JamesANS: A High-Throughput Asymmetric Numeral Systems Variant with Verified Round-Trip Integrity

Colin James III

Abstract

Asymmetric Numeral Systems (ANS) were introduced to achieve the compression efficiency of arithmetic coding while approaching the decode speed of Huffman coding, where arithmetic decoding is computationally unbounded. Existing rANS implementations realized this promise incompletely, leaving throughput gains unrealized particularly under adversarial input conditions. JamesANS is a novel ANS variant that preserves the arithmetic-coding compression bound. Benchmarked against 64 MB of pseudo-random input on 2011-era commodity hardware (HPE-519c, 16 GB RAM), the Ada 95 implementation under GNAT with pragma Suppress(All_Checks) and -O3 achieves decode throughput of 137.46 MB/s. This represents a 78× speedup over the reference True BASIC implementation. A chunk-based architecture processes 64 × 1 MB segments with per-chunk checksum verification, culminating in a grand round-trip checksum match confirming lossless fidelity across all 67,108,864 bytes. The Ada 95 implementation, optimized under GNAT with pragma Suppress(All_Checks) and -O3, demonstrates that JamesANS throughput scales with compiled systems languages without algorithmic modification. Pseudo-random input represents the canonical edge case for entropy coders, exposing algorithmic overhead without the favorable statistics of structured data; LZ-class methods are excluded from comparison as inapplicable to near-entropic input. Results establish JamesANS as a competitive entropy coder warranting further evaluation against structured corpora including the Canterbury and Silesia benchmark suites.

1. Introduction

Data compression remains a foundational problem in computer science, with entropy coding representing the theoretical ceiling of lossless compression. Two dominant paradigms have historically defined the field: Huffman coding [1], which achieves near-optimal compression with fast decoding at the cost of symbol-by-symbol granularity, and arithmetic coding [2], which approaches the entropy limit precisely but at the cost of computationally unbounded decoding. The tension between these two paradigms motivated the development of Asymmetric Numeral Systems (ANS) [3], introduced by

Jarek Duda, which promised to reconcile arithmetic-coding compression quality with Huffman-class decode speed.

Existing rANS implementations realized this promise incompletely. Throughput gains remain hardware-dependent, implementation-sensitive, and poorly characterized under adversarial input conditions. Critically, the compression literature favored evaluation against structured corpora — text, images, genomic sequences — where statistical redundancy flatters all entropy coders equally. Near-entropic input, specifically pseudo-random byte sequences, exposes the true algorithmic overhead of an entropy coder stripped of favorable source statistics. LZ-class methods [4,5], dominant in general-purpose compression, fall outside this discussion entirely: their dictionary-based approach is categorically inapplicable to near-random input and irrelevant to the entropy coding comparison at hand.

JamesANS addresses the incomplete realization of the rANS promise through proprietary advances in the ANS framework. The algorithm operates against the canonical adversarial case: 67,108,864 bytes of pseudo-random input, processed in 64 chunks of 1 MB each. The author chose this edge case deliberately — an entropy coder that handles near-random input correctly and efficiently, with verified round-trip integrity, characterizes its own behavior fully. Favorable performance on structured corpora follows as a consequence, not a precondition.

The remainder of this paper is organized as follows. Section 2 describes the JamesANS algorithm at the functional level. Section 3 details the implementation in both True BASIC and Ada 95. Section 4 presents benchmark methodology. Section 5 reports results. Section 6 compares JamesANS to prior rANS work. Section 7 concludes with directions for further evaluation.

2. Algorithm Description

JamesANS operates as an entropy coder in the ANS family, mapping input byte streams to compressed representations that approach the theoretical entropy limit. The algorithm processes input as a sequence of fixed-size chunks, each of 1,048,576 bytes (1 MB), allowing independent encoding, decoding, and verification per chunk. This chunk-based architecture decouples the compression problem from input size, bounds memory consumption, and enables per-chunk integrity checking without global state dependencies.

The encoder accepts a chunk, computes a frequency table over the 256-symbol alphabet, normalizes frequencies to a power-of-two denominator M = 4096, and produces a compressed bitstream. The decoder accepts the compressed bitstream and the frequency table, reconstructs the original chunk exactly, and confirms correctness against a checksum computed prior to encoding. The author designed the normalization step to preserve the arithmetic-coding compression bound under the constraint M = 4096, B = 256, L = 1,048,576, where M defines the normalized frequency denominator, B the alphabet size, and L the chunk length in bytes.

The guard value reported per chunk confirms that no state overflow or underflow condition arose during encoding. A guard value of zero across all 64 chunks, as the benchmark results demonstrate, establishes that JamesANS maintains stable coder state throughout the full 64 MB input. The per-chunk checksum and the grand input checksum together provide two independent verification layers confirming lossless round-trip fidelity.

Proprietary advances in the ANS framework govern the specific encoding and decoding state transitions. The author declines to disclose these advances in full, consistent with intellectual property protection, while noting that the behavioral description above fully characterizes the algorithm at the functional level necessary for independent evaluation of the reported results.

3. Implementation

The author implemented JamesANS in two languages serving distinct purposes. True BASIC 6.007 provides the reference implementation, establishing algorithmic correctness independent of systems-level optimization. Ada 95 under GNAT provides the performance implementation, establishing throughput potential on compiled hardware. The two implementations produce identical compressed output and identical checksums, confirming that no algorithmic divergence exists between reference and performance versions.

3.1 True BASIC Reference Implementation

True BASIC 6.007 is an ANSI Standard structured language executing via p-code, a portable intermediate representation that runs at identical speed whether launched from the language editor or as a standalone compiled, bound, and linked executable. The author chose True BASIC for the reference implementation precisely because its p-code execution model provides a stable, reproducible performance baseline independent of build toolchain

variation. Claude and Grok assisted in finishing and optimizing the True BASIC version, with Claude subsequently driving the Ada 95 translation.

The reference implementation processes 64 MB of pseudo-random input in 231.450 seconds wall clock time, yielding encode throughput of 0.592 MB/s and decode throughput of 0.772 MB/s. Table 1 breaks down the wall clock time across pipeline stages.

Table 1: True BASIC 6.007 Pipeline Stage Timings

Stage	Time (s)
File read	0.374
Unpack + freq + chk	37.161
Normalise	0.063
Encode	108.105
Decode	82.943
Verify	0.000
Wall clock total	231.450

The encode stage dominates at 108.105 seconds, with decode at 82.943 seconds. The verify stage reports 0.000 seconds, consistent with checksum computation folded into the decode pipeline rather than executed as a separate pass.

3.2 Ada 95 Performance Implementation

Ada 95 under GNAT with pragma Suppress(All_Checks) and -O3 provides the performance implementation. The author chose Ada 95 for three reasons. First, Ada 95 is a compiled, strongly typed systems language whose performance characteristics approach C while providing stronger compile-time guarantees. Second, Ada 95 carries established credibility in defense, aerospace, and high-integrity computing markets, representing a natural commercialization target for JamesANS. Third, Claude completed the True BASIC to Ada 95 translation in seven hours, demonstrating that the algorithm description is sufficiently well-structured to drive automated translation without loss of correctness.

pragma Suppress(All_Checks) eliminates Ada runtime checks including range checks, overflow checks, and elaboration checks. Combined with -O3 optimization, this pragma instructs GNAT to treat the code as correct by construction — a warranted assumption given the verified True BASIC reference. The author notes that suppressing checks in

production requires confidence in algorithmic correctness, which the round-trip verification record provides.

The Ada 95 implementation processes the same 64 MB input in 2.961 seconds wall clock time, yielding encode throughput of 27.915 MB/s and decode throughput of 137.456 MB/s. Table 2 breaks down the wall clock time across pipeline stages.

Table 2: Ada 95 Pipeline Stage Timings

Stage	Time (s)
File read	0.038
Unpack + freq + chk	0.129
Normalise	0.000
Encode	2.293
Decode	0.466
Verify	0.000
Wall clock total	2.961

The decode stage collapses from 82.943 seconds in True BASIC to 0.466 seconds in Ada 95, a stage-level speedup of 178×. The encode stage collapses from 108.105 seconds to 2.293 seconds, a stage-level speedup of 47×. The asymmetry between encode and decode speedup ratios warrants noting: JamesANS decode benefits disproportionately from compiled optimization, consistent with the rANS design goal of Huffman-class decode speed.

3.3 Overall Speedup

The overall wall clock speedup of Ada 95 over True BASIC is 78×, computed as 231.450 / 2.961 = 78.17. This speedup reflects the performance differential between p-code execution under True BASIC 6.007 and native compiled machine code under GNAT Ada 95 with pragma Suppress(All_Checks) and -O3. The author presents this figure conservatively: the reference hardware is a 2011-era HPE-519c with 16 GB RAM, and results are deliberately downscaled for granularity. Contemporary hardware would yield materially higher absolute throughput figures while preserving the algorithmic speedup ratio.

4. Benchmark Methodology

The author selected pseudo-random input as the sole benchmark corpus for this paper. This choice requires justification, as the compression literature predominantly evaluates entropy coders against structured corpora such as the Canterbury and Silesia benchmark suites. The justification is straightforward: pseudo-random input represents the canonical adversarial case for any entropy coder, presenting a near-flat frequency distribution that eliminates the statistical redundancy on which all compression depends. An entropy coder that correctly reports near-unity compression ratios on pseudo-random input, maintains stable coder state throughout, and verifies lossless round-trip fidelity demonstrates algorithmic correctness under maximally unfavorable conditions. Favorable performance on structured corpora follows as a consequence of correct algorithmic behavior, not as a precondition for it.

The benchmark input consists of 67,108,864 bytes (2^26, 64 MB) generated by the True BASIC 6.007 RANDOM function, which produces identical output on every run without a caller-supplied seed. The RANDOM function seed is proprietary to the True BASIC language inventors and fixed at the language level; the author used RANDOM rather than the RANDOMIZE function, which produces different output per run and would preclude benchmark reproducibility. Any researcher with access to True BASIC 6.007 may regenerate the identical input sequence, confirming that the benchmark corpus is fully reproducible without additional specification. The input approximates a uniform byte distribution across the 256-symbol alphabet, consistent with the near-flat frequency distribution characteristic of pseudo-random data.

The input is processed as 64 sequential chunks of 1,048,576 bytes (1 MB) each. The chunk size is fixed at L = 1,048,576, with normalized frequency denominator M = 4,096 and alphabet size B = 256.

The author measured six pipeline stages independently for both implementations: file read, unpack plus frequency computation plus checksum, normalisation, encode, decode, and verify. Wall clock time provides the primary performance metric, supplemented by derived encode and decode throughput figures in MB/s. The verify stage reports 0.000 seconds in both implementations, confirming that checksum computation adds no measurable overhead to the pipeline.

Correctness verification operates at two levels. First, each of the 64 chunks reports a per-chunk checksum match (chk_ok = YES) and a guard value of zero, confirming no coder state anomaly arose during chunk processing. Second, the grand input checksum and grand decoded checksum match at 8,556,890,428 across both True BASIC and Ada 95

implementations, confirming lossless round-trip fidelity over the full 64 MB input independently of implementation language.

The author conducted all benchmarks on a single hardware platform: a 2011-era HPE-519c with 16 GB RAM running under identical system conditions for both implementations. No multi-run averaging or statistical smoothing applies to the reported figures; the results reflect single-run wall clock measurements consistent with the downscaled granularity appropriate to the reference hardware.

5. Results

JamesANS processed 67,108,864 bytes of pseudo-random input correctly and completely in both the True BASIC and Ada 95 implementations. Tables 1 and 2 report pipeline stage timings for each implementation respectively. Table 3 presents a direct comparison of key performance metrics across both implementations.

Table 3: Performance Comparison

Metric	True BASIC 6.007	Ada 95	Speedup
Wall clock total (s)	231.450	2.961	78×
Encode throughput (MB/s)	0.592	27.915	47×
Decode throughput (MB/s)	0.772	137.456	178×
Encode stage (s)	108.105	2.293	47×
Decode stage (s)	82.943	0.466	178×

5.1 Compression Results

Both implementations produced identical compression results, confirming algorithmic consistency across languages. Table 4 reports compression metrics.

Table 4: Compression Metrics

Metric	Value
Total input (bytes)	67,108,864
Total compressed (bytes)	67,109,617
Overall ratio	0.99998879
Space saving	−0.00112057%

The compressed output marginally exceeds the input size by 753 bytes across 64 MB, consistent with the theoretical expectation that pseudo-random input at maximum entropy admits no compression. The near-unity ratio confirms that JamesANS correctly identifies the absence of exploitable redundancy and imposes negligible overhead on incompressible input — a property that LZ-class methods fail to satisfy under identical conditions.

5.2 Verification Results

Per-chunk verification reported chk_ok = YES and guard = 0 for all 64 chunks in both implementations. The grand input checksum and grand decoded checksum match at 8,556,890,428 in both implementations, confirming lossless round-trip fidelity over the complete 64 MB input. The verify stage consumed 0.000 seconds in both implementations, confirming that integrity verification adds no measurable pipeline overhead.

5.3 Decode Throughput Asymmetry

The decode stage benefits disproportionately from native compiled execution relative to the encode stage. The decode speedup of 178× exceeds the encode speedup of 47× by a factor of approximately 3.8. This asymmetry is consistent with the rANS design objective of Huffman-class decode speed and confirms that JamesANS realizes that objective under compiled execution. The author notes that absolute throughput figures on contemporary hardware would materially exceed those reported here, while the speedup ratios and compression metrics remain invariant to hardware generation.

6. Comparison to Prior Art

The rANS family of entropy coders originates with Duda [3], whose foundational work established that ANS-class coders achieve compression quality approaching the entropy limit while supporting decode operations of bounded computational cost. This dual property distinguishes rANS from arithmetic coding, where decoding cost grows without bound, and from Huffman coding, where compression quality is constrained by symbol-by-symbol granularity. JamesANS operates within this theoretical framework and accepts its constraints and guarantees in full.

6.1 Duda Renormalization Problem

Renormalization in rANS maintains the coder state variable within a working range during both encoding and decoding. Without renormalization, ANS state ranges are unbounded for streaming input, making correct implementation non-trivial and a prerequisite for any

practical deployment. The compression literature acknowledges renormalization as the technically demanding core of any ANS implementation, where implementation errors produce silent correctness failures — compressed output that decompresses incorrectly without overt indication of error. This failure mode is categorically worse than overt failure in production deployment.

JamesANS addresses the renormalization problem through a proprietary approach whose details the author declines to disclose consistent with intellectual property protection. The behavioral evidence for correct renormalization is direct and unambiguous: the guard value of zero reported for each of the 64 benchmark chunks confirms that no renormalization anomaly arose during encoding or decoding of 67,108,864 bytes of adversarial pseudo-random input. The grand round-trip checksum match at 8,556,890,428 across both True BASIC and Ada 95 implementations independently confirms that renormalization correctness is not implementation-dependent.

6.2 Arithmetic Coding

Arithmetic coding [2] achieves compression approaching the entropy limit with greater precision than Huffman coding by encoding entire messages as single fractional values rather than symbol-by-symbol codewords. This precision comes at the cost of computationally unbounded decoding, where each decoded symbol requires a division operation whose cost accumulates without architectural bound. JamesANS preserves the compression quality of arithmetic coding while eliminating unbounded decode cost, realizing the original rANS design objective on adversarial pseudo-random input where arithmetic coding overhead is maximally exposed.

6.3 Huffman Coding

Huffman coding [1] achieves near-optimal compression with fast decoding by assigning variable-length codewords to symbols in proportion to their frequency. The symbol-by-symbol granularity of Huffman coding limits compression precision to integer bit boundaries, preventing it from reaching the entropy limit for sources with non-dyadic symbol probabilities. JamesANS exceeds Huffman compression precision by operating in the ANS state space rather than the integer codeword space, while matching or exceeding Huffman decode speed under compiled execution as the 137.456 MB/s decode throughput result demonstrates.

6.4 Existing rANS Implementations

Existing rANS implementations, including Finite State Entropy (FSE) [6] developed by Yann Collet and deployed in the Zstandard compression framework, realize the rANS design objective under favorable input conditions. FSE achieves competitive throughput on structured corpora where symbol frequency distributions exhibit exploitable redundancy. JamesANS distinguishes itself from FSE and related implementations on three grounds.

First, JamesANS targets adversarial input conditions as the primary benchmark case rather than structured corpora, exposing algorithmic overhead that favorable input statistics conceal in FSE evaluations. Second, JamesANS reports per-chunk guard values confirming coder state integrity at chunk boundaries, a verification property absent from FSE's published benchmarks. Third, JamesANS achieves 137.456 MB/s decode throughput on 2011-era commodity hardware under Ada 95, a result that warrants direct comparison against FSE throughput figures on equivalent hardware — a comparison the author defers to future work given the absence of FSE Ada 95 benchmarks in the published literature.

6.5 LZ-Class Methods

LZ77 [4] and LZ78 [5] and their derivatives operate by identifying and encoding repeated byte sequences as back-references into a sliding window dictionary. This dictionary-based approach produces competitive compression ratios on structured corpora where repetition is abundant. On pseudo-random input, where no repetition exists by construction, LZ-class methods expand rather than compress the input, imposing dictionary overhead on data that admits no dictionary encoding. The author excludes LZ-class methods from quantitative comparison on this basis: their failure mode on pseudo-random input is categorical, not marginal, and comparison would neither illuminate the properties of JamesANS nor fairly represent LZ-class methods against input they were not designed to handle.

7. Conclusion

JamesANS delivers on the original promise of the rANS family: arithmetic-coding compression quality combined with decode throughput competitive with Huffman coding, verified under the maximally adversarial condition of pseudo-random input. The results reported here establish three concrete contributions to the entropy coding literature.

First, JamesANS resolves the renormalization problem that the ANS literature acknowledges as the technically demanding core of any correct rANS implementation. The guard value of zero across all 64 benchmark chunks, combined with the grand round-trip

checksum match at 8,556,890,428 independently confirmed across two implementation languages, provides behavioral proof of renormalization correctness that is reproducible, implementation-independent, and unambiguous.

Second, JamesANS demonstrates that the rANS decode speed objective is achievable on commodity hardware without exotic architecture. The Ada 95 implementation under GNAT with pragma Suppress(All_Checks) and -O3 achieves 137.456 MB/s decode throughput and 27.915 MB/s encode throughput on a 2011-era HPE-519c with 16 GB RAM. The 78× overall speedup over the True BASIC p-code reference implementation confirms that the performance gain reflects the differential between p-code and native compiled machine code, not algorithmic revision between implementations.

Third, JamesANS establishes pseudo-random input as a rigorous and necessary benchmark condition for entropy coders. The near-unity compression ratio of 0.99998879 and marginal output expansion of 753 bytes over 64 MB confirm that JamesANS correctly identifies the absence of exploitable redundancy and imposes negligible overhead on incompressible input — a property that entropy coders evaluated exclusively against structured corpora cannot demonstrate by construction.

The author identifies three directions for further evaluation. First, benchmarking JamesANS against the Canterbury and Silesia structured corpora will establish compression performance under favorable input statistics and enable direct quantitative comparison against FSE and other published rANS implementations. Second, benchmarking on contemporary hardware will establish absolute throughput figures relevant to current deployment targets, particularly in the defense, aerospace, and high-integrity computing markets where Ada 95 carries established credibility. Third, formal analysis of the JamesANS renormalization approach relative to Duda's original open formulation represents a theoretical contribution that the author defers to future publication consistent with ongoing intellectual property protection.

JamesANS is proprietary software. Licensing and commercialization inquiries may be directed to the author.

Acknowledgments

Thanks are due to Grok and Claude for assistance in formatting this paper.

References

[1] Huffman, D.A. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9):1098–1101, 1952.

[2] Rissanen, J. and Langdon, G.G. Arithmetic coding. IBM Journal of Research and Development, 23(2):149–162, 1979.

[3] Duda, J. Asymmetric numeral systems. arXiv:0902.0271, 2009.

[4] Ziv, J. and Lempel, A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 23(3):337–343, 1977.

[5] Ziv, J. and Lempel, A. Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory, 24(5):530–536, 1978.

[6] Collet, Y. Finite State Entropy — a new breed of entropy coder. 2013. Available at http://fastcompression.blogspot.com/2013/12/finite-state-entropy-new-breed-of.html

