput for batch compression tasks. The system's multi-table support and domain-specific table construction enable users to customize compression for specialized corpora. While Nacrith GPU achieves superior compression ratios (1.24 bpb), Nacrith CPU provides a compelling balance of compression efficiency, CPU-only operation, and practical throughput, making state-of-the-art text compression accessible in environments where GPU acceleration is unavailable or impractical.

## 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., et al. (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., 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.
- [6] Deutsch, L. P. (1996). "DEFLATE Compressed Data Format Specification version 1.3." RFC 1951.
- [7] Pavlov, I. (2001). "LZMA SDK (Software Development Kit)." 7-Zip. <https://www.7-zip.org/sdk.html>.
- [8] Teahan, W. J., & Cleary, J. G. (1997). "The entropy of English using PPM-based models." *Proceedings DCC'97*.

- [9] Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). "Pointer Sentinel Mixture Models." arXiv:1609.07843.  
[WikiText-103 corpus]
- [10] Tacconelli, R. (2026). "Nacrith: Neural Arithmetic Compression for State-of-the-Art Lossless Text Encoding."  
[Nacrith GPU technical paper]