Appendix

A CODEC State Storage Analysis

For the main configuration studied, the key advantage of *Shannonic* is that it compresses the symbol space from 8 bits (256 values) to 4 bits (16 ranges). This enables an 8× reduction in tANS state count: $L = 128$ states (2^7) instead of the $L = 1024$ (2^{10}) states typically required for 8-bit symbols in production codecs such as `zstd` (Collet & Kucherawy, 2021). This reduction yields storage savings in two ways: 1. all tables have 8× fewer entries, and 2. each entry stores fewer bits proportional to the reduced state and symbol counts. The reduction comes with a small overhead for partition mapping and metadata. We analyze the storage requirements below, comparing *Shannonic* ($K = 16$ ranges, $L = 128$ states) against baseline flat tANS ($L_{\text{flat}} = 1024$ states, 256 symbols).

***Shannonic* encoder tables.** The encoder requires a range identifier map `rangeId[256]` that maps each byte to its range index, requiring $256 \times \lceil \log_2 K \rceil = 256 \times 4 = 1024$ bits = 128 B. Additionally, for each of the $K = 16$ ranges, the encoder stores: `base[s]` the minimum symbol value, 8 bits, `nb[s]` the number of offset bits, 3 bits, and additional encoding metadata, 11 bits. These arrays are accessed together using the same range index s and collectively require $16 \times (8 + 3 + 11) = 16 \times 22 = 352$ bits = 44 B. Finally, the encoder state transition table `encTable[128]` stores the next state for each of the $L = 128$ states, requiring $\lceil \log_2 L \rceil = 7$ bits per entry, totaling $128 \times 7 = 896$ bits = 112 B. **Total: 306 B.**

***Shannonic* decoder tables.** The decoder uses a transition table `decTable[128]` storing for each of the $L = 128$ states the range ID $\lceil \log_2 K \rceil = 4$ bits, bit count $\lceil \log_2 8 \rceil = 3$ bits, and next state base $\lceil \log_2 L \rceil = 7$ bits, totaling $128 \times (4 + 3 + 7) = 128 \times 14 = 1792$ bits = 224 B. The partition metadata `base[16]` and `nb[16]` totaling $16 \times (8 + 3) = 176$ bits = 22 B is required by the decoder. Note that `rangeId[256]` is only used during encoding, not decoding. **Total: 246 B.** When encoder and decoder are co-located, the partition metadata `base[16]`, `nb[16]`, and `rangeId[256]` totaling 150 B can be shared between them, reducing the combined footprint to 530 B rather than 552 B.

Standard tANS tables. For a fair comparison using the same implementation approach as *Shannonic*, the baseline encoder implementing standard tANS stores per-symbol parameters for each of the 256 symbols: start position, 10 bits, bias for bit-flush calculation, approximately 11 bits, and frequency parameters, 4 bits, accessed together and requiring $256 \times (10 + 11 + 4) = 256 \times 25 = 6400$ bits = 800 B. The encoder state transition table stores next states for 1024 entries at 11 bits each, totaling $1024 \times 11 = 11264$ bits = 1408 B. The decoder explicitly stores for each of the $L_{\text{flat}} = 1024$ states the symbol ID, 8 bits, bit count, 4 bits, and next state base, 10 bits, totaling $1024 \times (8 + 4 + 10) = 1024 \times 22 = 22528$ bits = 2816 B. **Encoder total: 2208 B; Decoder total: 2816 B.**

Comparison. Table 6 summarizes the storage requirements of *Shannonic* vs. standard tANS. Compared to standard tANS, *Shannonic* requires 7.2× less storage for the encoder and 11.4× less storage for the decoder.

This reduction benefits all deployment contexts: on high-performance systems, the 530 B combined tables remain L1-resident, enabling multi-stream parallelism without memory bandwidth bottlenecks; on resource-constrained edge devices, the compact footprint fits within on-chip memory budgets that standard tANS would exceed; and in dedicated hardware, smaller tables reduce LUT and block RAM requirements, enabling single-cycle SRAM access with lower area and power.

Table 6: CODEC storage requirements.

Component	*Shannonic*	Flat tANS	Reduction
Encoder tables	306 B	2208 B	7.2×
Decoder tables	246 B	2816 B	11.4×
Combined (shared)	530 B	5024 B	9.5×

B Software Implementation

Implementation. We implement *Shannonic* as a compact C routine optimized for throughput on both high-performance and resource-constrained platforms. We use a hybrid storage strategy that incurs 3% storage overhead while improving throughput: the `rangeId[256]` map is bit-packed at 4 bits per entry, requiring one shift and mask operation for extraction, while `start[16]` and `nb[16]` arrays are byte-aligned and combined for direct access. State transition tables use 8-bit entries for encoding and 16-bit entries for decoding. Bits are accumulated in a 64-bit register and flushed in 8-byte batches.

Indexing. All accesses are sequential and cache-friendly, contributing to the L1-resident working set discussed in Section 5.2. The encoder accesses: 1. bit-packed `rangeId[x]`, 2. metadata `start[s]` and `nb[s]`, and 3. `encTable[start[s] + X]`. The decoder accesses: 1. `decTable[X]` retrieving range ID, bit count, and base state (256 B), and 2. the same metadata (32 B).

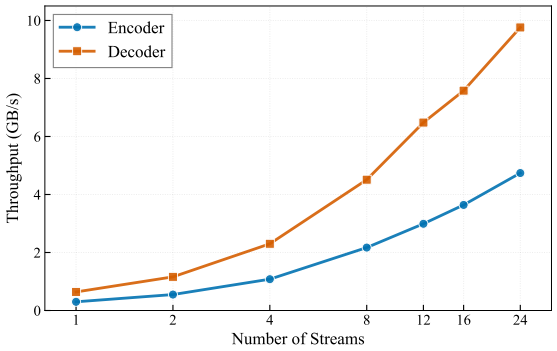

Figure 4: Multi-stream throughput scaling on i9-10920X. Both encoder and decoder scale effectively across all thread counts, reaching 4.74 GB/s and 9.76 GB/s respectively at 24 streams.

Performance evaluation. We compile with GCC 13 `-O3` and measure performance on two platforms: an Intel i9-10920X (3.50 GHz) as a representative high-performance system, and a Raspberry Pi 5 (ARM Cortex-A76, 2.4 GHz, 4 cores) as a representative resource-constrained device. Measurements use RDTSC over 10^8 symbols after cache warm-up.

Table 7 presents single-stream performance with word-based I/O batching. The encoder achieves 300 MB/s on the i9-10920X and 168 MB/s on the Raspberry Pi 5. The decoder achieves 640 MB/s and 286 MB/s respectively.

Table 7: Single-stream software performance.

Platform	Encoder MB/s	Decoder MB/s
i9-10920X	300	640
Raspberry Pi 5	168	286

To achieve higher bandwidth, we process multiple independent streams in parallel, with streams sharing lookup tables. Figure 4 shows throughput scaling on the i9-10920X across 1–24 concurrent streams. At 8 streams, the system achieves 2.17 GB/s encoding and 4.51 GB/s decoding. Utilizing all 24 hardware threads reaches 4.74 GB/s encoding and 9.76 GB/s decoding. Table 8 shows the Raspberry Pi 5 achieves 669 MB/s encoding and 1.14 GB/s decoding at 4 streams.

Table 8: Raspberry Pi 5 multi-stream performance.

Streams	Encoder MB/s	Decoder MB/s
1	168	286
2	336	571
4	669	1139

C Hardware Implementation

We implement *Shannonic* in RTL and synthesize it to the 7nm ASAP7 PDK (Clark et al., 2016), validating area, power, and throughput targets. The architecture directly implements the encoding and decoding algorithms from Section 4.2, with dual-lane interleaving to achieve single-cycle throughput.

Table organization. The hardware implementation uses the same table structures analyzed in Section A. Small metadata arrays totaling 44 B are realized as register arrays for single-cycle access. The larger state transition tables—128 B for the encoder `encTable[128]` and 256 B for the decoder `decTable[128]` with byte-aligned 16-bit entries—are implemented as single-port SRAMs. The `rangeId[256]` partitioner map of 128 B is bit-packed.

Encoder microarchitecture. The encoder implements dual-lane interleaving: two independent ANS state registers `ans_state0` and `ans_state1` alternate on consecutive clock cycles, enabling an initiation interval of one cycle. Each lane processes symbols independently while sharing the lookup tables. The pipeline consists of three stages:

1. **Metadata access (Stage 1):** Corresponds to Algorithm 2 Steps 2-4. The partition index s is used to look up encoding metadata from register arrays; compute the number of bits to emit and the SRAM address for state transition.

2. **Address registration (Stage 2):** Pipeline the computed state transition address to balance timing.

3. **State transition (Stage 3):** Corresponds to Steps 5-6. Look up the next state from the 128 B SRAM and emit output bits.

The front-end partitioner (Algorithm 2 Steps 1-2) adds one cycle of latency, mapping input symbols to partition indices and offsets via the `rangeId` table. Offset bits are extracted and passed through untouched to enable independent buffering in the decoder.

Decoder microarchitecture. The decoder employs the same dual-lane structure with a three-stage pipeline:

1. **State lookup (Stage 1):** Corresponds to Algorithm 2 Step 8. Access the 256 B SRAM using the current ANS state X as the address to retrieve the partition index s, bit count `nbBits`, and base state `newX`.

2. **Data alignment (Stage 2):** Register the retrieved values and align the independently streamed offset data.

3. **State reconstruction and merge (Stage 3):** Corresponds to Algorithm 2 Steps 9-11. Combine the base state with consumed bits from the bitstream to produce the next state $X \leftarrow \texttt{newX} + b$, then merge the partition index with the aligned offset to reconstruct the original symbol $x \leftarrow \texttt{base}[s] + o$.

Evaluation. We synthesize both encoder and decoder using SiliconCompiler (Zero ASIC Corporation, 2024) with the ASAP7 7nm predictive PDK (Clark et al., 2016). Since ASAP7 lacks SRAM compiler support, we replace the state transition SRAMs with simple dummy logic providing predictable timing characteristics that enables complete synthesis, place-and-route, and timing analysis of the control logic, datapath registers, and routing overhead. To estimate the area and power of the actual SRAM blocks, we use FN-CACTI (Raha et al., 2022) configured for 7nm FinFET technology with parameters matching the ASAP7 process assumptions.

Results. Table 9 summarizes the post-layout metrics from the SRAM-less synthesis combined with FN-CACTI estimates for the memory blocks. Both designs meet timing at the target frequency of 1.43 GHz corresponding to a 0.7 ns clock period with positive slack, and achieve maximum frequencies $f_{\max} > 1.5$ GHz. Core areas are 223 μm^2 encoder and 162 μm^2 decoder for the synthesized logic. Power consumption is approximately 0.8 mW encoder and 1.1 mW decoder at the typical process corner.

Table 9: Hardware implementation results (ASAP7 7nm). Logic from SRAM-less synthesis; SRAM estimates from FN-CACTI.

Metric	Encoder	Decoder
Target / $f_{\max}$ [GHz]	1.43 / 1.97	1.43 / 1.55
Logic area [μm^2]	222.9	162.4
Logic power [mW]	0.787	1.08
SRAM capacity [B]	128	256
SRAM area [μm^2]	56	105
SRAM access time [ns]	0.212	0.219
SRAM read energy [pJ]	0.0161	0.0285
Total area [μm^2]	**278.9**	**267.4**
Total power [mW]	**0.787**	**1.08**

The FN-CACTI-estimated SRAM overhead is modest: 56 μm^2 encoder and 105 μm^2 decoder—adding 25% and 65% respectively to the logic area. SRAM access times of 0.21–0.22 ns are comfortably below the 0.7 ns clock period (1/1.43 GHz), confirming that single-cycle table lookups are feasible at the target frequency. Read energy per access is 0.016–0.029 pJ, contributing negligible dynamic power relative to the pipeline logic.

Throughput analysis. With dual-lane interleaving and one-cycle initiation interval, both encoder and decoder sustain one symbol per cycle at steady state. At the achieved $f_{\max}$ values and 8 bits per symbol:

$$\text{Throughput}_{\text{enc}} = 1.97\,\text{GB/s}, \qquad \text{Throughput}_{\text{dec}} = 1.55\,\text{GB/s}$$

The hardware decoder achieves 1.55 GB/s—2.4× faster than the single-threaded software implementation (640 MB/s, Appendix B)—while consuming <1.1 mW. The encoder achieves 1.97 GB/s (6.6× faster than the 300 MB/s single-threaded software baseline). The encoder's higher throughput reflects its shorter critical path: the encoder performs a simple 128 B SRAM lookup in Stage 3, while the decoder performs a 256 B SRAM lookup with subsequent merge logic, resulting in a longer critical path and slightly lower throughput.

System-level implications. This combination of compact area (267–279 μm^2), low power (<1.1 mW), and multi-GB/s throughput per instance makes *Shannonic* suitable for bandwidth-constrained systems where computational resources for compression/decompression are limited. In scenarios requiring high aggregate bandwidth, multiple parallel instances can be deployed within modest silicon budgets while consuming negligible energy compared to data movement costs. For example, consider an edge device with LPDDR4X DRAM providing 34 GB/s peak bandwidth (JEDEC Solid State Technology Association, 2019): saturating this link would require 22 parallel decoders, occupying 0.0059 mm^2 total area and consuming 23.8 mW. The decompression energy overhead is approximately 0.70 pJ/byte (1.08 mW / 1.55 GB/s), which represents less than 6% of typical LPDDR4X read energy ∼12 pJ/byte (JEDEC Solid State Technology Association, 2019; Choi, 2024). Section 6 demonstrates how this efficiency enables practical deployment across diverse application contexts.

D Use Case 3: Multi-Tier Memory LLM Inference

As an additional use-case for *Shannonic* we consider running inference on memory constrained devices utilizing a solid-state storage device as active component during model execution.

Background and motivation. Large language models frequently exceed single-accelerator memory capacity, necessitating systems that virtualize across GPU HBM, CPU DRAM, and SSD. FlexGen (Sheng et al., 2023) exemplifies this approach, serving OPT-175B on a single NVIDIA T4 16GB with 208GB DRAM and 1.5TB SSD through coordinated tensor placement and I/O scheduling. At this scale, the 2GB/s SSD read bandwidth becomes the primary

bottleneck when weights or KV cache must reside on disk. FlexGen mitigates this via 4-bit group-wise quantization of weights and KV cache, at the cost of accuracy degradation. We investigate whether *Shannonic* can deliver comparable capacity benefits while preserving 8-bit accuracy through lossless compression.

FlexGen performance scaling. For OPT-175B with 512-token prompts and 32 generated tokens, FlexGen achieves 0.69 tokens/s permitting a batch size 256 when offloading to disk on the T4+208GB+SSD platform (Sheng et al., 2023). Applying 4-bit compression to both weights and KV cache fits all tensors in CPU DRAM, raising throughput to 1.12 tokens/s restricting batch size to 144, a 1.6× improvement that directly reflects the cost of disk I/O (Sheng et al., 2023).

Lossless compression for in-DRAM placement. *Shannonic* compresses 8-bit LLM weights to 4.25 bits on average CSR=0.53, Table 1 and activations to 5.5 bits CSR=0.69. For OPT-175B at INT8 precision 175GB, compression reduces weight footprint to approximately 93GB, enabling full residency within the 208GB DRAM alongside activations and working KV cache. This eliminates SSD accesses for weights. The inference pipeline streams compressed weights from DRAM, decompresses them on CPU cores, and transfers decompressed data to GPU over PCIe.

Sustaining PCIe bandwidth through parallel decompression. Since the weights are stored compressed in CPU RAM, executing the corresponding layers in the GPU is implemented by first decompressing the weights and then sending the uncompressed weights over PCIe to the GPU. Not all layers need to execute in the GPU, however, in the worst possible case for the scenario considered here, all layers need to be communicated. In this worst case, 10 threads of *Shannonic* software codec are sufficient for the PCIe 3.0 links assumed in the original study as representive of constrained systems (Sheng et al., 2023).

Summary. *Shannonic* enables in-DRAM weight placement through lossless compression, eliminating SSD weight I/O while maintaining 8-bit accuracy. FlexGen's 1.6× throughput improvement from removing disk offload (Sheng et al., 2023) establishes a realistic performance scale. For workloads where weights/activations benefit from compression to fit within DRAM budgets, *Shannonic* provides disk-avoidance benefits without quantization-induced quality loss. This is particularly valuable for applications where sub-8-bit quantization degrades accuracy. The codec's minimal per-layer overhead of 530 B (Section A) enables seamless integration with existing multi-tier inference frameworks.