Noname manuscript No.
(will be inserted by the editor)

Efficient Lossy Compression for IoT Using SZ and Reconstruction with 1D U-Net

Joseph Azar · Gaby Bou Tayeh · Abdallah Makhoul · Raphaël Couturier

the date of receipt and acceptance should be inserted later

Abstract Several recent research has centered on maximizing Internet of Things (IoT) devices' lifetime by deploying data reduction techniques on IoT nodes to reduce data transmission. Data compression methods can be seen as a direct way of achieving energy efficiency. The trade-off between compression ratio and data distortion is usually considered when using a lossy compressor. This paper proposes a light SZ compressor with a maximal compression ratio without considering this trade-off. The proposed approach was tested on ESP Wroom 32 and WiFi LoRa 32 microcontrollers.

Given the importance of data quality arriving at the gateway for analysis, the proposed lossy compressor with a high compression ratio can discard important data features and patterns. This paper solves this problem by proposing a method for data enhancement based on the U-Net architecture. Therefore, the contribution of this paper is twofold: (1) Efficient data reduction approach for energy optimization at the level of IoT nodes. (2) 1D U-Net-based data recovery approach at the level of the edge.

Keywords IoT · Energy efficiency · LoRaWAN · Lossy compression · Data reduction · Deep learning · SZ · Data denoising

1 Introduction

Numerous and diverse sensors generate a massive amount of data in the IoT, which must be processed and stored with slight data loss. Additionally, the market for low-power, long-range sensory data transmission is growing rapidly. The majority of data collected from IoT devices is time series data. A time series is a collection of data that represents the values of a phenomenon through time. Time series data are used in a wide variety of applications, including healthcare, social sciences, environmental science, and computer networks.

Many of the IoT devices deployed are smart objects that store, process and transmit data. Such devices are still working on batteries and resource constrained. One of the direct ways to transmit the collected data effectively and keep the IoT devices working for as long as possible is to use a data reduction technique which does not require heavy processing and enables a high reduction in data volume. In other words, the energy needed for the computation and transmission of the reduced data should be less than the energy needed for the original data transmission. Such surveys [15][19] explored different approaches to data compression for IoT applications. Although lossy compression algorithms deliver a high reduction ratio, the high decrease in data size comes at a cost, namely data distortion. When it comes to medical applications, where data quality is critical, the impact of data compression techniques on the data obtained for analysis is controversial. The goal is to reduce energy use and data volume while maintaining data quality. Deep learning approaches have lately demonstrated excellent results in removing compression artifacts from images. Similarly, this approach was employed to time series data.

1.1 Related work

With advancements in deep learning and Neural Networks (NNs), it became possible to recover or denoise a highly distorted signal. In [1], the author proposed a

FEMTO-ST Institute, UMR 6174 CNRS, Univ. Bourgogne Franche-Comté, France
E-mail: joseph.azar@univ-fcomte.fr

hybrid approach of a Recurrent Neural Network (RNN) and a Denoising Autoencoder (DAE) for electrocardiograms (ECGs) signal denoising. The proposed model has been trained with synthetic ECG data, and it was found that the network trained with synthetic data showed better performance than the network trained with original data. In [24], the authors proposed a DAE and wavelet transform approach for ECG signal denoising. The wavelet transform is used to remove basic noise, while the DAE handles complex remaining noise that is difficult to remove using traditional methods. For denoising non-Gaussian noise in gravitational waves, the authors in [17] proposed an unsupervised model that uses bi-directional Long Short Term Memory (LSTM) and a DAE. While various publications investigate recurrent layers for time series denoising, there has been a recent shift toward other methods for this problem. The Generative Adversarial Network (GAN) models have attracted the interest of the scientific community since its introduction in 2014. Recent advances in medical time series denoising have included the use of different variants of GANs, Convolutional Neural Network (CNN)-based GANs, and DAE [26]. The authors in [2, 10] proposed generative adversarial approaches to remove the noise from ECG time series and produce realistically looking ECG signals. The results obtained were encouraging and demonstrate the potential medical use of GANs.

1.2 Motivation and contribution

Motivated by recent advances in deep neural networks and their signal denoising capabilities, in addition to the need for energy efficiency in IoT applications, this paper proposes an efficient lossy data compressor for resource-constrained IoT devices. In addition, it proposes a deep learning-based autoencoder model for data enhancement at the edge/sink node. The work conducted in this paper is a continuation of the one proposed in [3]. The previous work considers the trade-off between compression ratio and data distortion and provides a compression strategy that maximizes data compression while retaining a high level of reconstruction quality. On the other hand, this work disregards this trade-off and instead emphasizes on increasing the compression ratio in order to conserve more energy. This will very certainly result in highly distorted reconstructed data. To address this issue and restore the data's quality, this paper proposes a deep learning approach for data enhancement using a temporal convolutional network framework.

The contributions of this paper in relation to the previous work are: (1) Enabling a higher compression ratio in IoT nodes, hence significantly lowering the amount of data transferred. (2) Developing and evaluating the proposed approach on microcontrollers with fewer resources than a wearable device. (3) Using a 1-D convolutional autoencoder based on the U-Net architecture, enabling improved data recovery at the edge.

The experimental results obtained validate and emphasize the added value of this paper's contribution by demonstrating the ability of the proposed approach to strongly compress time series and recover the data at the edge without affecting the quality.

1.3 Organization

The rest of this paper is as follows. Section 2 gives a background information on IoT data compression and Autoencoders. The proposed data compression approach is discussed in section 3. Section 4 explains the advantages of the proposed approach in the context of LoRa-based applications. Section 5 details the proposed deep learning model for data enhancement. Experimental setup and results are described in section 6 and discussed in section 7. Section 8 discusses the limitations of the work presented in this paper and suggests some future research directions. Section 9 concludes this paper.

2 Background

This section provides background information on IoT data compression and denoising autoencoders.

2.1 IoT data compression

Data compression methods can be classified into two categories: lossless compression (no data loss involved) and lossy compression (leads to some data loss). In lossless compression, the maximum amount of data that may be compressed without loss is limited. Typically, the primary attribute to evaluate is the Compression Ratio (CR) while selecting an algorithm, as expressed in Equation 1:

$$CR = \frac{size\_orig}{size\_comp}, \tag{1}$$

where $size\_orig$ is the size of the uncompressed data, and $size\_comp$ is the size of the compressed data.

Lossless compression algorithms function by removing redundant data. The most often used lossless compression algorithms include Arithmetic Coding, Huffman coding, Run-length Encoding (RLE), and dictionary-based general-purpose algorithms such as Lempel-Ziv compression. Sprintz was recently proposed in [5] for IoT as an effective lossless compression algorithm for multivariate integer time series.

A lossy compression technique may allow for the removal of some redundant data during transmission. Numerous IoT applications do not require perfect signal reconstruction and can evaluate lossy signals such as in [3]. Lossy compression has a higher compression ratio than lossless compression, enabling IoT systems to transfer fewer data. Curve fitting, fractal resampling, box car, and wavelet, Fourier, and Chebyshev compression are all examples of lossy compressors.

2.2 Autoencoders

An autoencoder (AE) is a neural network (NN) that extracts features from a dataset using data-driven learning. An Autoencoder has the same number of input and output nodes. It is trained to rebuild the input vector rather than apply a label to it. The addition of noise to the encoding stage is an effective method for increasing the model's robustness. Denoising Autoencoder (DAE) addresses the problem of the hidden layer having more nodes than the input layer, causing the AE to duplicate those values without merely identifying relevant features. A randomly changed version of the input is used to train the AE. The output of the AE is matched with the original values, not the modified values, once the data has been processed through it. Due to the random distortion of the input, the DAE is considered as a stochastic form of an AE, with the undistorted version of the original input serving as the target for the decoding process.

Apart from being employed as a regularization method, the goal of DAE is to reduce noise from the input rather than learning the exact representation of a sample in order to reproduce it from low-dimensional characteristics. DAE is now being used to restore photos that have been damaged. This DAE feature has also been effectively applied to 1D signals, such as time series denoising. The model proposed in this paper is based on DAE's theory.

3 The proposed data compression with SZ

This section introduces the SZ compression method and explains its implementation on low-power system-on-a-chip microcontrollers.

3.1 Background

The authors in [9] proposed the *Squeeze* (SZ) lossy compressor to efficiently compress run-time High Performance Computing (HPC) snapshots. SZ is a versatile compressor since its compression errors are controlled by an absolute or relative error bound that the user can manage. Additionally, SZ is applicable to both one-dimensional floating-point arrays and multi-dimensional arrays.

The compression process can be divided into three steps: (1) transforming multi-dimensional array to 1D array, (2) compressing the linearized array using best-fit curve-fitting models, (3) compressing unpredictable data by examining their binary representation. The second step of SZ attempts to forecast each data point by using the previous data point's value, fitting a line to the previous two consecutive data points, or fitting a curve to the previous three consecutive values. The forecasted data point is replaced with a two-bit code that indicates the type of prediction model. If the predicted data point does not fall inside the given error-bound, it is labeled as unpredictable and encoded using binary representation analysis. Finally, lossless methods like as Huffman encoding and LZ77 are used by SZ.

This technique is an excellent candidate for IoT applications due to its adherence to the defined error bound and ability to compress multidimensional arrays of floating values. This approach, however, performs well on a PC/server in its current state, but not on a device with severe resource constraints. In [3], SZ was deployed on a wearable device by slightly altering the algorithm using the Android NDK toolkit. Because microcontrollers may have higher resource constraints than wearable devices in terms of memory, space, and processing, additional modifications to SZ should be done to ensure that it works well on any low-power IoT device. The next section discusses the lightweight version of SZ.

3.2 Light SZ for IoT

As previously stated, SZ is a lossy compression technique optimized for use in high-performance computing environments. Numerous modifications are necessary to adapt the code for use with ESP-32 like devices. Numerous SZ versions have been proposed for a variety of use cases, including visualization, accelerated I/O, memory and storage footprint reduction, and more advanced use cases such as compression of NN models and training sets[1]. The SZ version considered in this paper for adaptation is v2.1.5.

To begin, the code is lengthy (about 20,000 lines) and was written for 64-bit processors. Thus, the first adaption involved altering several pointers and types to expressly require the compiler to utilize 32-bit pointers. Then, because the code was written for various data types (double, float, int), only the float case was considered. Keep in mind that the storage capacity of IoT devices is quite limited, which is why all unneeded components have been deleted. Numerous big buffers were utilized in the code, however they were minimized to allow the code to run on ESP-32 processors. Because SZ can compress the output of lossy compression in a lossless manner using Gzip or Zlib, only Zlib was selected due to its ease of usage on Arduino devices. Finally, because developing code with an Arduino is more difficult than developing code on a standard machine, it took us long to modify it. Indeed, the Arduino ecosystem lacks an effective debugger.

4 LoRa for IoT

In this section, we will first provide a brief introduction on LoRa and LoRaWan for IoT. Moreover, the advantages the SZ compression algorithm provides in the context of LoRa-based sensor network applications is explained.

4.1 LoRa and LoRaWan: Definition

LoRa (Long Range) is a wireless communication protocol that combines extremely low power consumption and a long-range. While long-range is heavily dependent on the environment and any obstacles in the transmission line of sight, LoRa has a range of tens of kilometers. These characteristics provide LoRa an advantage over more power-hungry and/or short-range communication protocols such as Zigbee, Bluetooth Low Energy, WiFi, 3G, and LTE.

LoRaWAN (Long Range Wide Area Network) is a Low Power Wide Area Network (LPWAN) protocol explicitly designed for the IoT. It is intended for long-range, low-power operation, with sensor devices powered by batteries or energy harvesting systems. It takes full advantage of LoRa's capabilities to provide services like reliable message delivery, end-to-end security, location, and multicast capabilities.

[1] https://szcompressor.org/

Data Rate	Configuration	bits/s	Maximum payload size
DR0	SF12/125kHz	250	59
DR1	SF11/125kHz	440	59
DR2	SF10/125kHz	980	59
DR3	SF9/125kHz	1760	123
DR4	SF8/125kHz	3125	230
DR5	SF7/125kHz	5470	230
DR6	SF7/250kHz	11000	230

Table 1: Spreading Factor, Bandwidth, and Data Rate

4.2 Network Topology

While most networks adopt the mesh network topology, LoRaWAN uses the star network architecture. This enables the sensor end-devices in the LoRaWAN network to periodically go to sleep after each transmission. In comparison, end-devices in a mesh topology are generally constantly on, repeating transmissions from other devices to extend range. This is a crucial part in the low power consumption and long battery life of LoRa end devices. As seen in Figure 1, The LoRa Network Architecture comprises of four major parts:

- End Devices: These are sensors or actuators at the network edge that perform various functions and have different requirements.
- Gateways: In single-hop wireless communication, gateways connect to the network server using standard IP connections and relay messages between the centralized server and end devices.
- Network Server: LoRa network server connects the application server and gateways. It sends commands from the application server to the gateways and receives data from the gateways.
- Application Server: The application server decides what to do with the end-user data. Operations such as data visualization could be done here.

4.3 Spreading Factor and Data Rate

LoRa uses variable bandwidth and spreading factors (SF7-SF12) to adjust the data rate to the transmission range (see Table 1). A longer range could be allowed by increasing the spreading factor at the cost of the data rate. The bandwidth and spreading factor can be selected based on the link parameters and the data level to be sent. A larger spreading factor enhances transmission performance for a given bandwidth while decreasing data rates, increasing transmission time.

4.4 Rules and Regulations

LoRa operates on a free ISM frequency bands. For instance, in Europe, LoRa uses the 863 to 870 frequency

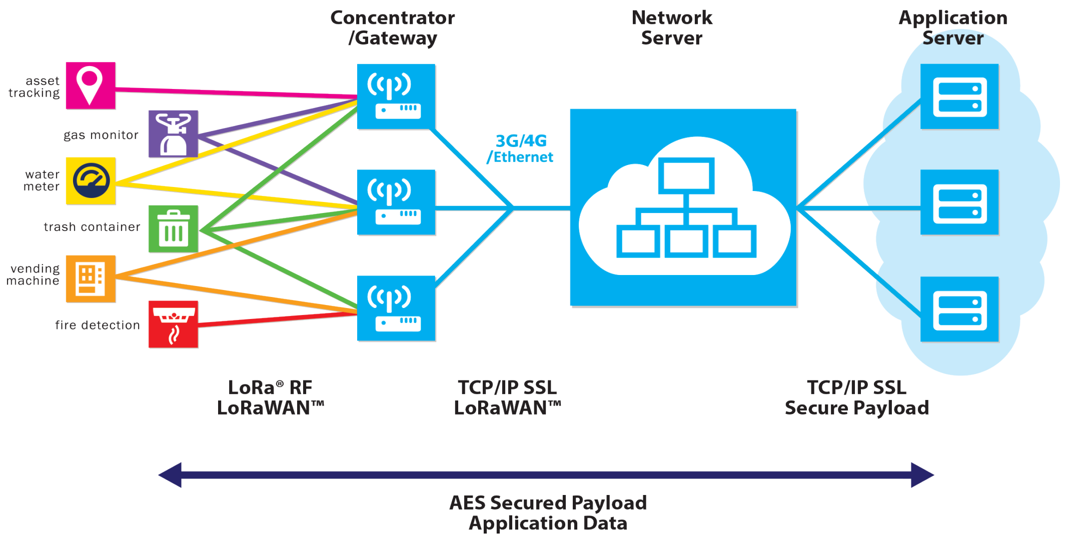


Fig. 1: LoRaWAN Network Architecture

band to transmit data packets. Anyone is allowed to use these frequencies and no licence is required. However, this means there is a lots of interference. Therefore, limitation rules are set by several international organizations, otherwise these bands will become unusable. For instance, in Europe users must comply to the following rules:

– The maximum transmission power must be limited to 14dBm.
– Only a duty cycle of 0.1% to 1.0% per day is allowed.
– Maximum allowed antenna gain is +2.15 dBi.

The duty cycle is the most problematic for IoT applications. This basically means that a LoRa IoT-device is allowed to transmit one packet of data every "X" seconds. "X" depends on several parameters, such as the Spreading Factor (SF), Bandwidth (BW), and payload size. For instance, assuming a LoRa device operating on the following parameters, SF = 12, BW = 125 Khz, and Payload Size = 59 bytes. For a 1% duty cycle, this device is allowed to transmit one packet every 04:23(mm:ss) approximately (X=236 seconds). For more information, readers are advised to check this LoRa air time calculator [13].

4.5 SZ compression for LoRa Sensor Networks: Motivation

As previously stated, LoRa end-devices have a maximum data rate of approximately 50kb/s. In comparison to the majority of other technologies, this rate is the slowest, which makes it unsuitable for some applications that demand high data rates. Additionally, LoRa-IoT devices must adhere to certain transmission duty cycles.

Data compression can be used to mitigate the impact of these limitations on LoRa-based IoT applications. Assume a sensor operates at 256Hz and acquires 256 floating-point samples per second, equating to 1024 bytes of data. Suppose this sensor uses SF7 and the payload is 230 bytes (transmission time = 36 seconds). So, six packets over 3.6 minutes are required to send the 1024 Bytes to the gateway. By heavily compressing the data and reducing the size from 1024 to 100 bytes, just one packet is required to transport the compressed data in less time.

By reducing the quantity of data that needs to be transmitted, we can accomplish the following benefits:

– With a lengthy period of time between transmissions mandated by the Duty Cycle, the time necessary to communicate the acquired data set to the gateway is significantly decreased.
– Because transmission consumes battery power, fewer transmissions leads to reduced battery usage, provided that compressing the data set consumes less power than transmitting it.

Section 7 will discuss the impact of compression on transmission time and energy consumption.

5 The proposed 1D U-Net for data enhancement

This section describes the proposed technique for removing compression artifacts from collected time series data. It is feasible to have paired data for these types of situations. In other words, the uncompressed form of the reconstructed data can be used as the output to train the network.

The approach presented in this section to enhance the compressed time series is inspired from [8]. The authors in [8] proposed a fully convolutional network that is based on the U-Net architecture originally presented in [16] and composed of an encoder decoder with skip connections. Given the time series nature of the data to be processed, we propose a 1D U-Net with skip connections between encoder and decoder layers on the same level.

5.1 Motivation

While the recurrent layer is an appealing conceptual choice for processing time series data, recent literature suggests that CNN can accomplish what LSTM has been employed for and excels at, namely analyzing sequences, but in a much faster and more computationally efficient manner[4, 14, 22, 27]. The authors in [22] showed that CNNs outperform LSTMs with and without attention on a prediction problem while requiring significantly less training time.

Additionally to the aforesaid, the U-Net architecture was adopted for time series denoising in this paper due to its recent use for time series segmentation. For example, in time series segmentation applied to sleep staging, the U-Time, a temporal fully convolutional network based on the U-Net[14], attained or exceeded state-of-the-art models from the literature as well as CNN-LSTM.

5.2 Architecture

As illustrated in Figure 2, the proposed architecture accepts a 1D signal input of size n and is divided into three parts: the contraction, the bottleneck, and the expansion. The contraction part consists of several contraction blocks. Each block receives an input and applies two one-dimensional convolution layers with a kernel size of three, followed by batch normalization and one-dimensional maximum pooling with a pool size of two. Following each block, the number of feature maps is doubled, allowing the architecture to efficiently learn complicated structures. The bottommost layer functions as a link between the contraction and expansion layers.

The expansion, like the contraction, is composed of multiple blocks. The decoding phase begins with the bottommost layer and an upsampling process that consists of an upsampling layer with a size of two, followed by a one-dimensional convolution layer with a kernel

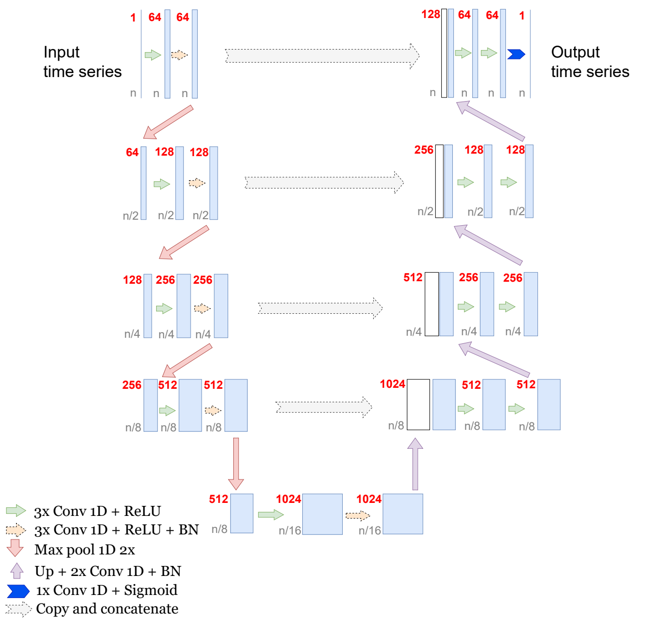


Fig. 2: Schematic diagram of the proposed deep encoder-decoder network architecture

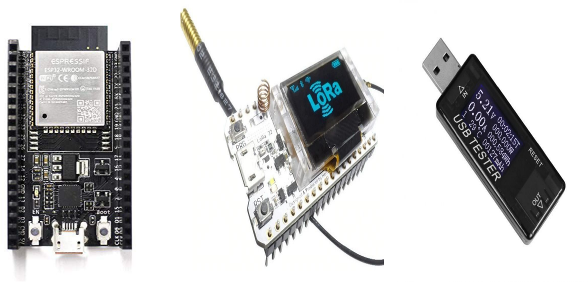
size of two and batch normalization. The input of each expansion block is connected by the feature maps of the corresponding contraction layer, and the concatenated results are passed to two one-dimensional convolution layers with a kernel size of three, followed by an upsampling layer. Following each block, the number of feature maps is reduced by half to maintain symmetry. This procedure ensures that the contraction part's learned features are employed in reconstructing the time series. Take note that the number of expansion blocks equals the number of contraction blocks.

Given an input range of [0,1], We map the final top decoder layer back to the clean output signal using 1D convolution with a kernel size of one, followed by sigmoid activation. Similarly, we use binary cross-entropy as the loss function and a stochastic gradient descent variant ('Adam') as the optimizer. Note that parameters such as the number of contractions, expansions, and filters are the same as in the original implementation of U-Net [16].

6 Experimental setup

Two microcontrollers were considered to evaluate the lightweight SZ compressor, namely the ESP Wroom 32 and WiFi LoRa 32. ESP Wroom 32 is a microcontroller unit designed for low-power sensor network applications and can use WiFi and Bluetooth for transmission. WiFi LoRa 32 is a popular IoT development board developed by Heltec Automation (TM) and supports communication via LoRa.

Two experiments were conducted. The objective of the first experiment is to determine the impact of the SZ compressor on energy consumption and the time needed for processing and transmitting the data to the gateway. We used a USB Tester to detect the voltage and current drawn and determine the amount of energy utilized in milliwatt hours by the microcontroller connected to the USB port. The used devices in the first experiment are shown in Figure 3.



(a) esp wroom 32 **(b)** Wifi LoRa 32 **(c)** USB tester

Fig. 3: The materials utilized to determine the impact of data compression on energy consumption and processing/transmission time

The objective of the second experiment is to enhance the transferred data to the gateway using the 1D U-Net described earlier. We consider in this paper the context of healthcare applications and, more particularly, physiological time series such as electrocardiogram (ECG) and photoplethysmogram (PPG). The challenge of working with such data is that the quality is a primary concern for doctors to analyze and extract meaningful information. The heavy compression of SZ may limit the analysis of physiological data. In the following section, we demonstrate how the proposed 1D U-Net enhances the quality of the compressed data.

We consider the ECG-ID Database published in Physionet [11] and initially presented in [21] for the ECG data. Concerning the PPG dataset, we used a Shimmer3 GSR+Unit [18] to collect the data from several PhD students from the FEMTO-ST/DISC/AND laboratory, Belfort, France. To train the NN model, NVIDIA Tesla Titan X GPU was used.

7 Results

This section summarizes the findings of the experiments performed to validate the approach given in this paper. The following examines four metrics: data compression, processing and transmission time, energy consumption, and data enhancement.

7.1 Data reduction

The approach taken in this paper is to employ a compression technique that maximizes compression ratio without regard for data distortion while also ensuring that the compressor is suitable for resource-constrained devices. To repair the compressor's damage, a deep learning technique is employed to enhance the reconstructed data. We evaluate the lightweight SZ with three error bounds on several batches of ECG and PPG time series: 10^{-1}, 10^{-2}, and 10^{-3}. Note that choosing an error bound e for SZ means that the reconstructed data points d are within the range $[d - e, d + e]$.

This experiment considers the cases in which the sensors transmit data on a periodic basis. We had set the period to $t = 30s$ and compressed a 10kb batch of data for 80 periods while recording the compressed size. This procedure is repeated using the three error bounds for the ECG and PPG data.

The results presented in Figure 4 show that in order to attain a maximal compression ratio, a higher error bound should be used. A 10^{-1} error bound gave a compression ratio that ranges from 80:1 to 100:1 for the ECG and PPG data. An error bound of 10^{-2} yields a compression ratio that varies from 40:1 to 85:1. This compression ratio falls to an average of 20:1 when SZ is used with an error bound of 10^{-3}. Table 2 summarizes the obtained results.

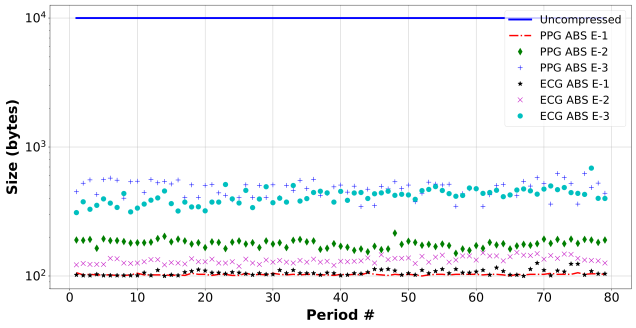


Fig. 4: Compression of 10Kb of ECG and PPG data across 80 periods using SZ with various absolute error bound settings

To better understand the gain of using a lossy technique such as SZ compared to a lossless method, one can consider the recent survey on time series compression published by Chiarot et al. [7]. Sprintz, one of the most effective lossless time series compression for the IoT[6], was evaluated with various lossless algorithms

	SZ E-1	SZ E-2	SZ E-3
PPG	97:1 ± 3	56:1 ± 10	20:1 ± 6
ECG	94:1 ± 13	75:1 ± 9	23:1 ± 9

Table 2: SZ compression ratios on ECG and PPG sets with three distinct error bounds

on diverse datasets in this survey. It is shown that the compression factor ranges from 3 to 10, which is less than using SZ with an absolute error bound of 10^{-3} and much less than using SZ with an error bound of 10^{-1}.

While this article focuses on extending the life of IoT devices, the remainder of our experiments employ SZ compression with an absolute error bound of 10^{-1}.

7.2 Transmission/Processing time

A significant benefit of data compression is that it reduces the time required to transfer data to the gateway. This section compares the time needed for uncompressed data transmission against the time needed for compressed data transmission. To accomplish this, the ESP Wroom 32 was used to generate the Mackay-Glass time series, and the time required to transfer various sizes of this time series via WiFi to a local PC was measured. The same process was used to measure the time needed for compressing+transmitting the data, and it was performed 20 times for each data size. The average time required in milliseconds for transmission and compression+transmission is depicted in Figure 5.

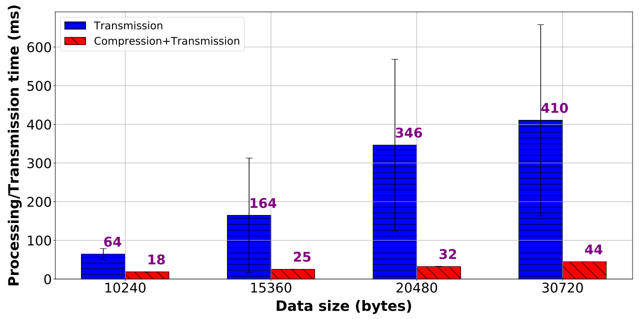


Fig. 5: Comparing the time required for data transmission with the time required for data compression and transmission in milliseconds

We can notice in Figure 5 that using lightweight SZ before transmission leads to a time gain of three to ten times compared to direct transmission without compression. It is important to note that SZ yields the optimal performance when it compresses a larger batch of data. This can be seen when the data size is above 20Kb in Figure 5. When using the proposed compressor in an IoT application with the periodic transmission, the advice that can be given is to use a longer period t if possible. This allows for collecting and transferring more data with lesser transmissions.

7.3 Energy consumption

This section discusses the impact of data compression on energy consumption. This experiment evaluated two different communication protocols: WiFi and LoRa. We implemented the proposed compression technique on an ESP Wroom 32 and a WiFi LoRa 32, and we estimated the energy consumption using a USB tester.

7.3.1 WiFi

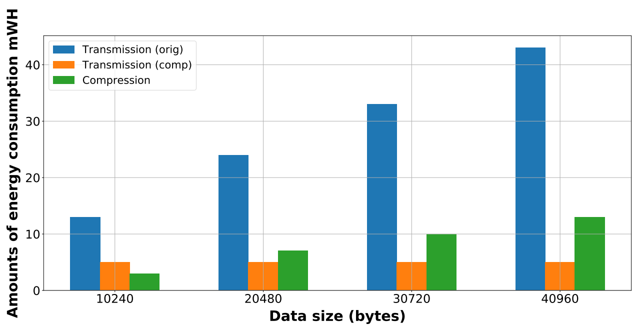


Fig. 6: Energy consumption in milliwatt hours for transmission of original data, compressed data, and the compression operation after 2500 cycles

The first experiment involves sending various sizes of the Mackay-Glass time series via WiFi from the ESP Wroom 32 to a local PC. We considered three scenarios:

– Data transmission without compression.
– Data compression with lightweight SZ without transmission.
– Data compression with lightweight SZ followed by transmission of the compressed data.

We repeated each scenario 2500 times consecutively for different data sizes, and we reported at the end the energy consumption in mWh. Note that in this experiment, the microcontroller does not enter a sleep state between consecutive transmissions.

Figure 6 shows that it needs 13, 24, 33, and 43 mWh to transmit 2500 times approximately 10Kb, 20Kb, 31Kb, and 41Kb of floating-point data. On the other hand, it takes approximately 8, 12, 15, and 18 mWH to compress

Data rate	Configuration	bits/s	Max payload size (Bytes)	Time on Air (ms)	Duty Cycle (mm:ss)
DR0	SF12/125kHz	250	59	2629.63	04:23
DR1	SF11/125kHz	440	59	1478.66	02:28
DR2	SF10/125kHz	980	59	657.41	01:06
DR3	SF9/125kHz	1760	123	656.38	01:06
DR4	SF8/125kHz	3125	230	635.39	01:04
DR5	SF7/125kHz	5470	230	363.78	00:36

Table 3: LoRa experimental parameters

and transfer (transmission+compression) the same batches of floating-point data 2500 times. It is worth noting that the energy required to transmit compressed data remains constant (five mWH) as the data size grows, although the processing energy increases slightly.

The energy savings ratio varies between around 1.5 and 2.5. For example, while transmitting 41Kb utilizing the proposed approach, energy consumption can be reduced by up to 2.5 times compared to sending the data without compression. It is possible to infer the same conclusion as in the preceding section, namely that raising the period duration *t* and collecting additional data, followed by compression and transmission, is the optimal technique for maximizing energy and storage gain.

7.3.2 LoRaWan

The second experiment involves transmitting various sizes of the Mackay-Glass time series to a LoRaWan gateway. Similarly to the first experiment, the following scenarios are considered: (1) data transmission without compression, (2) and data compression+transmission. Figure 7 shows the average values of the obtained results. Note that each scenario was repeated ten times.

In contrast to WiFi, and to adhere to the LoRaWan Duty Cycle, the device is compelled to communicate a limited quantity of data periodically after a predetermined length of time has passed. Meanwhile, the device is put to sleep because it is not able to communicate.

The maximum size of the transmitted packet and the sleep duration are determined by the Spreading Factor employed (SF). Table 3 provides detailed information.

The amount of energy consumed varies significantly based on the SF. The greater the latter, the lower the data rate, forcing the radio module to work for more extended periods to send the packet, thus increasing transmission energy. Furthermore, when the spreading factor grows, the packet size shrinks. As a result, more packets are required to transport the same total amount of data, resulting in increased energy consumption. Finally, the sensor consumes some energy even while it is in sleep mode. The more significant the spreading factor, the longer is the time on the air (ToA), forcing the sensor to wait a long time for the subsequent transmis-

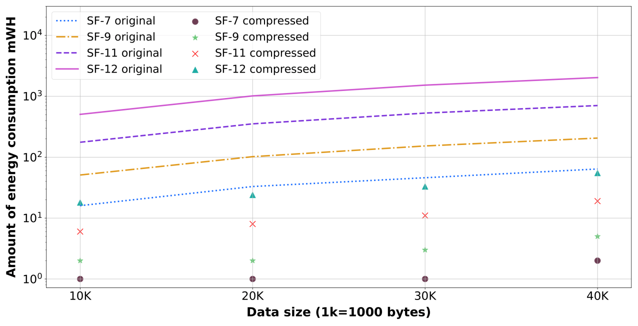


Fig. 7: The impact of the spreading factor and the data size on energy consumption

sion. The longer the sensor waits to communicate the data it has stored, the more power it consumes.

Figure 7 depicts all of this; the more significant the SF, the more energy is consumed. But, even most interestingly, we can observe how compression affects energy consumption. Indeed, compressing data before transmission reduces dramatically the amount of energy consumed.

7.4 Data enhancement

Metrics	Description
RMSE	Root mean square error
RMSSD	Root mean square of the Inter-beat (RR) Intervals (the time intervals between consecutive heart beats)
meanNN	Mean RR interval
BPM	Beats per minute
pNN20	The number of interval differences of successive RR intervals greater than 20 ms divided by the total number of RR intervals
pNN50	The number of interval differences of successive RR intervals greater than 50 ms divided by the total number of RR intervals
sdNN	Standard deviation RR interval

Table 4: time- and frequency-domain features extracted from PPG and ECG batches

This section demonstrates how the proposed deep learning model can significantly improve the data's quality and make analysis possible when applied to highly compressed data. For such cases, the conventional evaluation technique compares the root mean square error of the reconstructed and improved data. However, it may not be sufficient for all applications, particularly medical ones. Different time- and frequency-domain features of the compressed and improved signals are extracted and compared to those extracted from the original signals in the use case studied in this paper.

The PPG and ECG batches have an input size of 256 and 496, respectively. These numbers represent the

	RMSE	RMSSD	meanNN	BPM	pNN20	pNN50	sdNN
Original PPG		4.33	999.44	60	0	0	2.89
Reconstructed PPG	0.05	64.50	997.48	60	0.76	0.61	41.09
Enhanced PPG	0.02	8.86	1000	60	0	0	5.71
Original ECG		1.81	1000	60	0	0	1.09
Reconstructed ECG	0.05	116.51	974.06	68	77.27	63.63	85.28
Enhanced ECG	0.01	3.24	1000	60	0	0	1.73

Table 5: Comparison of the root mean squared error and the extracted feature values from the reconstructed and enhanced signals with those from the original signals

sampling frequency utilized to obtain the data. The entire time series were handled with a sliding window with the same size as the sampling frequency and had no overlap. The deep learning models enhance each window separately and then concatenate the windows to form the improved complete time series.

The features retrieved from ECG and PPG data using time and frequency domain analysis techniques such as FIR filters and fast fourier transform are summarized in Table 4.

Table 5 shows the metrics taken from the original, reconstructed, and enhanced signals. The RMSE is the first metric used to evaluate the proposed model. For the PPG and ECG, the RMSE was reduced from 0.05 to 0.02 and 0.05 to 0.01. It is clear that the extracted features from compressed data, such as RMSSD and sdNN, differ dramatically from the extracted features from uncompressed data. Because of this considerable discrepancy, analyzing such physiological data is unfeasible. Interestingly, the 1D-Unet enhanced the signals in such a way that the recovered features closely matched those of the original signal. Figures 8 and 9 show examples of enhanced ECG and PPG batches from the testing set. These figures depict the effect of lightweight SZ on the signal form (green signals) and how the proposed 1D U-Net recovered and enhanced the compressed signals (red signals). The proposed model allowed for heavier compression, resulting in a higher compression ratio even when signal quality was crucial.

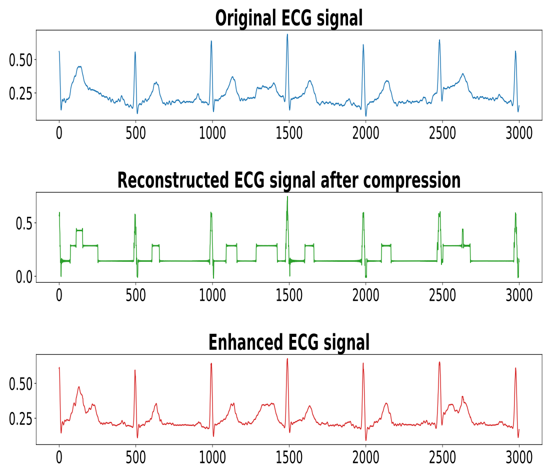


Fig. 8: Illustration of the enhancement of a compressed ECG batch using 1D-UNet

8 Limitations and future work

There are few floating-point data compressors available for resource-constrained microcontrollers. This article discusses how to adapt and apply SZ compression for IoT devices. To our knowledge, there are few lightweight compressors that can exceed SZ in terms of compression ratio. It is important to note that SZ has been tested against other lossy compressors and shown to outperform them[2][9]. As a result, we found that performing another compression ratio comparison will provide no further information, particularly given that not all lossy compressors are suitable for microcontrollers. However, one of this work's limitations is that it does not consider other metrics outside the compression ratio and compare with more recent works. This part will be revisited and considered in future works.

Concerning the implementation of the SZ on the microcontrollers, we intend to further clean the code before making it public and utilize a more recent version of SZ since it has been updated many times when we started working on this project. Also, we intend to repeat some experiments using low-power management technologies (low-power clocks, ultra-low power coprocessors, etc.).

Additionally, it is worth noting that the proposed approach would perform optimally when the window to be compressed is large. The appropriate use-case is to initially store the acquired data in a large buffer and subsequently compress it. If the number of samples to

[2] https://szcompressor.org/tabs/performance/

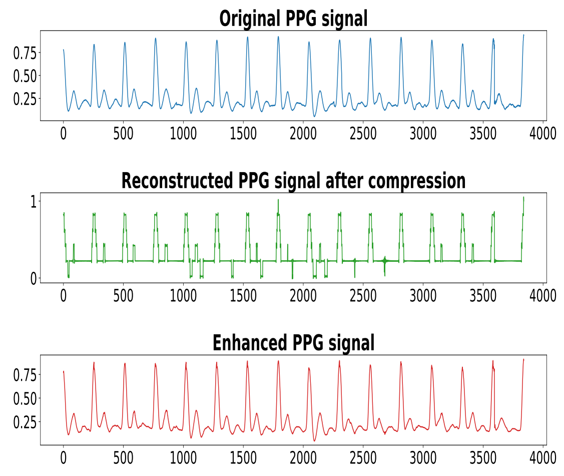


Fig. 9: Illustration of the enhancement of a compressed PPG batch using 1D-UNet

compress is small, the proposed approach may not produce the desired results.

Another limitation of this work concerning the data enhancement part is that it does not compare to the most recent time series denoising techniques, such as those based on GAN. Given that implementing and comparing several deep learning architectures for time series denoising is a lengthy and significant task in and of itself, we concluded that it is more appropriate to keep it for future work.

The input size of the 1D U-Net model in this paper is equal to the data sampling frequency. Initial experiments were conducted using slightly larger input windows, such as doubling the sampling frequency. The obtained results are similar to those presented in this paper. In the future, we aim to test the model on more large windows and consider different parameters than those of the original U-Net architecture. It is also worth noting that the size of the input time series linearly affects the memory needed for the model. Table 6 shows the relation between the batch size, input size, and the memory required for the model in Gb.

9 Conclusion

Given that IoT devices are often limited in terms of computation, storage, and energy consumption, data compression might be viewed as an efficient technique to extend the life of these devices. This article proposes and develops a lightweight Squeeze (SZ) compressor on real Internet of Things (IoT) hardware. The findings indicated that the proposed method can reduce the energy consumption of an ESP 32 by up to 2.5 times. Additionally, compressing big batches of data before transmission speeds up data transfer from the IoT to the gateway. It has been demonstrated that the proposed technique saves up to ten times the time of transmission compared to the direct transmission of uncompressed data. As a result, the algorithm proposed in this research can be used for many IoT applications that require a simple method that minimizes processing/transmission time and maximizes device lifetime. Given that this paper applies a heavy compressor and does not consider the trade-off between compression ratio and data distortion, the 1D U-Net architecture was proposed for enhancing the quality of compressed time series transferred to the gateway. The performance of the proposed architecture was tested on real electrocardiogram (ECG) and photoplethysmogram (PPG) datasets.

batch size	input size	needed memory (Gb)
16	256	0.088
	512	0.136
	1024	0.232
32	256	0.136
	512	0.232
	1024	0.423
64	256	0.232
	512	0.423
	1024	0.806

Table 6: Memory needed in Gb for the proposed architecture given the input size and the batch size

Acknowledgment

This work has been supported by the EIPHI Graduate School (contract “ANR-17-EURE-0002”). Computations have been performed on the supercomputer facilities of the “Mésocentre de calcul de Franche-Comté”

References

1. Antczak K (2018) Deep recurrent neural networks for ecg signal denoising. ArXiv abs/1807.11551
2. Antczak K (2020) A generative adversarial approach to ecg synthesis and denoising. arXiv preprint arXiv:200902700
3. Azar J, Makhoul A, Barhamgi M, Couturier R (2019) An energy efficient iot data compression approach for edge machine learning. Future Generation Computer Systems 96:168 – 175, DOI https://doi.org/10.1016/j.future.2019.02.005
4. Bai S, Kolter JZ, Koltun V (2018) An empirical evaluation of generic convolutional and recurrent

networks for sequence modeling. arXiv preprint arXiv:180301271

5. Blalock D, Madden S, Guttag J (2018) Sprintz: Time series compression for the internet of things. Proc ACM Interact Mob Wearable Ubiquitous Technol 2(3):93:1–93:23, DOI 10.1145/3264903
6. Blalock D, Madden S, Guttag J (2018) Sprintz: Time series compression for the internet of things. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2(3):1–23
7. Chiarot G, Silvestri C (2021) Time series compression: a survey. arXiv preprint arXiv:210108784
8. Couturier R, Perrot G, Salomon M (2018) Image denoising using a deep encoder-decoder network with skip connections. In: Cheng L, Leung ACS, Ozawa S (eds) Neural Information Processing, Springer International Publishing, Cham, pp 554–565
9. Di S, Cappello F (2016) Fast error-bounded lossy hpc data compression with sz. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), vol 00, pp 730–739, DOI 10.1109/IPDPS.2016.11
10. El Bouny L, Khalil M, Adib A (2021) Convolutional denoising auto-encoder based awgn removal from ecg signal. In: 2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA), pp 1–6, DOI 10.1109/INISTA52262.2021.9548524
11. *et al* ALG (2000) Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. Circulation
12. Liu F, Zhang G, Lu J (2020) Heterogeneous domain adaptation: An unsupervised approach. IEEE transactions on neural networks and learning systems 31(12):5588–5602
13. LoRa air time calculator (2021) https://www.loratools.nl/#/airtime, accessed: 2021-04-09
14. Perslev M, Jensen MH, Darkner S, Jennum PJ, Igel C (2019) U-time: A fully convolutional network for time series segmentation applied to sleep staging. arXiv preprint arXiv:191011162
15. Razzaque MA, Bleakley C, Dobson S (2013) Compression in wireless sensor networks: A survey and comparative evaluation. ACM Trans Sen Netw 10(1):5:1–5:44, DOI 10.1145/2528948
16. Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. CoRR abs/1505.04597, `1505.04597`
17. Shen H, George D, Huerta EA, Zhao Z (2017) Denoising gravitational waves using deep learning with recurrent denoising autoencoders. `1711.09919`
18. Shimmer sensing (2021) http://www.shimmersensing.com/products/shimmer3-wireless-gsr-sensor, accessed: 2021-04-09
19. Srisooksai T, Keamarungsi K, Lamsrichan P, Araki K (2012) Practical data compression in wireless sensor networks: A survey. Journal of Network and Computer Applications 35(1):37 – 59, DOI http://dx.doi.org/10.1016/j.jnca.2011.03.001, collaborative Computing and Applications
20. Sun G, Cong Y, Wang Q, Zhong B, Fu Y (2020) Representative task self-selection for flexible clustered lifelong learning. IEEE Transactions on Neural Networks and Learning Systems
21. TS L (June 2005) Biometric human identification based on electrocardiogram. Master's thesis, Faculty of Computing Technologies and Informatics, Electrotechnical University "LETI", Saint-Petersburg, Russian Federation
22. Weytjens H, De Weerdt J (2020) Process outcome prediction: Cnn vs. lstm (with attention). In: International Conference on Business Process Management, Springer, pp 321–333
23. Wu Z, Song A, Cao J, Luo J, Zhang L (2017) Efficiently translating complex sql query to mapreduce jobflow on cloud. IEEE Transactions on Cloud Computing 8(2):508–517
24. Xiong P, Wang H, Liu M, Zhou S, Hou Z, Liu X (2016) Ecg signal enhancement based on improved denoising auto-encoder. Engineering Applications of Artificial Intelligence 52:194 – 202, DOI https://doi.org/10.1016/j.engappai.2016.02.015
25. Yi H (2021) Secure social internet of things based on post-quantum blockchain. IEEE transactions on Network Science and Engineering
26. Zhang H, Zhao M, Wei C, Mantini D, Li Z, Liu Q (2021) Eegdenoisenet: A benchmark dataset for deep learning solutions of eeg denoising. Journal of Neural Engineering 18(5):056057
27. Zhang X, Zhao J, LeCun Y (2015) Character-level convolutional networks for text classification. Advances in neural information processing systems 28:649–657