The 23rd Annual International Conference on Information Security and Cryptology

ICISC 2020

December 2 (Wed) ~ December 4 (Fri), 2020 | Virtual Conference

Hosted by
Korea Institute of Information Security and Cryptology (KIISC)
National Security Research Institute (NSR)

Korea Institute of Information Security & Cryptology

Generative Adversarial Networks-Based Pseudo-Random Number Generator for Embedded Processors

Hyunji Kim, Yongbeen Kwon, Minjoo Sim, Sejin Lim, Hwajeong Seo

IT Department, Hansung University, Seoul, Korea

Contents

Motivation and Contribution

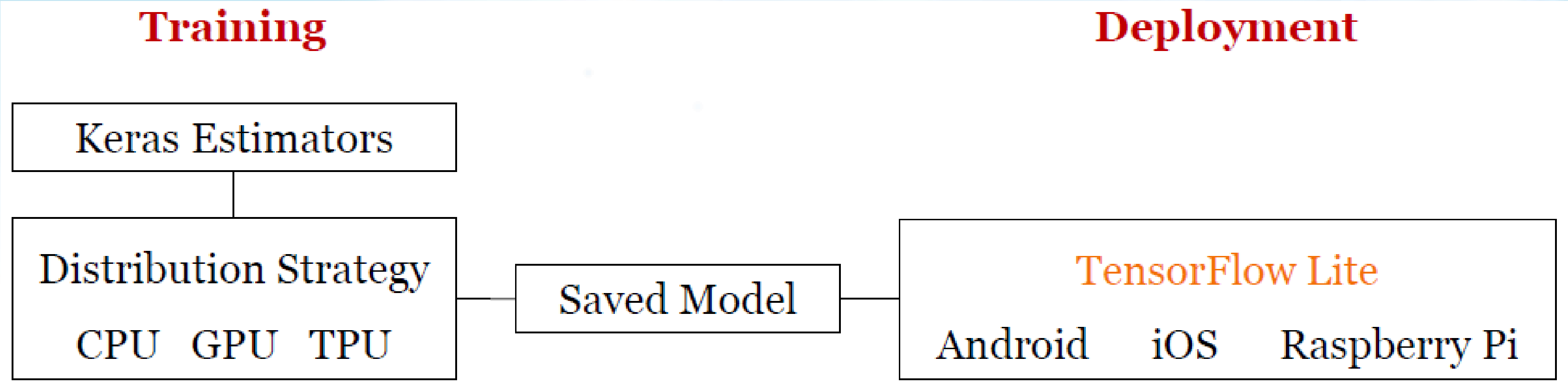
- **Motivation**
  - Improve the randomness of the previous work.
  - Let’s make the Cryptographically Secure Pseudo Random Number Generator, CSPRNG) for Embedded Processor.

- **Contribution**
  - Novel GAN based PRNG (DRBG) mechanism design for embedded processors.
  - High randomness validation through NIST test suite.

Random Number Generator

- **Random Number Generator (RNG)**
  - Produce a sequence of numbers that cannot be predicted better than by a random chance.
- **True Random Number Generator (TRNG)**
  - Must produce unpredictable bits even if every detail of the generator is available.
- **Pseudo Random Number Generator (PRNG)**
  - Deterministic Random Bit Generator (DRBG)
    : Generate random numbers by producting the random sequence with perfect balance between 0's and 1's.

TensorFlow and TensorFlow Lite

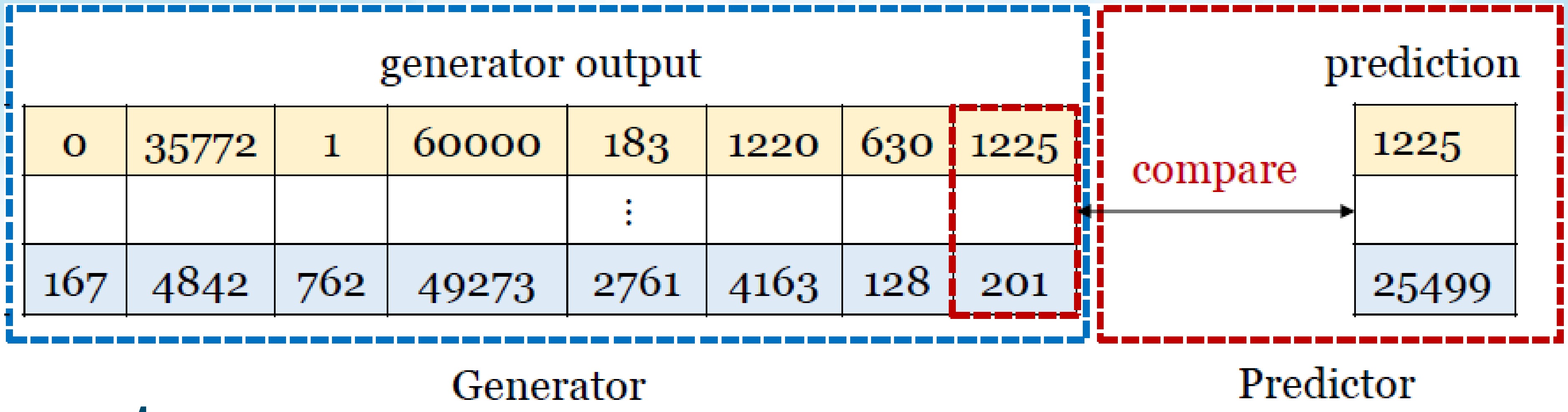


- **TensorFlow**
  - Open-source software library for machine learning applications, such as neural networks.
- **TensorFlow Lite**
  - Official framework for running TensorFlow model inference on edge devices.

Edge TPU

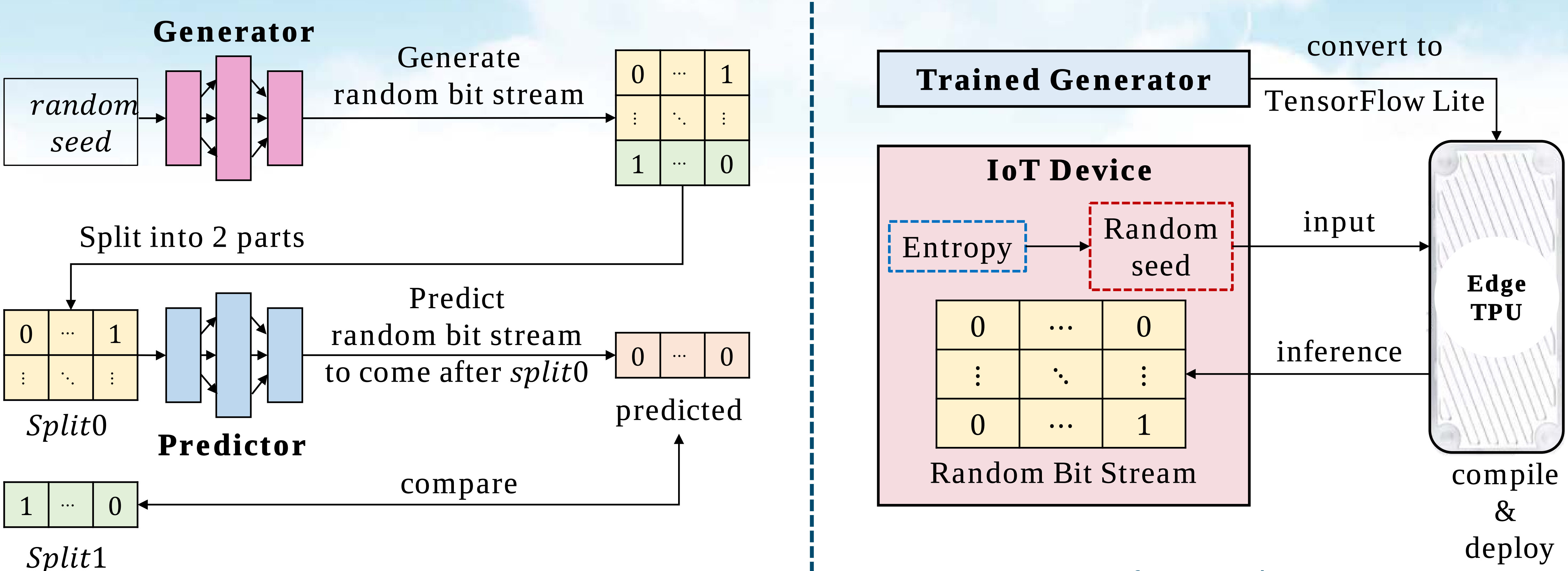
- USB type hardware accelerators.
- ASIC designed to run inference at the edge.
- Support the TensorFlow Lite.
- Small footprint, low power.

Previous GAN based PRNG Implementation



- **Generator**
  - Generate random decimal number
  - The range of output : $[0, 2^{16} - 1]$
- **Predictor**
  - Used as a discriminator and training data is not required.
  - Consist of 4 Conv1D layers.

System Configuration – Training & inference

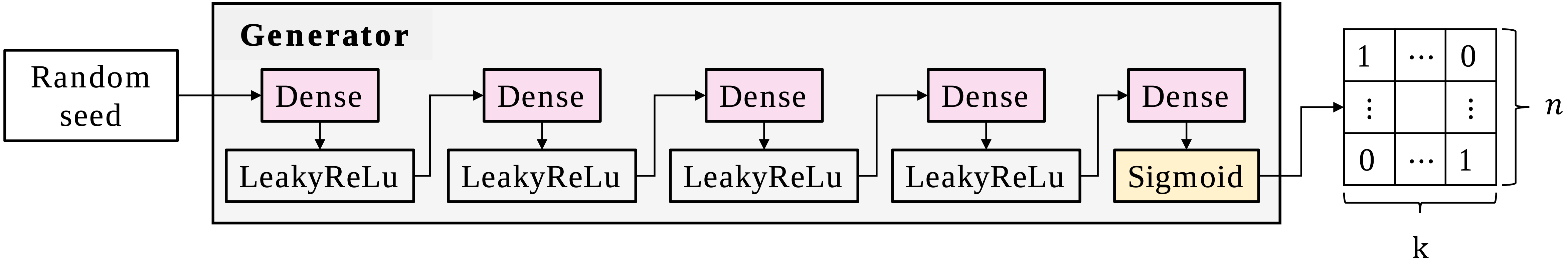


Training phase

Inference phase

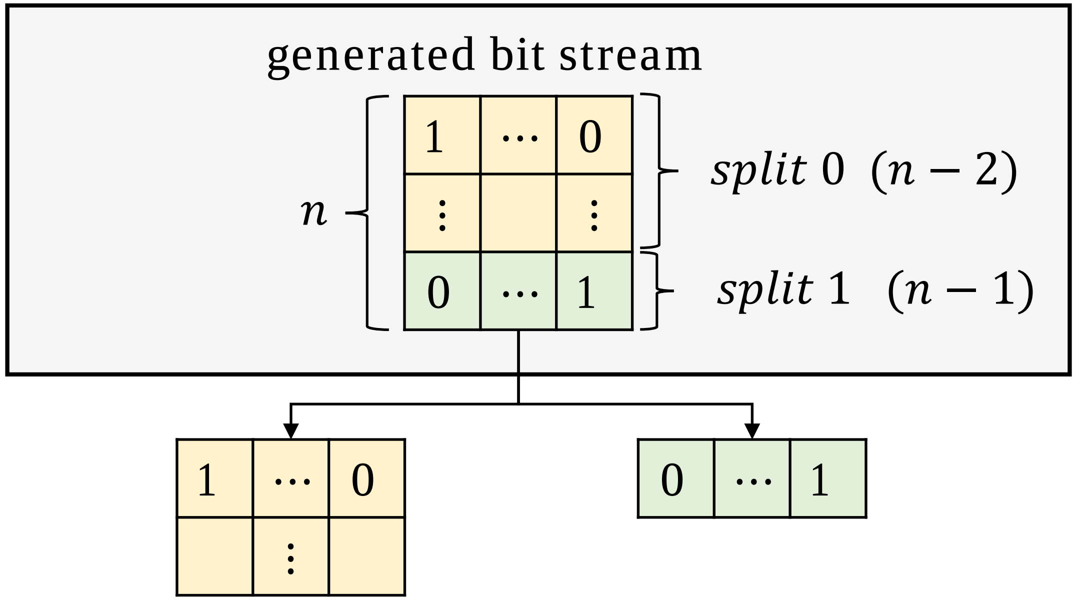
The generator model

- **n, k are adjustable hyperparameter**
  - Determine the number of bits to train.
- **sigmoid activation function**
  - Set the number of the desired range through bit-wise training (0 or 1) instead of training with a specific range of numbers.



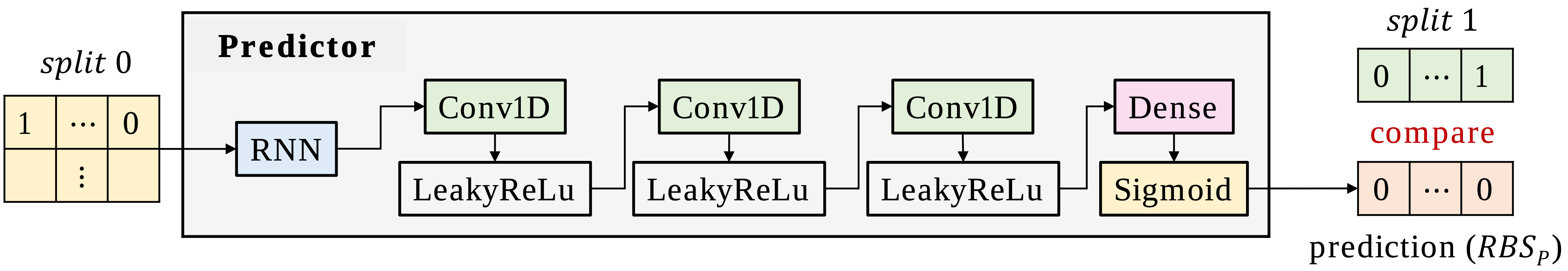
The predictor model

- **Split generated bit stream into 2 parts.**
  - $split0$: for training
  - $split1$: for comparision with predicted bit stream



The predictor model

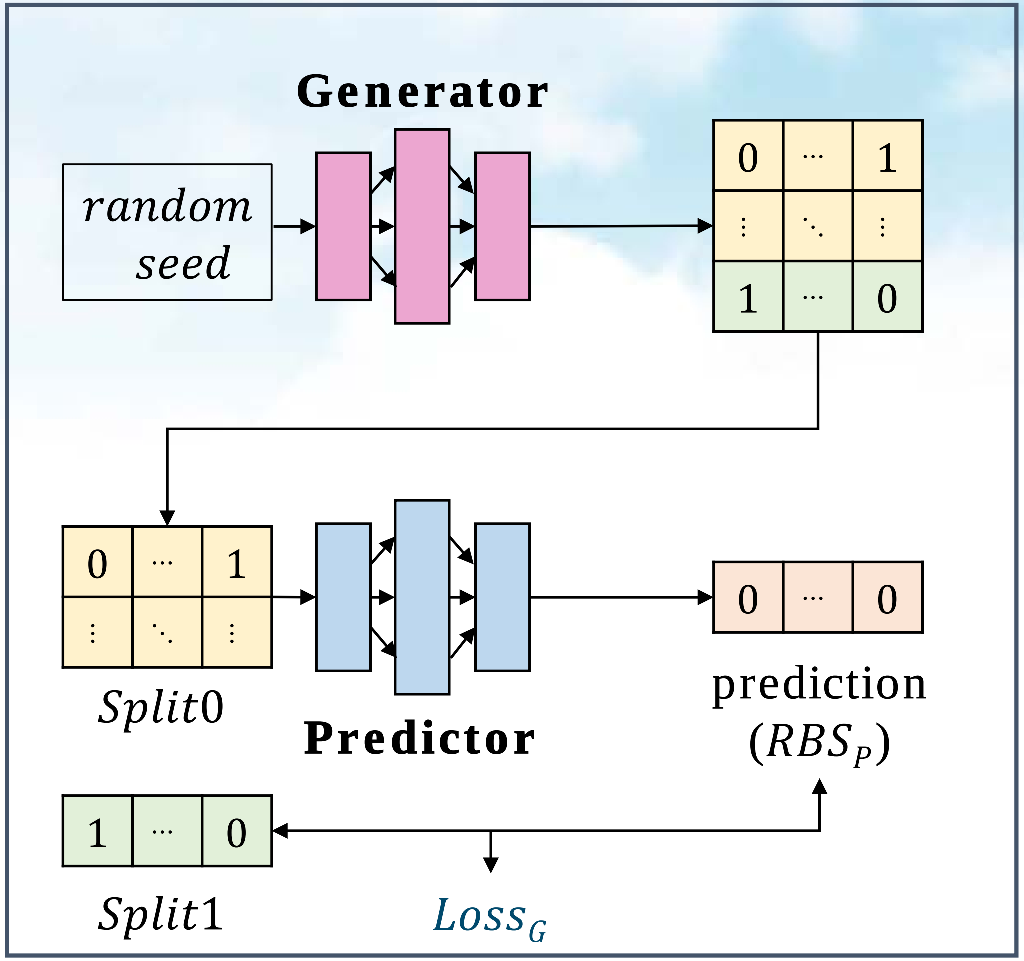
- **Using RNN**
  - Time series analysis using only CNN is difficult to have a mutual effect as the distance between data increases.
  - RNN is used to predict data following a random walk and have long-term dependency.
- $\boldsymbol{Loss_P = mean(|split1 - RBS_P|)}$



GAN based PRNG

- **Training the generator**
  - Trough combined model.
  - Loss is calculated by $split1$ and RBS_P.
    $Loss_G = mean(|1 - split1 - RBS_P|) \cdot 0.5$
- **Convert to decimal number.**
  - $c \leftarrow \sum_{i=0}^{m+t-1} 2^i \cdot RBS_i$
    $num \leftarrow c \bmod r$
  - The range of number is determined by setting r and m.

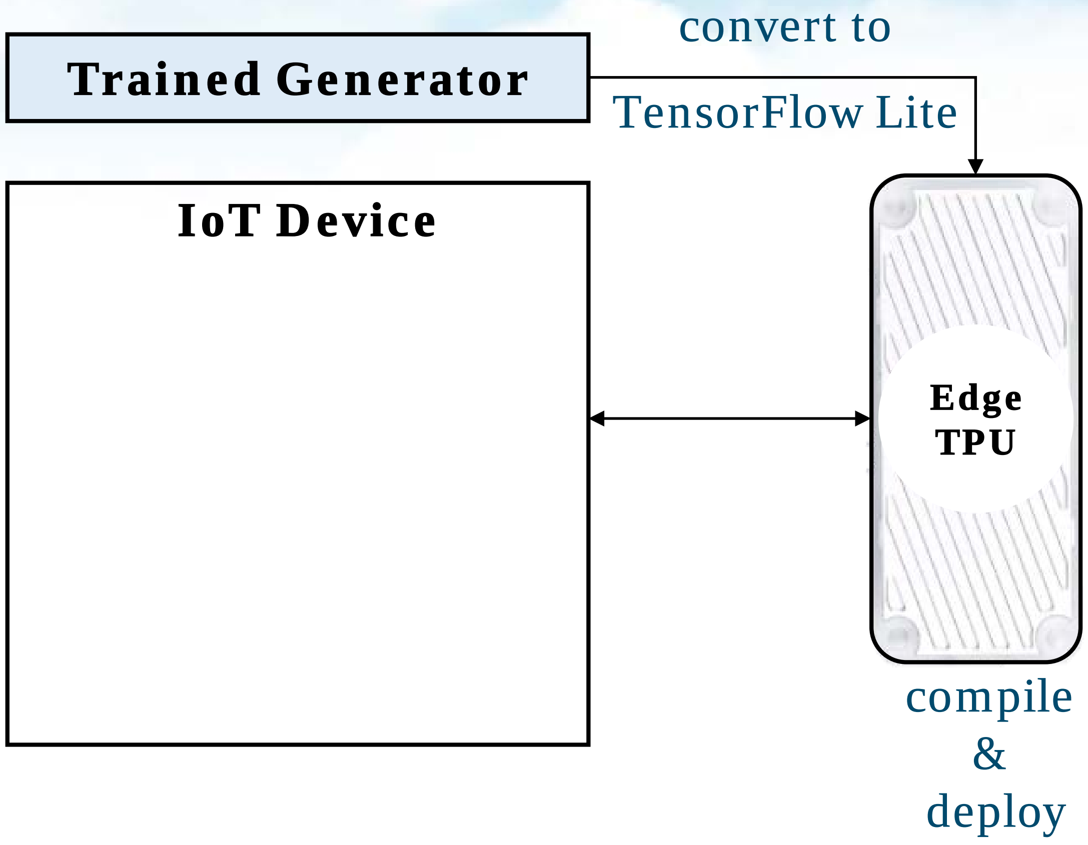
Combined model (Generator + Predictor)



$* \ Secure\ parameter\ (t), Range\ of\ random\ number\ (r), The\ number\ of\ bits\ needed\ to\ represent\ random\ number\ (m)$

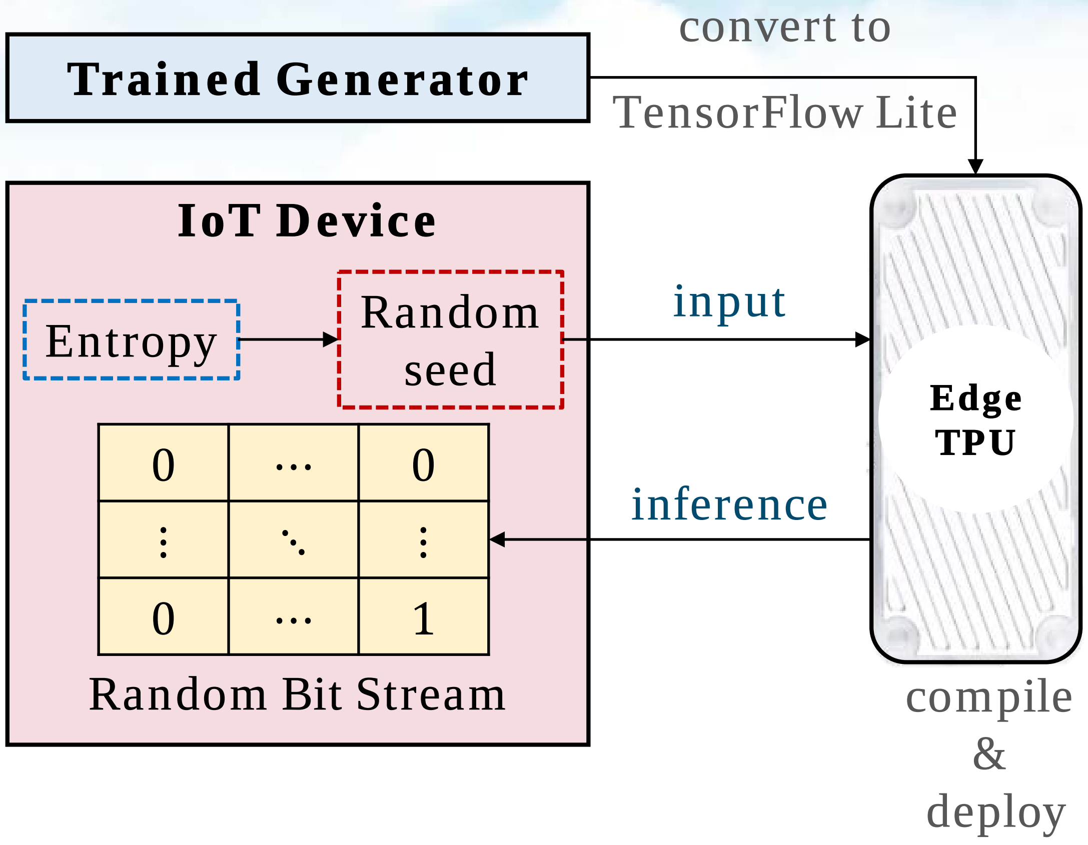
GAN based PRNG for Embedded Processors

- **Deploy only generator model**
  - The predictor is not required to generate the random bit stream.
  - Simple architecture for resource-constrained environment.



GAN based PRNG for Embedded Processors

- **Entropy for random seed**
  - The trained generator is a PRNG with a fixed internal state.
    → random seed with sufficiently high entropy is required.
  - Collected from IoT device**.** (e.g. sensor data)



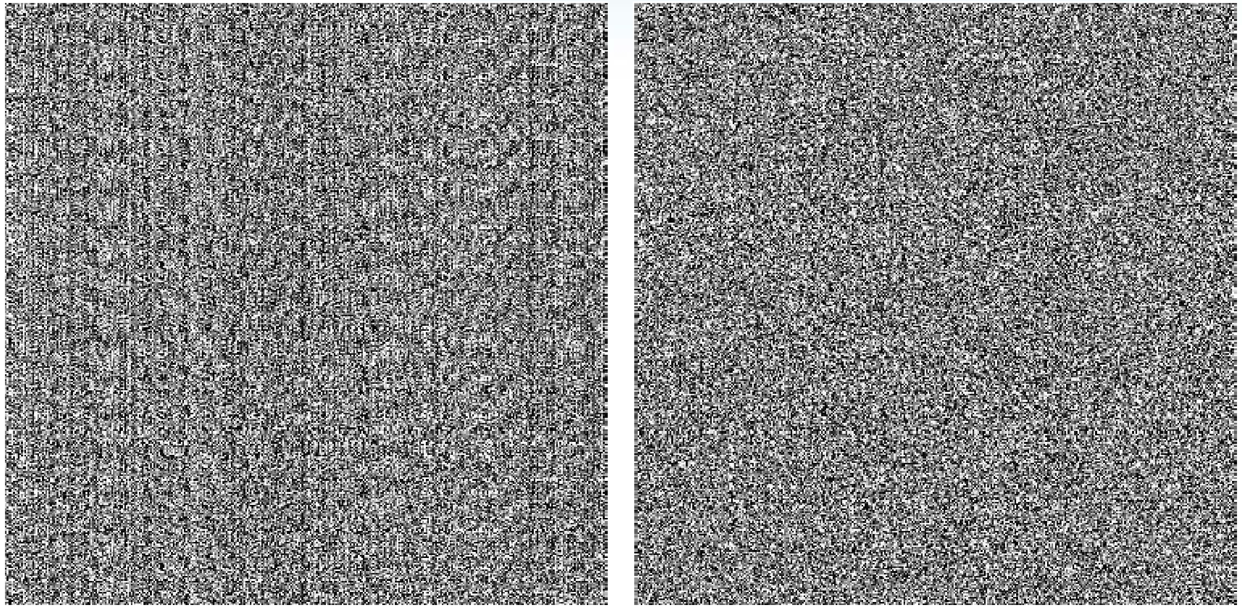
Comparison with the previous work

Table 1: Comparison of parameters with the previous works.

	Bernardi et al. [5]	This work
Data type	Decimal	Bit
Activation	Custom (range[0,2^{16}-1])	Sigmoid (0 or 1)
Loss	Mean Square Error	Mean Absolute Error
Seed : Output (bits)	64 : 262,144	64 : 1,099,200
Output Length	104,857,600-bits	109,920,000-bits
Optimizer	Adam (lr=0.02)	Adam (lr=0.0002)
Epoch	200,000	30

Visualization

- After training, the internal state changes.
- The generated bit stream is distributed without a pattern.



Visualization of random bit stream generated by the generator.
Before training (left) and after training (right).

NIST SP 800-22 : Randomness test for PRNG

- Improving the randomness of PRNG.
  - In the previous work, tests such as frequency and cumulative sums failed because they only used convolution layer.

generator is (data/c-1.pi)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
10	0	0	0	0	0	0	0	0	0	0.000000 *	0/10	* Frequency
2	2	3	0	0	1	1	1	0	0	0.350485	9/10	BlockFrequency
10	0	0	0	0	0	0	0	0	0	0.000000 *	0/10	* CumulativeSums
10	0	0	0	0	0	0	0	0	0	0.000000 *	0/10	* CumulativeSums
9	0	0	0	0	1	0	0	0	0	0.000000 *	1/10	* Runs
2	1	0	2	1	0	1	0	3	0	0.350485	10/10	LongestRun
0	0	1	5	2	2	0	0	0	0	0.004301	10/10	Rank
4	4	1	0	0	0	0	0	1	0	0.004301	8/10	FFT
3	0	1	1	2	0	0	2	0	1	0.350485	8/10	NonOverlappingTemplate
3	0	1	1	2	1	1	0	1	0	0.534146	10/10	OverlappingTemplate
0	0	1	1	2	2	1	3	0	0	0.350485	10/10	Universal
3	4	0	0	2	1	0	0	0	0	0.017912	9/10	ApproximateEntropy
0	0	0	0	0	0	0	0	0	0	----	7/7	RandomExcursions
0	0	0	0	0	0	0	0	0	0	----	7/7	RandomExcursionsVariant
0	1	3	1	1	1	0	0	1	2	0.534146	10/10	Serial
1	3	0	0	1	1	1	0	2	1	0.534146	10/10	Serial
1	1	1	1	2	0	0	1	1	2	0.911413	10/10	LinearComplexity

generator is (data/1.pi)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0	2	0	0	3	0	2	0	1	2	0.213309	10/10	Frequency
0	1	0	3	0	1	1	1	2	1	0.534146	10/10	BlockFrequency
0	1	1	2	1	0	1	2	2	0	0.739918	10/10	CumulativeSums
1	1	1	1	0	1	2	2	1	0	0.911413	10/10	CumulativeSums
1	0	0	5	0	0	1	1	0	2	0.008879	10/10	Runs
1	1	0	1	2	1	0	1	2	1	0.911413	10/10	LongestRun
0	1	1	1	1	2	1	0	0	3	0.534146	10/10	Rank
2	3	1	2	1	1	0	0	0	0	0.350485	10/10	FFT
1	1	1	0	1	3	0	1	1	1	0.739918	10/10	NonOverlappingTemplate
0	0	1	1	1	1	4	1	1	0	0.213309	10/10	OverlappingTemplate
1	0	0	0	1	2	1	0	3	2	0.350485	9/10	Universal
0	1	1	1	0	3	0	0	4	0	0.035174	10/10	ApproximateEntropy
0	0	0	2	0	1	2	0	0	2	----	7/7	RandomExcursions
0	0	0	1	1	0	0	2	2	1	----	7/7	RandomExcursionsVariant
2	0	1	3	0	0	1	0	1	2	0.350485	10/10	Serial
1	1	1	0	1	2	0	0	1	3	0.534146	10/10	Serial
0	1	0	3	2	0	2	2	0	0	0.213309	10/10	LinearComplexity

final analysis report of NIST test suite ; (left) previous work, (right) this work.

NIST SP 800-22 : Randomness test for PRNG

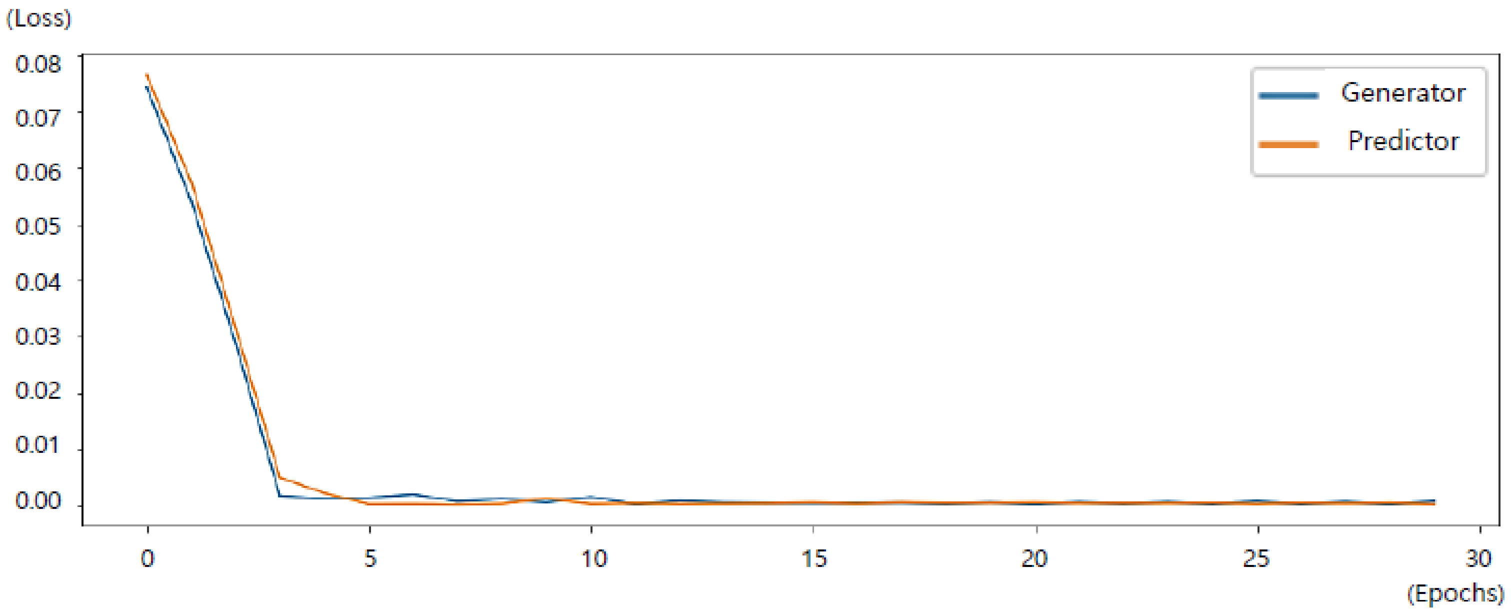
- The failed test instance (F_I/%) is reduced by about 1.91%.
- No failed p-value (F_P) in this work.
- The failed individual test (F%) is reduced by about 2.5%.

Table 2: Comparison of GAN based PRNG, where T, T_I, F_I, F_I/ %, F_P, F_T, F% are the number of individual tests, test instances, failed instances, their percentage, individual tests with p-value below the threshold, individual tests that failed, their percentage, respectively. The inference time is the time to generate a random number through trained generator.

	T	T_I	F_I	F_I/%	F_P	F_T	F%	inference time
Before training	188	1789	1769	98.8	160.8	186	98.9	177.32 ms
Bernardi et al. [5]	188	1830	56	3.0	2.7	4.5	2.5	187.09 ms
Proposed method	188	1794	19.6	1.09	0.00	0.1	0.00	13.27 ms

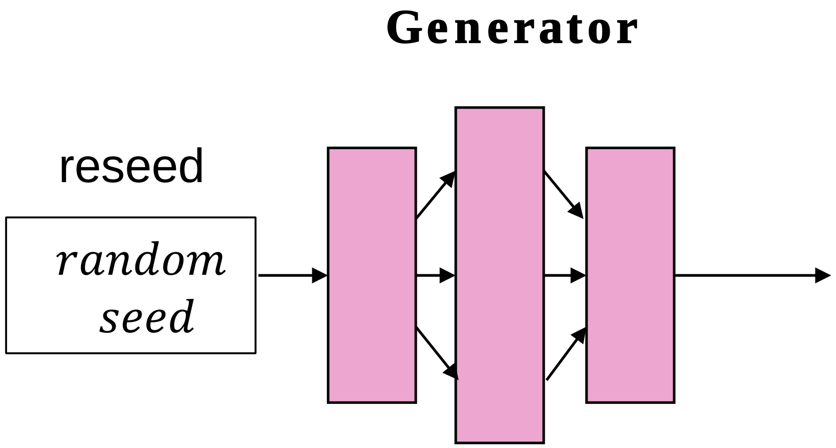
Unpredictability for CSPRNG

- Next bit test
  - The $m+1th$ bit cannot be predicted with the m-bit.
  - The training process means this test, so if the loss is minimized, the next bit will be unpredictable.



Unpredictability for CSPRNG

- State compromise attack resistance
  - When the internal state of PRNG is known at some time, the output can be predicted after or before.
  - Reseed for each batch to ensure resistance.



Comparison With Existing PRNGs

- Execution environment
  - The PRNGs on desktop : Intel Core i5-8259 CPU@2.30GHz x 8, 16GB.
  - MPCG64 : STM32F4.
  - This work : Edge TPU.

Table 3: Comparison with existing PRNGs.

	Throughput	Method	Machine
Xorshift128+	8.3 GB/s	XOR, Shift	Desktop
Xoroshiro128+	8.5 GB/s	XOR, Shift	Desktop
PCG64	4.3 GB/s	LCG	Desktop
MT19937-64	2.9 GB/s	Twisted GFSR	Desktop
MPCG [21]	0.16 GB/s	PCG	Embedded processors
This work	1.0 GB/s	GAN (Deep Learning)	Embedded processors

Conclusion and Future work

- **Conclusion**
  - GAN based PRNG (DRBG) for embedded processors.
  - High randomness validation through the NIST test suite.

- **Future work**
  - Optimizing to maintain high randomness while being more efficient for resource-constrained environments.
    - Applying other GAN models for high randomness and efficiency.
    - Designing a lightweight model through pruning.
    - Efficient entropy collection.

Thank you for your attention!