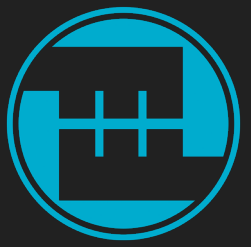
JEnt v2.2.0 LFSR Conditioning

Joshua E. Hill, PhD (KeyPair Consulting)
and
Yvonne Cliff, PhD (Teron Labs)

CMUF EWG
20251209 / 20260901

Meta Summary

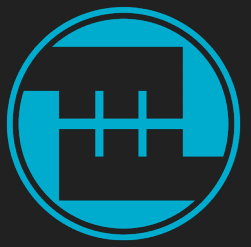
- This is a presentation of “JEnt v2.2.0 LFSR Conditioning Analysis” Technical Review Draft 20 [HC 2026].
- This paper is joint work with Yvonne Cliff (Teron Labs).
- This paper is still in development, but the parts presented today seem stable.
    - I think this is close to a v1.0 release.



TERON LABS

Dramatis Personae

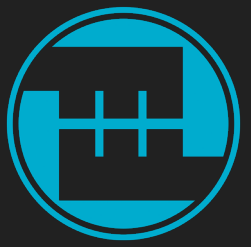
- JEnt v2.2.0 (released September 2019) has been widely integrated into many entropy sources.
  - e.g., #E8, #E19, #E20, #E37, #E47, #E48, #E50, #E54, #E59, #E60, #E61, #E62, #E90, #E99, #E117, #E151, #E174, #E175, #E226, #E235, #E315, #E325, #E328, #E337.
  - Many Linux kernel versions include a JEnt version that is based on JEnt v2.2.0.
- JEnt v2.2.0 uses an LFSR for conditioning.
- There is no public analysis of the LFSR conditioning to support ESV testing.
  - Testing requires mathematical evidence for various properties (see [SP 800-90B §3.2.3, Requirement 5] and [IG D.K, Resolution 5]; note that [NIST SHALL ID #52] is marked “Optional”, but [NIST SHALL IDs #106-#107] are marked as “Required”.)



TERON LABS

In Summary, We…

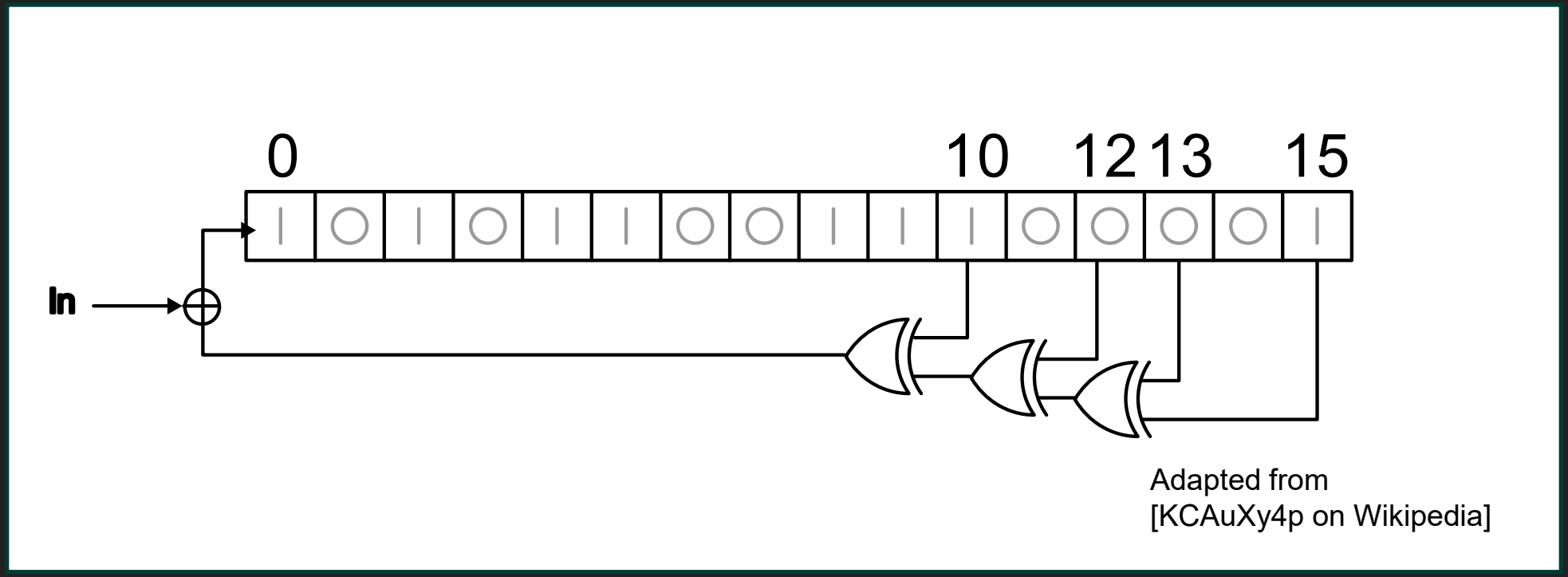
- Present a linear model for this conditioning
  - Use this model to verify various characteristics of the conditioning.
  - Verify that this model is equivalent to the implemented conditioning.
- Present a heuristic analysis using this model to obtain a conditioned output min entropy estimate.
- Provide the distribution of statistical entropy assessment results for most non-vetted conditioning functions.
- Discuss the JEnt v2.2.0 SP 800-90B conditioning analysis.



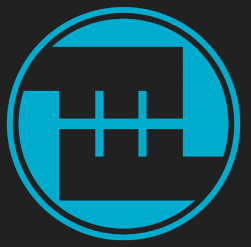
TERON LABS

LFSRs

- An LFSR is a shift register with linear (i.e., XOR-based) feedback.
- The JEnt v2.2.0 conditioning function uses a Fibonacci LFSR, e.g. something like:



Adapted from
[KCAuXy4p on Wikipedia]



TERON LABS

The JEnt v2.2.0 LFSR

- The LFSR is 64 bits wide and is used in multiplicative scrambler mode (i.e., takes an external input that is XORed into bit 0 of the LFSR)
- A single bit is fed in at a time until all 64 bits of the raw symbol are integrated into the LFSR.
- After sufficient ($64 \times$ osr) raw symbols are thus integrated, the entire internal state is output as conditioned data.
- The “taps” used in this LFSR are based on a primitive feedback polynomial:

$$p(z) = z^{64} + z^{61} + z^{56} + z^{31} + z^{28} + z^{23} + 1$$

- The LFSR has “good” structure, and loops through either 1 state (for the state value 0) or $2^{64} - 1$ states (for all other states).

TERON LABS

The JEnt v2.2.0 LFSR

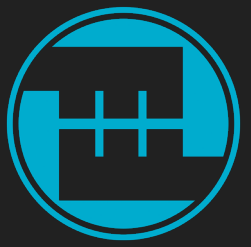
LFSRs are **linear** so the impact of the initial state and the input values can be broken apart by additivity:

$$\boldsymbol{f(s, x) = f(s, 0) + f(0, x)}$$

and each of these transforms can be represented using matrix multiplication:

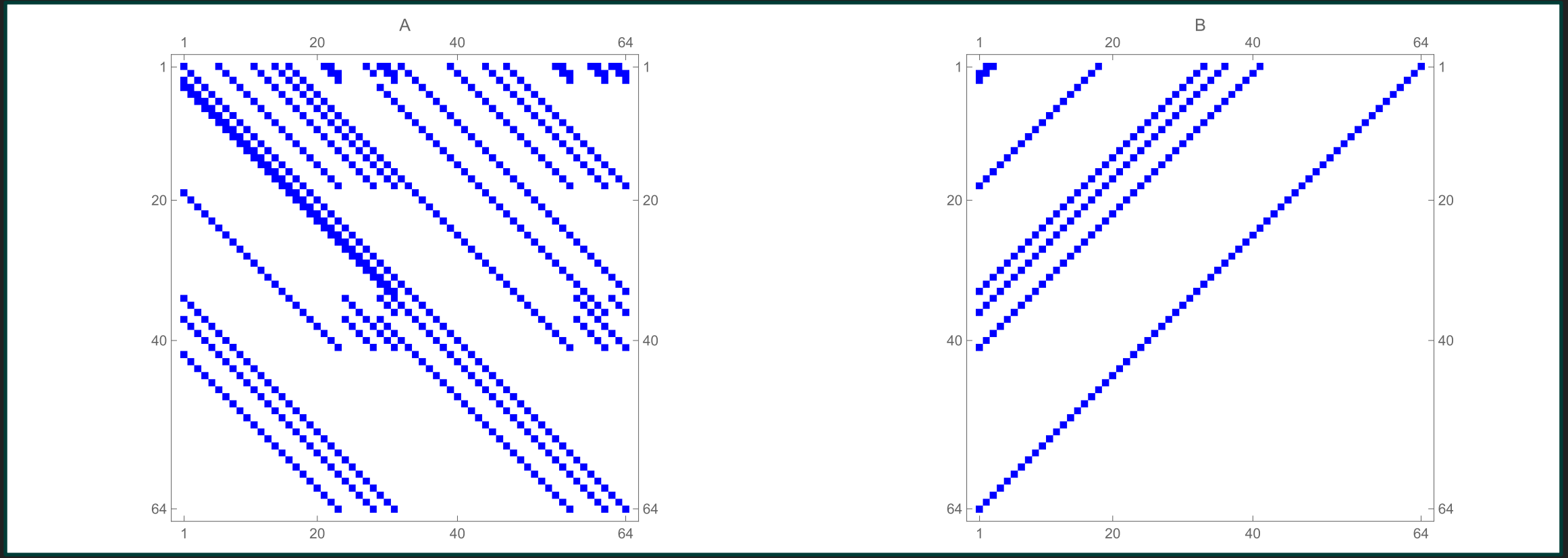
$$\boldsymbol{f(s, x) = As + Bx.}$$

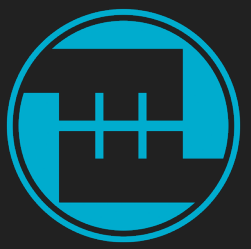
Further, we can find the explicit values for the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.



TERON LABS

The JEnt v2.2.0 LFSR

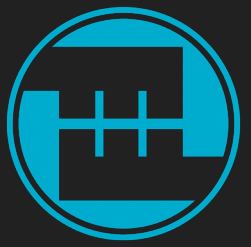




TERON LABS

The JEnt v2.2.0 LFSR

- So now we can implement JEnt v2.2.0 conditioning… really slowly?
- We can directly verify some LFSR characteristics (e.g., the order of the LFSR)
- We can also… <handwave>reason about the conditioning</handwave>!



TERON LABS

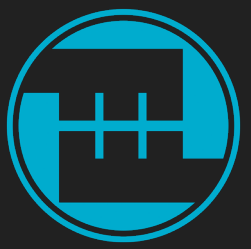
The JEnt v2.2.0 LFSR

We can make a recurrence relation:

$$\begin{aligned} \boldsymbol{o}_j(\boldsymbol{s}_j, \boldsymbol{x}_j) &= \boldsymbol{f}(\boldsymbol{s}_j, \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_j) \\ &= \boldsymbol{f}(\boldsymbol{o}_{j-1}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_{j-1}), \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_j) \\ &= \boldsymbol{A}\boldsymbol{o}_{j-1}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_{j-1}) + \boldsymbol{B}\boldsymbol{x}_j \end{aligned}$$

Thus…

$$\boldsymbol{o}_j(\boldsymbol{s}_1, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_j) = \boldsymbol{A}^j \boldsymbol{s}_1 + \sum_{k=1}^{j} \boldsymbol{A}^{j-k} \boldsymbol{B} \boldsymbol{x}_k$$



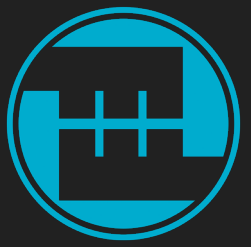
TERON LABS

The JEnt v2.2.0 LFSR

- This makes it clear what our matrices are accomplishing.
  - $\boldsymbol{A}$ performs 64 LFSR operations on the current internal state.
  - $\boldsymbol{B}$ deals with the loading of input data.
- We also see that every input is iteratively LFSR processed by a fixed function:

$$\boldsymbol{g}_j(\boldsymbol{x}) = \boldsymbol{A}^j \boldsymbol{B}\, \boldsymbol{x}$$

- Both the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible (+ some linear algebra) so $\boldsymbol{g}_j(\boldsymbol{x})$ is a bijection.



TERON LABS

Record Scratch

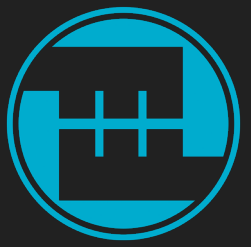
$$g_j(x) = A^j B\, x$$

is a bijection, but repeated XOR is **NOT bijective**, so

$$o_j(s_1, x_1, x_2, \ldots x_j) = A^j s_1 + \sum_{k=1}^{j} g_{j-k}(x_k)$$

is not bijective.

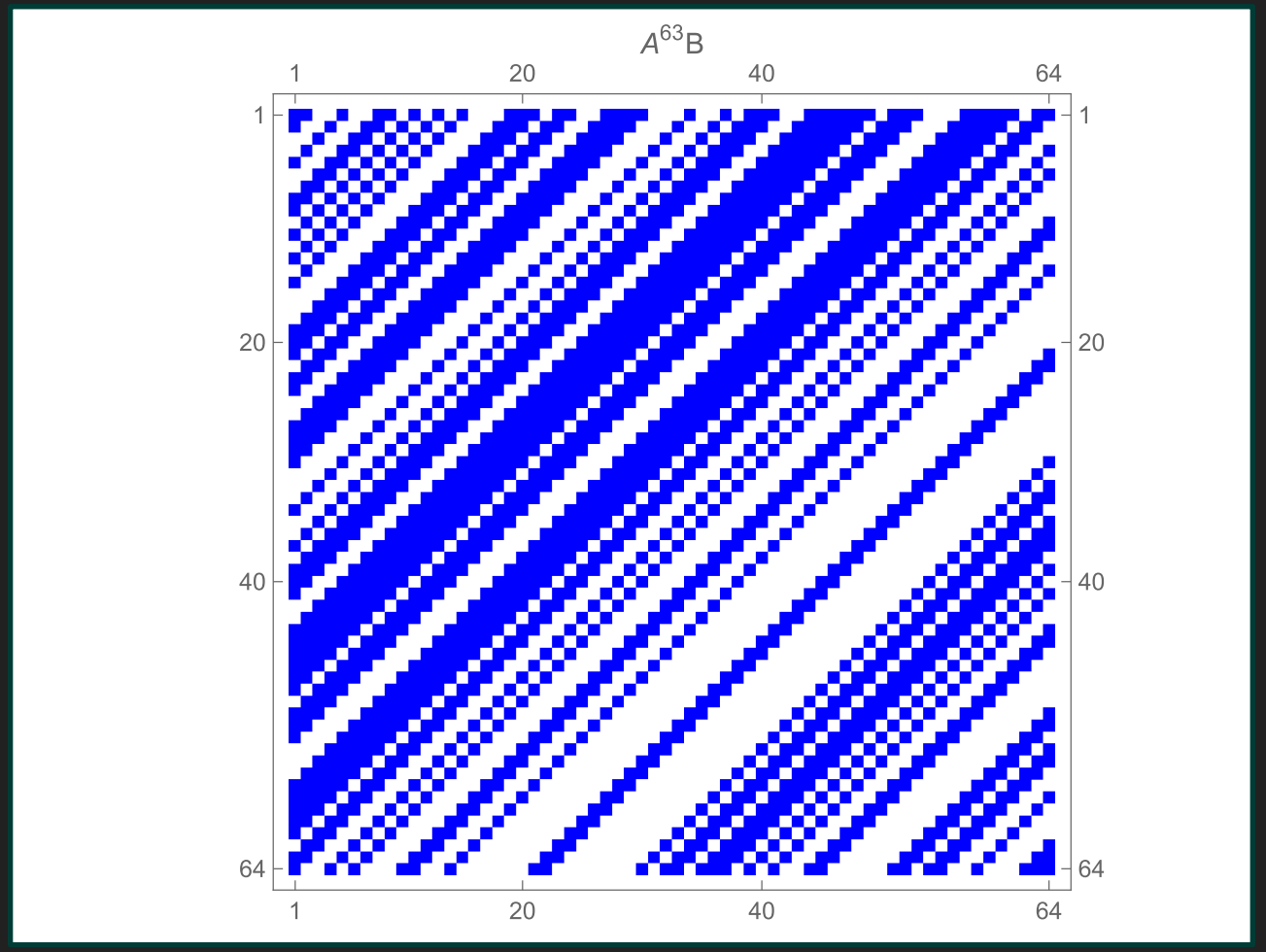
(This is good, as otherwise we could not accumulate entropy!)

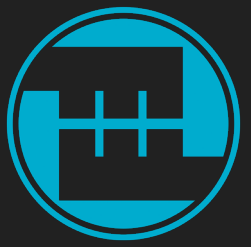


TERON LABS

The JEnt v2.2.0 LFSR

- We use matrices of the form $\boldsymbol{A^j B}$ so we are interested in their form, e.g.

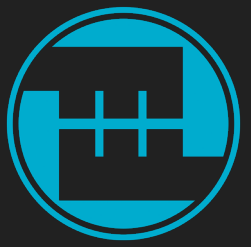




TERON LABS

The JEnt v2.2.0 LFSR

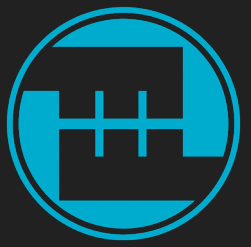
- All such matrices are symmetric.



TERON LABS

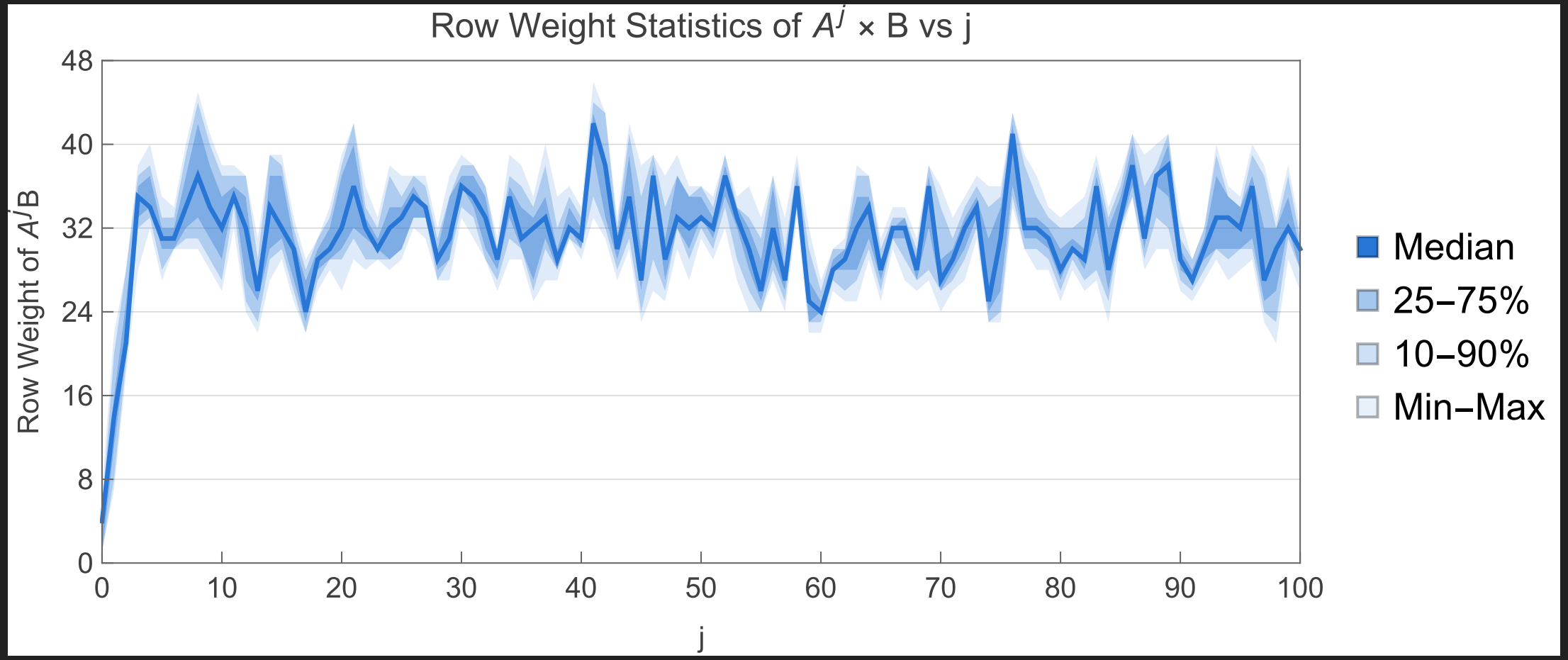
Heuristic Conditioning Min Entropy Analysis

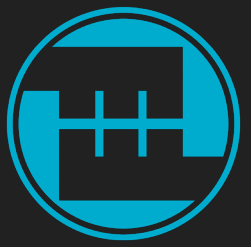
- The function $g_j(x)$ is a bijection, so the entropy for each term in the sum is fixed.
- Iterative application of additional LFSR processing spreads out the impact of each input bit.



TERON LABS

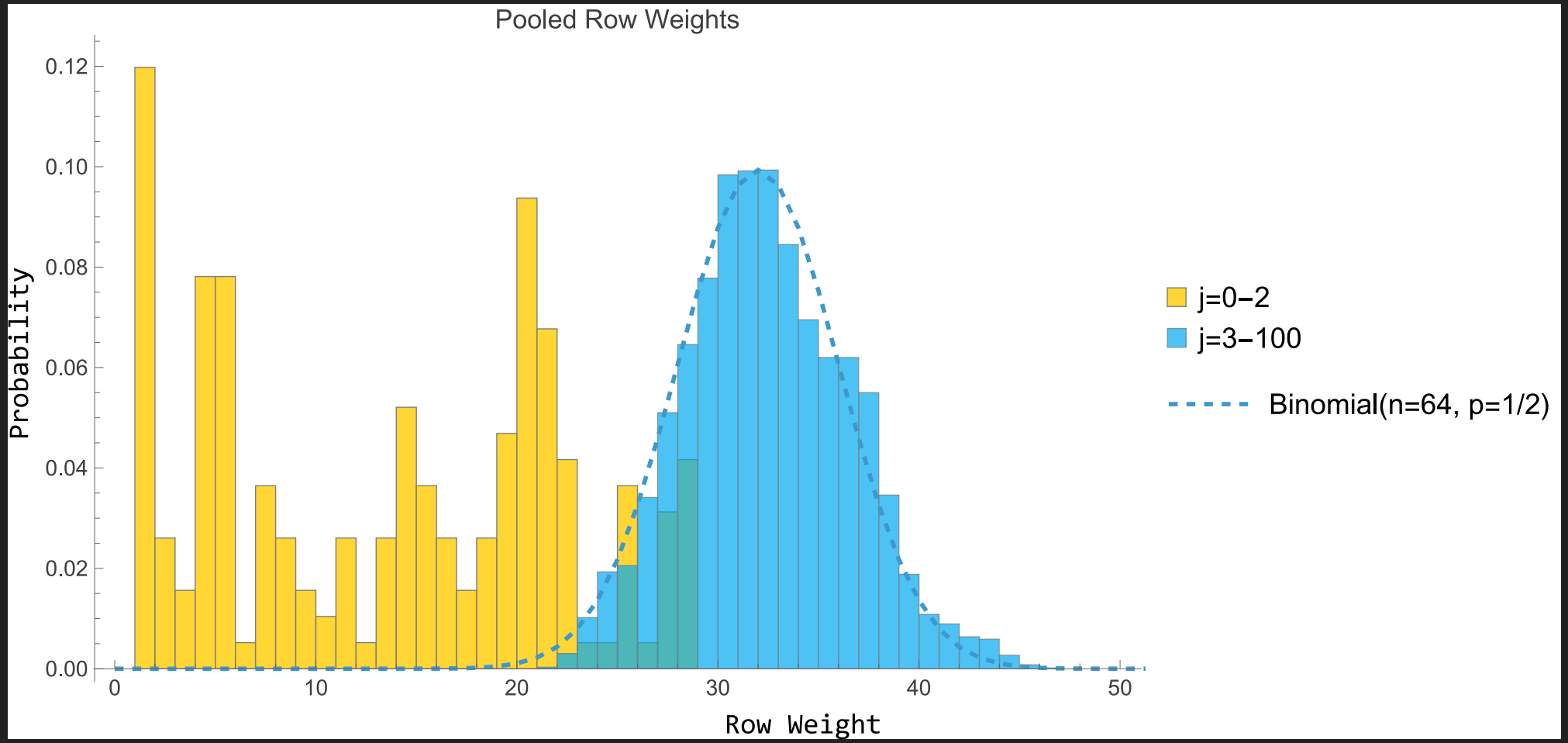
Heuristic Conditioning Min Entropy Analysis

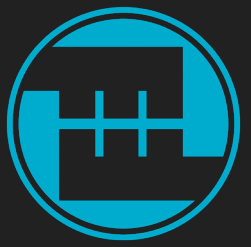




TERON LABS

Heuristic Conditioning Min Entropy Analysis



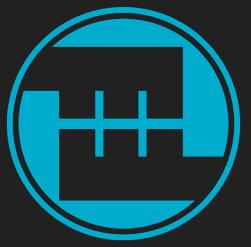


TERON LABS

Heuristic Conditioning Min Entropy Analysis

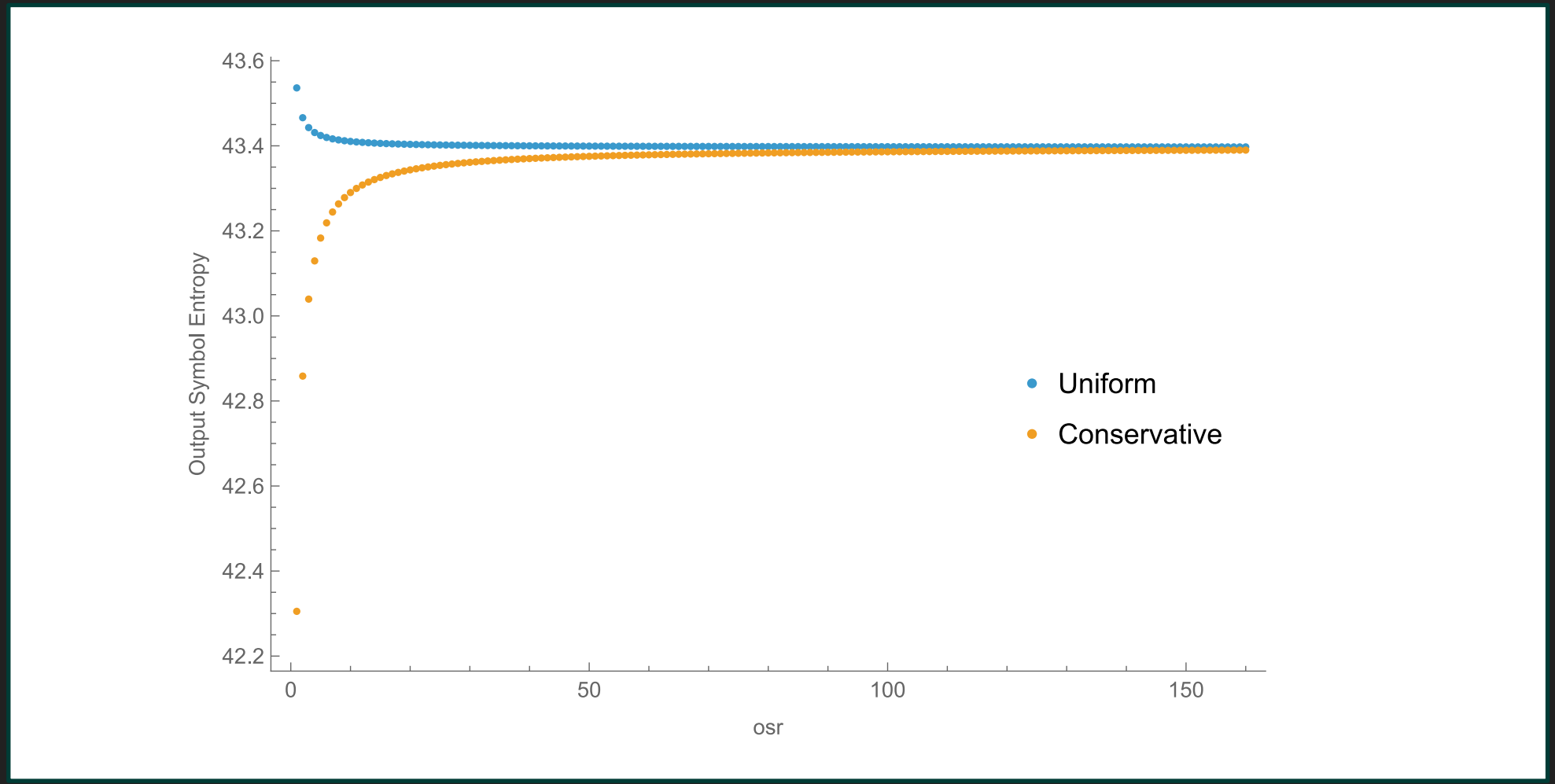
- By hypothesis, the min entropy for each raw symbol is $\geq H' \geq \frac{1}{\mathtt{osr}}$.
- After a few iterations (≥ 3), the entropy has been spread throughout the state.
- As such, we can model iteratively XORing w together bits, each with at least $\frac{H'}{64}$ bits of min entropy and then scale to the full 64 bits.

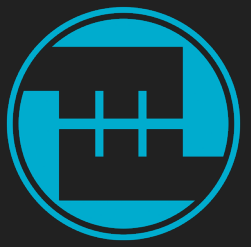
$$h^{\text{uniform}}_{\text{heuristic}}(H', w) = 64\left(1 - \log_2\left(\left(2^{1-H'/64} - 1\right)^w + 1\right)\right)$$



TERON LABS

Heuristic Conditioning Min Entropy Analysis





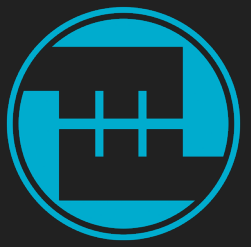
TERON LABS

Heuristic Conditioning Min Entropy Analysis

The `osr=1` case (discarding the entropy of the last three symbols) is the worst case:

$$h_{\text{heuristic}} \geq 42.3$$

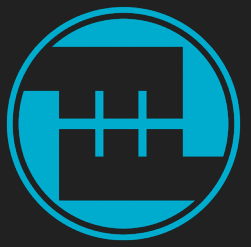
per 64-bit conditioned output block (assuming $H' \geq \frac{1}{\texttt{osr}}$)



TERON LABS

SP 800-90B Non-Vetted Conditioning Analysis

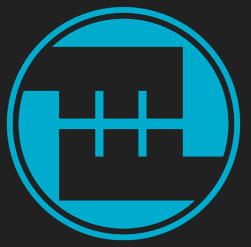
- The JEnt conditioning chain contains two conditioning components:
  - The “stuck” filter.
  - The conditioning LFSR
- The “stuck” filter has historically been ignored but it can be important.



TERON LABS

The Stuck Filter

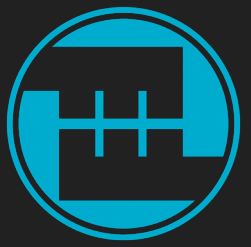
- In JEnt versions v2.2.0 – v3.6.3, only “non-stuck” raw values are provided to the conditioner
- The “non-stuck” values necessarily meet the following conditions:
    1. Non-zero ($\Delta_j \neq 0$; the first difference of the timer is non-zero). On most hardware, the raw data distribution does not include 0, so this should not have any practical impact for hardware that supports JEnt use. Additionally, JEnt’s initialization testing produces an error if it encounters any raw values that equal 0.
    2. Not the same as the prior raw data sample ($\Delta_j \neq \Delta_{j-1}$; the second difference of the timer is non-zero). This precludes integrating runs of the same symbol into the conditioner.
    3. The change between the raw values cannot be fixed ($\Delta_j - \Delta_{j-1} \neq \Delta_{j-1} - \Delta_{j-2}$; the third difference of the timer is non-zero). This precludes changes in the raw data by fixed amounts.



TERON LABS

The Stuck Filter: A Toy Example

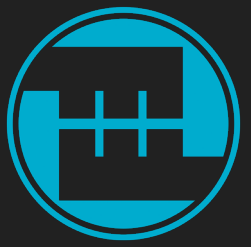
- Imagine having an IID noise source that outputs exactly two symbols with equal probability.
- Outputs from this noise source would have a min entropy of exactly 1 bit.
- When passing such data through the stuck filter, Stuck Filter Condition 2 will prevent whichever symbol was last output to the conditioner from being repeated.
- In this case, the filtered data can only be a fixed string that alternates between the two possible symbols.
- Even though we started with a stream that had entropy, the post-stuck-filtered min entropy is 0.



TERON LABS

The Stuck Filter: A Toy Example

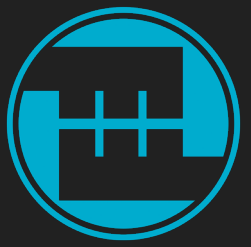
- This is not how our noise source works, but it serves as a good warning.
- We can bound any such entropy loss for IID sources.
  - These bounds are coarse and have limited applicability to actual sources.
  - The output of this filter induces dependence.
- It makes more sense to do a separate analysis on the post-filtered datasets.



TERON LABS

The Stuck Filter

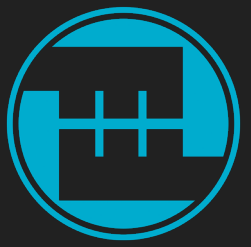
- This “discard stuck samples” behavior was removed in JEnt v3.7.0.
- This behavior was introduced in the JEnt v2.1.2 to v2.2.0 transition so this filtered data finding applies to JEnt versions v2.2.0 – v3.6.3.



TERON LABS

The Stuck Filter

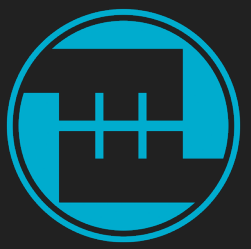
- Stuck Filter Condition 1 is unlikely to have any impact for any supported hardware
  - This would suggest that the timer resolution is much too slow to observe changes in system behavior.
- Stuck Filter Conditions 2-3 have the practical impact of excluding exactly 1 or 2 values for each raw symbol (namely, $\Delta_j \neq \Delta_{j-1}$ and $\Delta_j \neq 2\Delta_{j-1} - \Delta_{j-2}$).
- This test precludes exactly 1 value if and only if the prior raw symbol was itself a repeat (i.e., if $\Delta_{j-1} = \Delta_{j-2}$)
- Stuck Filter Conditions 2-3 otherwise preclude exactly 2 distinct symbols.
- Runs of fixed raw values trigger both Stuck Filter Condition 2 and Stuck Filter Condition 3.



TERON LABS

The Stuck Filter: Impacts

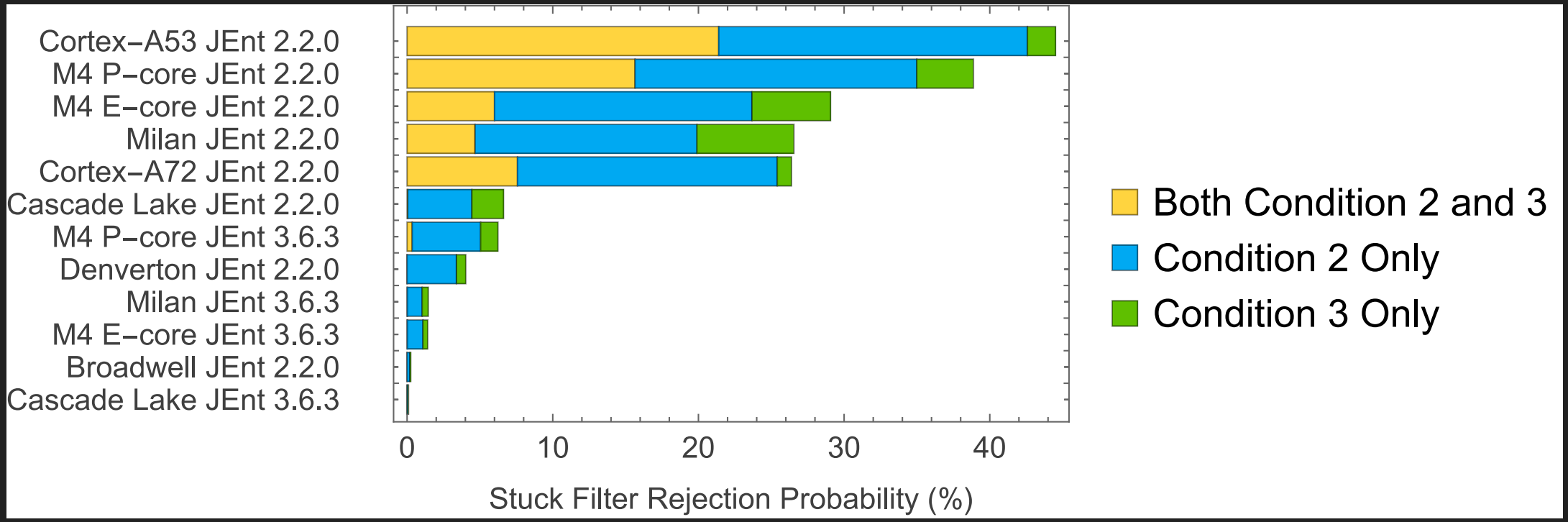
Architecture	JEnt Version	Raw Data 90B Tool Estimate	Average m	Filtered Data 90B Tool Estimate
ARM Cortex-A53	2.2.0	0.110118	0.45	0.131213
ARM Cortex-A72	2.2.0	0.126359	0.26	0.0907289
Intel Atom Denverton	2.2.0	0.614900	0.040	0.698771
Intel Xeon Broadwell	2.2.0	5.53007	0.0025	5.52947
Intel Xeon Cascade Lake	2.2.0	0.134489	0.066	0.229460
	3.6.3	7.50837	0.00097	7.52045
AMD EPYC Milan	2.2.0	1.17812	0.27	1.19899
	3.6.3	4.36032	0.015	4.38333
Apple M4 E-core	2.2.0	0.930320	0.29	0.372956
	3.6.3	4.73669	0.014	4.92845
Apple M4 P-core	2.2.0	0.586240	0.39	0.286127
	3.6.3	2.60697	0.062	2.64165

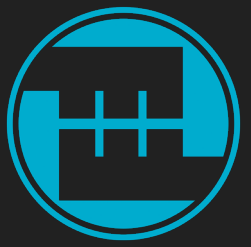


TERON LABS

The Stuck Filter

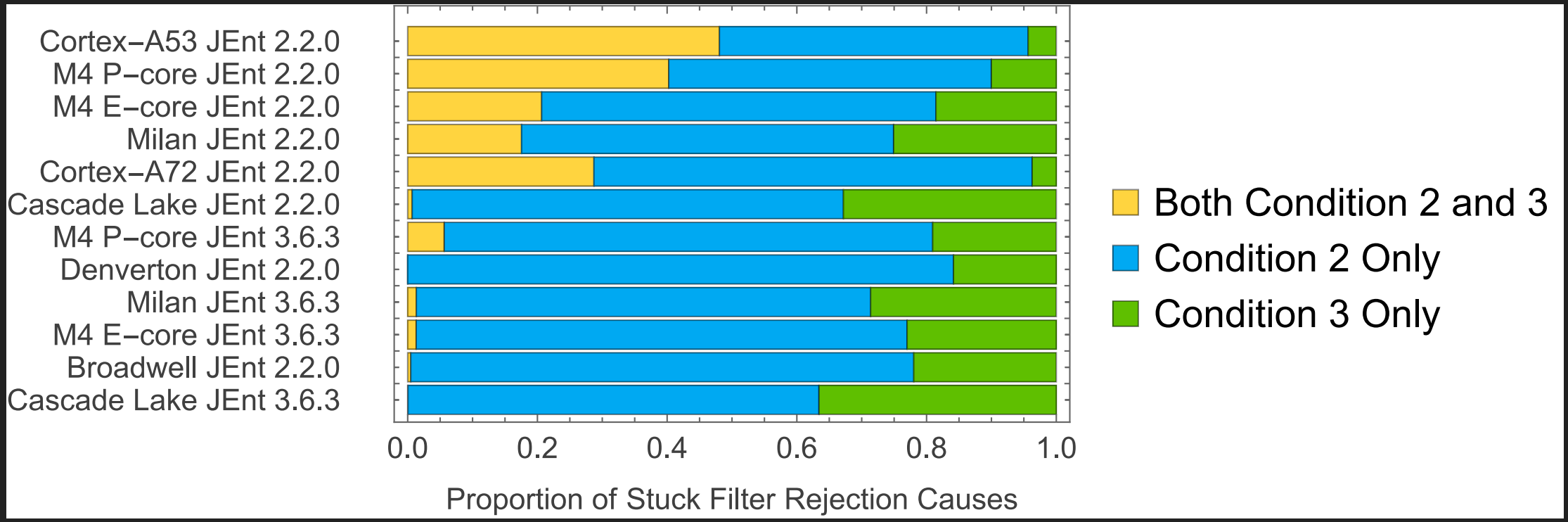
- We can examine actual data sets and get a sense for the likelihood that each condition excludes data.

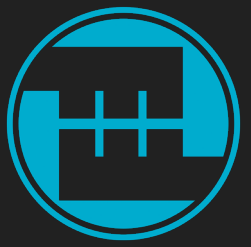




TERON LABS

The Stuck Filter: Relative Weights





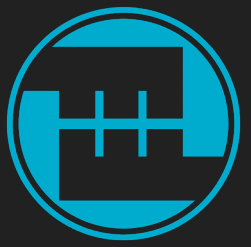
TERON LABS

SP 800-90B Non-Vetted Conditioning Analysis

- Combining [SP 800-90B §3.1.5.2] and [IG D.K, Resolution 5], we get a formula like:

$$h_{\text{out}} = \min\left(\text{Output\_Entropy}\left(n_{\text{in}}, n_{\text{out}}, nw, h_{\text{in}}\right), 0.999 \times n_{\text{out}}, h' \times n_{\text{out}}, h_{\text{heuristic}}\right)$$

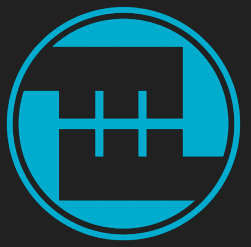
- The smallest of these terms dictates h_{out}.



TERON LABS

Stuck Filter Conditioning Analysis

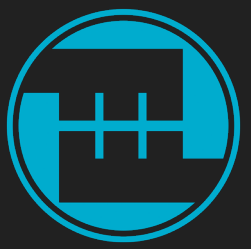
- It is important to assess the post-filtered data.
- This is probably best conceptualized as a first stage of non-vetted conditioning.
- The behavior is stochastic, so parameters need to be conservatively bounded.



TERON LABS

Stuck Filter Conditioning Analysis

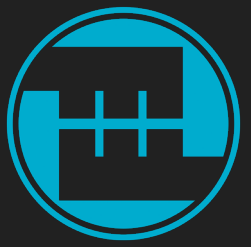
Parameter	Value
Symbol Width (n)	64
n_{out}	64
nw $(\leq n_{\text{in}})$	64
$n_{\text{in}}(w \times n)$	$\geq 1 \times 64$
$h_{\text{in}} = w \times H$	$\geq \mathrm{H}$
Output_Entropy$(\cdot)$	$\geq \mathrm{H}$



TERON LABS

Stuck Filter Conditioning Analysis

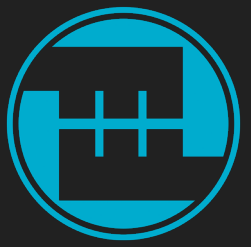
- An independent assessment needs to occur for the post-filtered data.
- Due to the stochastic nature of this filtering we are forced to use parameters that result in the inequality chain $h_{out} \leq h_{heuristic} \leq H'$, so the SP 800-90B assessment procedure precludes crediting entropy increases for this conditioning step.
- It is probably best to disable this functionality moving forward.



TERON LABS

LFSR Non-Vetted Conditioning Analysis

- Some of these are not likely to be the minimum here.
    - $0.999 \times n_{\text{out}}$ is essentially never the minimum of these expressions, as we necessarily have $h' < 0.999$ for any reasonable size of conditioned sequential dataset.

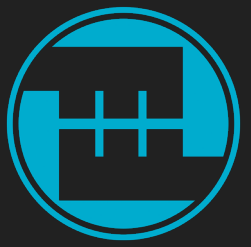


TERON LABS

LFSR Non-Vetted Conditioning Analysis

- Output_Entropy(·) Parameters:

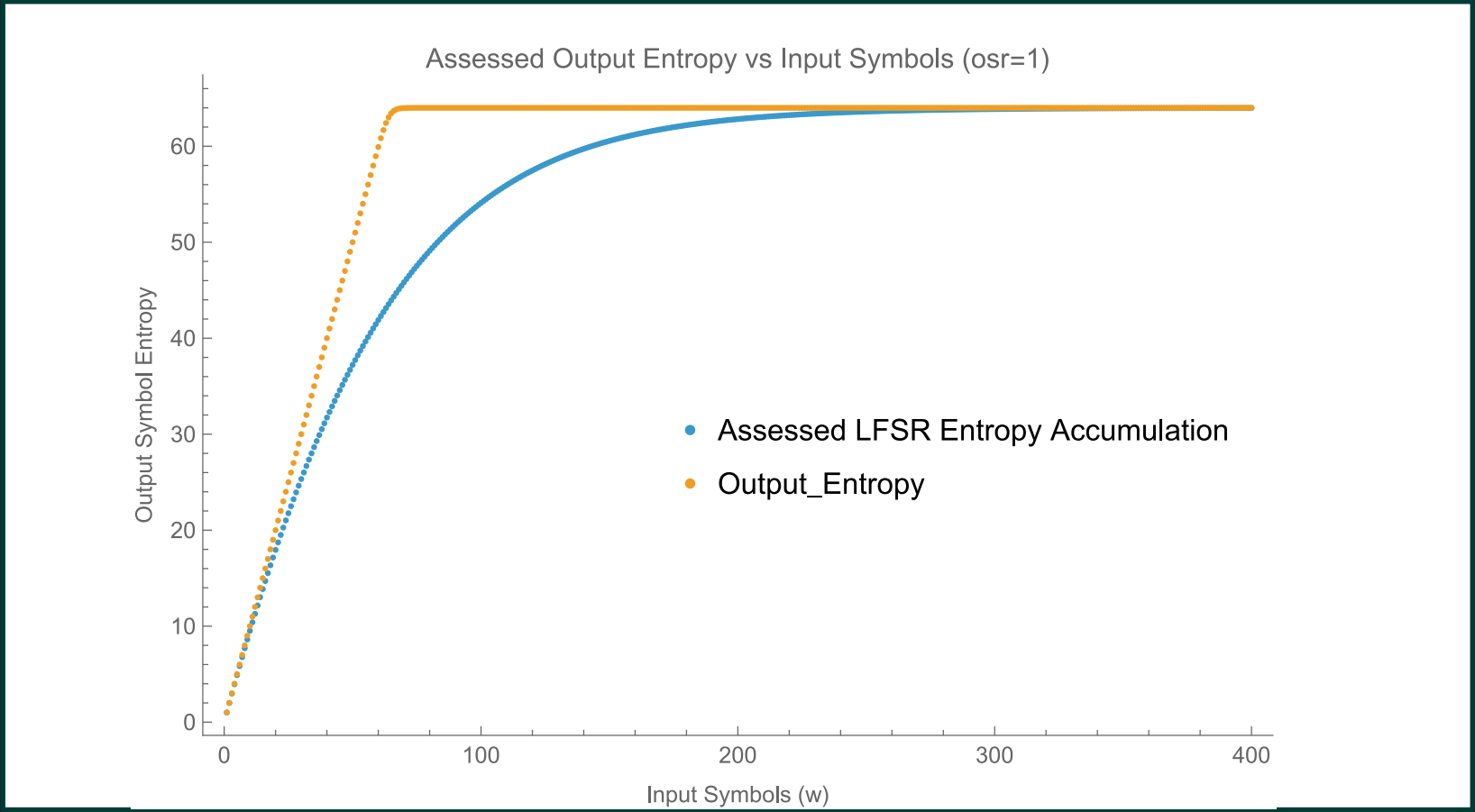
Parameter	Value
Symbol Width (n)	64
n_{out}	64
nw ($\leq n_{\text{in}}$)	64
$n_{\text{in}}(w \times n)$	$w \times 64$
H	$1/\text{osr}$
$h_{\text{in}} = w \times H$	$\geq w/\text{osr}$

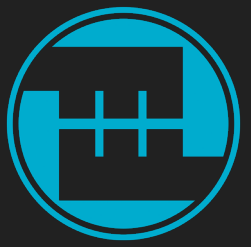


TERON LABS

LFSR Non-Vetted Conditioning Analysis

- Output_Entropy(·) Results:

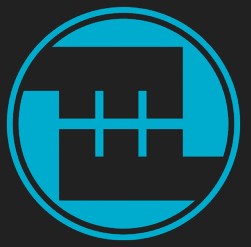




TERON LABS

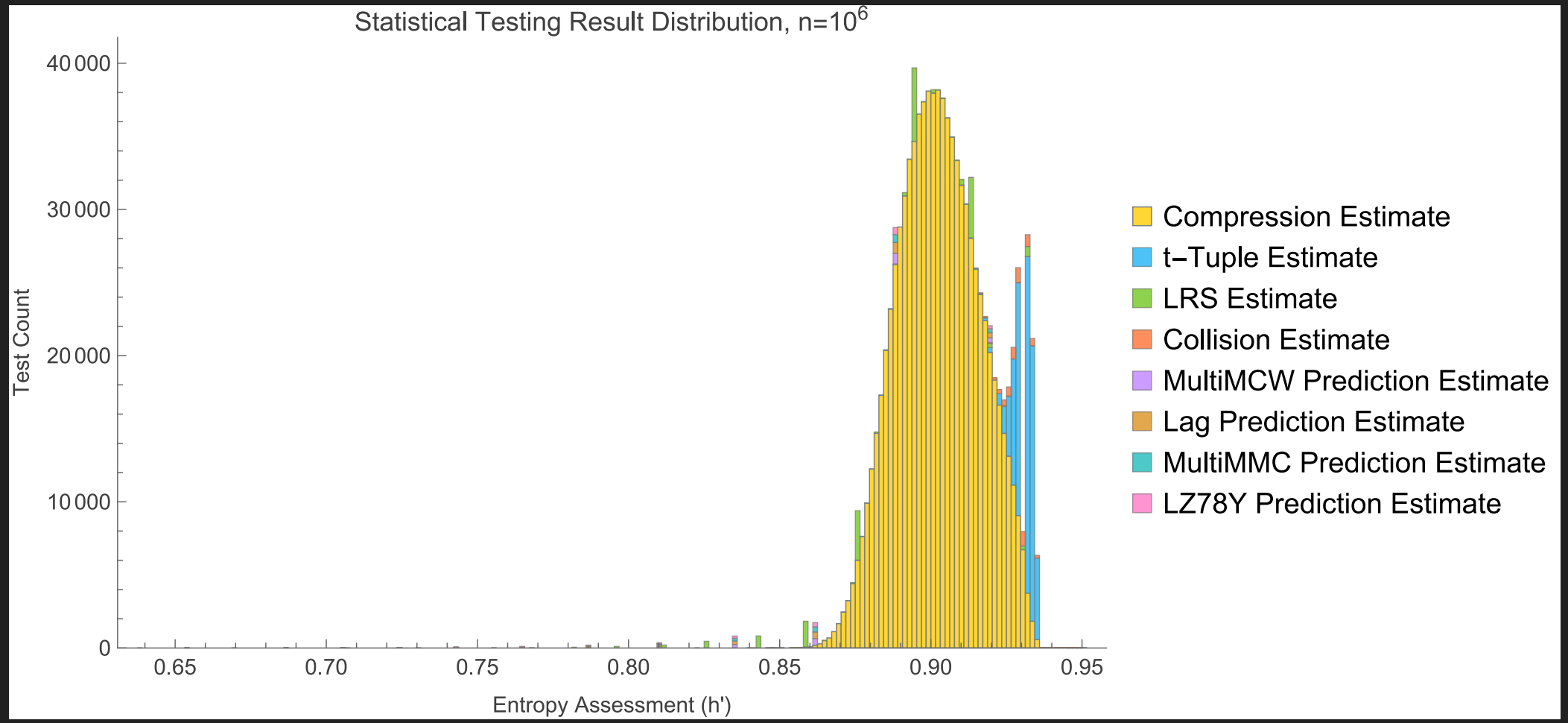
LFSR Non-Vetted Conditioning Analysis

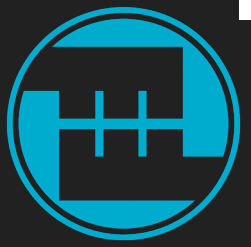
- The LFSR's output looks pseudorandom.
- Evaluation of pseudorandom data provides a practical "best case" for the h' term.
- This "best case" is essentially attained by any conditioner whose output is pseudorandom (which is most conditioners).
- The ESV program allows for submission of up to 1 million-byte samples (thus 8 million bits).
- With this size and type of data, this estimation approach yields a consistent distribution.



TERON LABS

h' Histogram

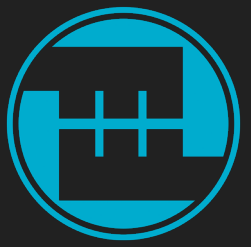




TERON LABS

h' Histogram

- The Compression Estimator provides the lower bound most frequently (89% of the time), but its result distribution looks reasonable.
- The t-Tuple test accounts for about 8% of the results.
  - 98.7% of these results are one of 8 distinct values.
  - Why?

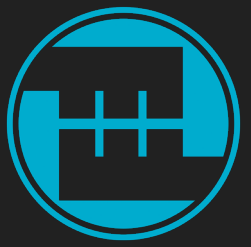


TERON LABS

Remember the t-Tuple!

- t is the length of the longest substring that repeats at least 35 times
- $Q[i]$ is the number of occurrences of the most common i-length substring in the data
- $i_{\max}$ is the index ($1 \leq i_{\max} \leq t$) associated with the largest normalized per-symbol probability.

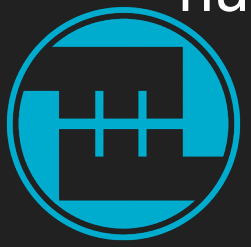
Entropy Estimate (Data Sample Atom)	Observed Mass (%)	t	$i_{\max}$	$Q[i]$
0.93570634778060169	0.5561	19	19	35
0.9335692443037808	1.8837	19	19	36
0.93149069917309046	2.3024	19	19	37
0.9294675887066988	1.5923	19	19	38
0.9274970326834304 2	0.8625	19	19	39
0.9255763696816026 2	0.4103	19	19	40
0.9237031354634257 5	0.1840	19	19	41
0.9218750439644997 1	0.0794	19	19	42



TERON LABS

Still More t-Tuple Battle Cries

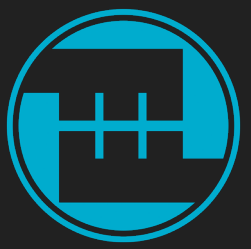
- These repeated results occur because of the construction of this estimator.
- In pseudorandom strings, the counts of such “longest repeated substrings of length i” is somewhat stable
- By construction, the number of times the longest repeated substring is observed ($Q[t]$) is likely to be close to 35.
- The cases where the t-Tuple produces the lowest min entropy estimate are often associated with the longest observed substring of length t.
    - All of the observed t-Tuple assessment spikes reflect this occurrence.
- This estimator takes a small set of discrete parameters and produces a small number of possible estimates



TERON LABS

$h' \times n_{\text{out}}$ for 64-Bit Blocks of Random Data

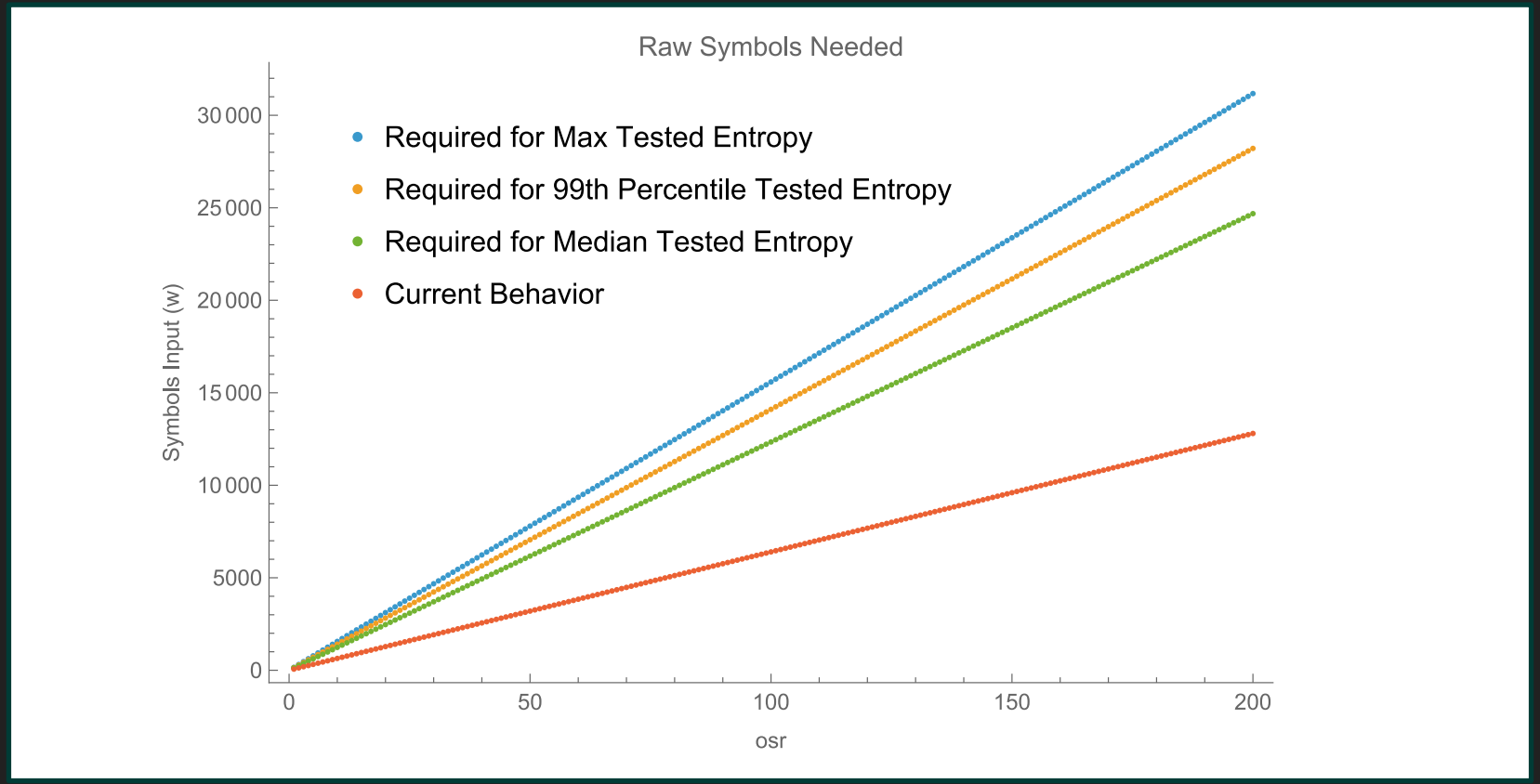
Probability the Result is in the Bound (%)	Result Interval
95	[56.0531, 59.7484]
99	[54.9680, 59.8852]
99.9	[50.3425, 59.8852]

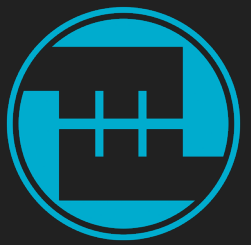


TERON LABS

Alternate values for w

- Now that we understand the distribution for $h' \times n_{\text{out}}$, we can set w to alter the chance that h_{out} is established by the heuristic estimate (which only happens when $h' \times n_{\text{out}} > h_{\text{heuristic}}$).

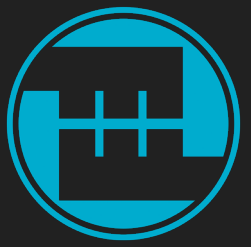




TERON LABS

SP 800-90B Non-Vetted Conditioning Analysis

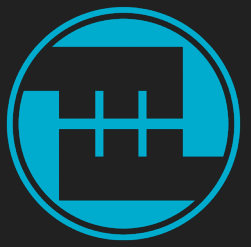
Approach	Chance of $h_{\text{heuristic}}$ Reducing h_{out}	Scaling Factor (S)	$w = [S \times 64] \times osr$	Average h_{out}	Normalized Efficiency ($E_{\text{normalized}}$)
Original Behavior	$\approx 100\%$	1	$w = 64 \times \text{osr}$	42.3053	1
$h_{\text{heuristic}}$ is likely above $h' \times n_{\text{out}}$	$\leq 50\%$	1.96875	$w = 126 \times \text{osr}$	57.4610	0.689904
$h_{\text{heuristic}}$ is very likely above $h' \times n_{\text{out}}$	$\leq 1\%$	2.25000	$w = 144 \times \text{osr}$	57.8775	0.608041
$h_{\text{heuristic}}$ is always above $h' \times n_{\text{out}}$	$\approx 0\%$	2.48438	$w = 159 \times \text{osr}$	57.8780	0.550683



TERON LABS

Comments on Efficiency

- The best rate of entropy per unit time is attained by setting w as small as possible.
- Higher claims (min entropy per 64 bit block of conditioned data) are possible, but they decrease the amount of entropy per unit time.

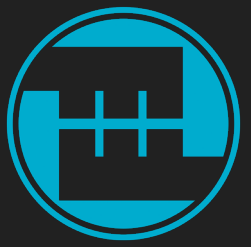


TERON LABS

Wait… What Were We **Just** Talking About?

We…

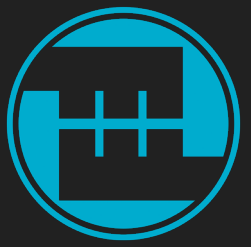
- Presented a linear model for this conditioning.
  - This was just a mathy way of conceptualizing the conditioning.
- Presented a heuristic analysis using this model
- Highlighted the impact of the stuck filter on the entropy analysis.
- Provided a broadly-applicable distribution of statistical entropy assessment results that applies to most non-vetted conditioning functions.
- Described changes to JEnt v2.2.0 that allow for higher conditioned output block min entropy claims (but at WHAT COST? AT WHAT COST?!?)



TERON LABS

References

- [IG] CMVP. *Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program*. NIST, August 19, 2026. https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf.
- [NIST SHALL] NIST CAVP. *90B-Shall-Statements*. NIST, August 9, 2021. https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/esv/90B%20Shall%20Statements.xlsx.
- [HC 2026] Joshua E. Hill and Yvonne Cliff. *JEnt v2.2.0 LFSR Conditioning Analysis*. TRD20. https://www.untruth.org/~josh/sp80090b/jent-lfsr/JEnt%20v2.2.0%20LFSR%20Conditioning%20Analysis%20TRD20.pdf
- [SP 800-90B] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish and Mike Boyle. *NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation*. NIST, January 2018. https://doi.org/10.6028/NIST.SP.800-90B.



TERON LABS