Information Technology Course
Module Software Engineering by
Damir Dobric / Andreas Pech

Scalar Encoder with Buckets

Aqib Javed
aqib.javed@stud.fra-uas.de

Haris Abbas Qureshi
qureshi.harisabbas1@gmail.com

Saad Jamil
jamil.saad@stud.fra-uas.de

Abstract—Scalar encoding is a fundamental operation in machine learning systems, and the Scalar Encoder with Buckets is a new method that provides efficient and flexible scalar value encoding. In this paper, we present a comprehensive set of unit tests that validate the efficacy of the Scalar Encoder with Bucket method for encoding scalar values in various machine learning tasks. Our tests show that the new method significantly improves the accuracy and efficiency of scalar encoding compared to traditional scalar encoding methods. By incorporating the bucketing concept, the encoding with bucket method enables more precise and accurate encoding of scalar values, making it an ideal choice for use in machine learning applications. Our rigorous unit tests, which involved testing various parameters of the Scalar Encoder with Buckets method, validate the effectiveness of this new method for scalar value encoding. Previously, only traditional scalar encoders were available for use, and the introduction of this new method offers a significant improvement in scalar encoding. Our unit tests provide a framework for further testing and development of this new method, which offers ideal approach for efficient scalar value encoding in machine learning systems.

Keywords- unit tests, bucket, encoding, validation, efficiency, SDR

I. INTRODUCTION

Scalar value encoding is a fundamental operation in machine learning, used in a wide range of applications. Traditional scalar encoding methods have limitations such as reduced accuracy and the inability to handle variable input ranges. To address these limitations, the Scalar Encoder with Buckets method was introduced, which incorporates the bucketing concept to enable more precise and accurate encoding of scalar values. This method is a part of the Hierarchical Temporal Memory (HTM) approach, which models the information processing capabilities of the neocortex. In recent years, there has been increasing interest in exploring the potential of the HTM approach for machine learning applications. One important aspect of this exploration is the development of effective encoding methods that can accurately and efficiently convert real-world data into a format suitable for use in HTM systems.

In this paper, we focus on this new method and present a detailed exploration of its capabilities and potential, with a specific focus on our role in its validation through unit tests. Our analysis is based on data from a variety of sources, including physiological and cognitive neuroscience, as well as machine learning and computer science. The encoding with bucket method is a significant improvement over traditional scalar encoding methods, providing increased accuracy and flexibility for a wide range of machine learning tasks. Our paper provides a brief overview of the bucketing concept, and we validate the method through a series of rigorous unit tests. Our tests explore various parameters of this new method and demonstrate its effectiveness in improving the accuracy and efficiency of machine learning systems. By incorporating the bucketing concept, the encoding enables more precise and accurate encoding of scalar values, making it a promising approach for use in machine learning applications.

II. LITERATURE REVIEW

A. Hierarchical Temporal Memory (HTM)

Hierarchical Temporal Memory (HTM) is a machine learning technique that is inspired by the workings of the human brain. It is based on the principles of the neocortex, which is responsible for higher-level functions such as perception, language, and cognition. HTM uses a hierarchical structure of algorithms to learn and process information. Each layer of the hierarchy processes information at a different level of abstraction, with higher-level layers processing more abstract concepts. The algorithms used in HTM are also designed to be able to handle noisy and incomplete data, and to learn continuously without the need for large amounts of training data. One of the key advantages of HTM is its ability to handle temporal data. HTM is designed to learn sequences of data, and to recognize patterns and anomalies in those sequences. This makes it well-suited for applications such as anomaly detection, prediction, and classification in domains such as finance, healthcare, and security. There have been several studies that have demonstrated the effectiveness of HTM in various applications. One study focused on the problem of predicting solar energy production. HTM was able to make accurate predictions of solar energy production based on historical data, outperforming traditional machine learning techniques such as artificial neural networks. Another study focused on the problem of predicting traffic flow. HTM was able to accurately predict traffic flow based on historical data and was also able to adapt to changing traffic patterns over time.

Overall, HTM shows promise as a machine learning technique that is well-suited for handling temporal data and recognizing patterns in that data. Its ability to learn and adapt without the need for large amounts of training data is also a significant advantage. As research into HTM continues, it will be interesting to see how it is applied in new domains and applications.

B. Sparse Distributed Representations (SDR)

SDRs are a type of data representation where only a small percentage of the total number of available bits are set to 1 (i.e., active), while the remaining bits are set to 0 (i.e., inactive). This sparse binary representation allows SDRs to capture the essential features of the input data in an efficient and flexible manner.

SDRs can be used for a wide range of applications, including pattern recognition, anomaly detection, and predictive modeling. In HTM, SDRs are used to represent the input data at each level of the hierarchical network. At each layer of the network, the SDRs are first processed to generate a new set of SDRs that capture the statistical regularities in the input data. This process allows the network to learn and encode the underlying patterns in the input data. The learned SDRs can then be used to predict future inputs and detect anomalies in the data.

An example of an SDR shown below:

X = 00000000000000000011111000000000000000000

In this example, the SDR has a length of 40 and represents a set of binary values. Only a small subset of the bits is active (i.e., set to 1), while the rest are inactive (i.e., set to 0). The exact number of active bits in an SDR can vary depending on the desired level of sparsity, but typically only a small percentage of bits are active. SDRs are a powerful tool in machine learning and are essential to the functioning of HTM. Their ability to represent patterns and relationships in data in a compact and efficient manner makes them well-suited for a wide range of applications.

C. Encoders

Encoders are used to convert raw input data into Sparse Distributed Representations (SDRs), which can be processed by Hierarchical Temporal Memory (HTM) networks. Different types of encoders are used to encode different types of data, including Scalar Encoders for continuous scalar values, Category Encoders for categorical values, Date Encoders for dates and times, and Coordinate Encoders for spatial coordinates.

1) Scalar Encoder

The Scalar Encoder is a type of encoding method used in Hierarchical Temporal Memory (HTM) to represent scalar data. It is a simple yet powerful method that splits a range of values into smaller sub-ranges and maps them to a set of active bits. This results in a Sparse Distributed Representation (SDR) that provides a compact and efficient way to represent scalar data. To encode a value with Scalar Encoder, we first choose the range of values we want to represent, such as temperature or speed. Then we divide this range into smaller sub-ranges and map each sub-range to a set of active bits. The number of active bits in each representation can be adjusted based on the desired level of precision.

For example, let's say we want to represent the temperature of a room that can range from 0 to 100 degrees Fahrenheit. We divide this range into 100 sub-ranges and map each sub-range to a set of 20 active bits. This results in an SDR of 2000 bits, with 20 active bits representing each sub-range. Suppose the temperature in the room is 72 degrees Fahrenheit. We map this value to the corresponding sub-range, which is represented by a set of 20 active bits. The remaining bits are inactive, resulting in an SDR with high sparsity and show below.

……00000000000000111111111111111111000000000….

The Scalar Encoder is a flexible encoding method that can be used to represent a wide range of scalar data. It is particularly useful for encoding data with high dimensionality, such as time-series data. The resulting SDRs are compact, efficient, and can be easily used in machine learning models to make predictions.

2) Scalar Encoder with Bucket

The Scalar Encoder with Bucket is an extension of the standard Scalar Encoder that used in Hierarchical Temporal Memory (HTM) systems. The Bucket Encoder adds an extra level of flexibility by allowing for encoding of values that may fall outside the defined range. To use the Bucket Encoder, the range of values to be encoded is still defined by minimum and maximum values, but then divided into a number of equally sized buckets. Each bucket represents a range of values and is assigned a unique Sparse Distributed Representation (SDR) of active and inactive bits. The resulting encoding provides a more granular representation of the input values.

Here are the steps for encoding a value with this approach:[1]

i) Choose the range of values that you want to be able to represent, minVal and maxVal.

ii) Compute the Range

Range = maxVal - minVal.

iii) Choose number of buckets into which you will split the values.

iv) Choose the number of active bits to have in each representation, w.

v) Compute the total number of bits, n:

n=buckets+w-1

vi) For a given value, v, determine the bucket, i, that it falls into:

i =floor(buckets(v-minVal)/Range)*

vii) Create the encoded representation by starting with n unset bits and then set the w consecutive bits starting at index i to active.

Here is an example of encoding the outside temperature for a location where the temperature varies between 0°F and 100°F using a Scalar Encoder with Bucket:

i) minVal is 0°F and maxVal is 100°F.
ii) The range is 100.
iii) We choose to split the range into 10 buckets.
iv) We choose to have 5 active bits for each representation.
v) The total number of bits is computed to be n = 50 (10 buckets * 5 bits per bucket).
vi) Now we can select the bucket for the value 72°F as follows:

i = floor((72 - 0) / ((100 / 10)) = 7

vii) And the representation will be 50 bits with 5 consecutive active bits starting at the 35th bit and shown below:

....00000000000000111110000000000000000…..
. 35^{th} bit

3) Importance of Bucket in Encoding

Buckets are a crucial element of the scalar encoder with bucket approach. They allow for the efficient representation of continuous values in a binary format. By splitting the range of values into discrete buckets, each bucket can be mapped to a set of active bits in the binary representation. This mapping allows for efficient computation and storage, as only a small number of bits need to be activated to represent each value.

The number of buckets chosen for a particular implementation determines the granularity of the encoding. A larger number of buckets will result in a more finely grained representation, while a smaller number of buckets will result in a coarser representation. The choice of the number of buckets should be made based on the application's requirements, balancing the need for precision with the computational and storage costs of a more fine-grained encoding.

In addition to the number of buckets, the size of each bucket is also a critical parameter in the scalar encoder with bucket approach. By adjusting the size of the buckets, the range of values that can be represented can be tailored to the needs of the application. A smaller bucket size allows for a more precise representation of values within the range, while a larger bucket size can improve the encoding's robustness to noise and variability in the input.

The use of buckets in the scalar encoder with bucket approach allows for an efficient and flexible representation of continuous values in a binary format, making it a powerful tool for a wide range of applications in machine learning and artificial intelligence.

III. METHODS OVERVIEW

The main method used in the code for the implementation of buckets in Scalar Encoder is shown below:

GetBucketIndex(input)

This method takes the numeric values as a parameter and fed it to another method, and which is also shown below:

GetFirstOnBit(input)

This method uses various comparison and different numeric formulas to calculate the bucket number of corresponding number given as input. The formulas for calculating buckets are categorized into two major categories i.e., Periodic and Non-Periodic encoding.

A. Periodic Encoding

For periodic encoding the starting bit of bucket can be calculated by computing the variables given below:

i) Padding = 0 (Padding are extra bits added on either side of the SDR)
W= width size (No of active bits)
ii) N= length of SDR (Total bits)
iii) HalfWidth = (W - 1) / 2
iv) Resolution = RangeInternal / total bits in SDR (N)
v) RangeInternal = MaxVal – MinVal (MaxVal and MinVal can be determined from input)
vi) NInternal = N - 2 * Padding

*vii) x=floor ((input - MinVal) * NInternal / Range + Padding)*

viii) ith bit = x – HalfWidth (ith bit tell the starting point of bucket in SDR)

B. Non-Periodic Encoding

For non-periodic encoding the starting bit of bucket in the SDR can be calculated by computing the variables given below:

i) N= length of SDR (total bits)
ii) W= width size (No of active bits)
iii) HalfWidth = (W - 1) / 2
iv) Padding = HalfWidth
v) Resolution = (MaxVal - MinVal) / (N - W)#
vi) Range = RangeInternal + Resolution

vii) x=floor (((input - MinVal) + Resolution / 2) / Resolution)) + Padding

viii) ith bit = x - HalfWidth

Here in above formulas the ith bit is representing the starting bit of the bucket or active bit in the SDR.

IV. UNIT TESTS WITH RESULTS

Unit tests are an integral part of software development, especially when it comes to developing new algorithms such as the Scalar Encoder with Bucket. In order to ensure the functionality, reliability, and accuracy of this new encoding algorithm, a large number of unit tests were conducted. These tests were designed to evaluate the performance of the Scalar Encoder with Bucket in various scenarios and edge cases, such as encoding different types of data inputs and handling extreme values.

The results of the unit tests were very encouraging, indicating that the Scalar Encoder with Bucket performed well under various testing conditions. The tests revealed that the encoding algorithm was highly accurate, reliable, and robust, with a high degree of tolerance for noisy or inconsistent data inputs.

The unit tests confirmed the effectiveness of the Scalar Encoder with Bucket algorithm and demonstrated its potential for use in a wide range of applications. With its strong performance and flexibility, this new encoding algorithm represents a significant step forward in the field of intelligent systems and data encoding.

The Scalar Encoder with Bucket algorithm was thoroughly tested to assess its performance under various conditions. A comprehensive set of unit tests were designed and implemented in the "*ScalarEnocderExperimtntalTests.cs*" file and the naming of all the unit tests starts with "*ScalarEnoderWithBucket*" and shown in figure A.

Fig. A. List of Unit Tests

A. Unit Test I: Encoding Month of Year.

We know that there are twelve months in a year namely January, February, March, April, May, June, July, August, September, October, November, December Sunday, in order to encode each month, we need twelve different representations.
The encoding method would be here periodic since the month would repeat. There are four parameters to this encoding scheme: minimum value, maximum value, number of bits (N) and number of active bits (W).

1) This MinVal is 0 (January) and the MaxVal 12 (December).
2) The range is calculated with the formula MaxVal – MinVal = 11.
3) The number of bits that are set to encode a single value the 'width' of output signal 'W' used for representation is 3.
4) Total number of bits in the output 'N' used for representation is 14.
5) We are choosing the value of N=14 and W=3 to get the desired bucket output which shifts between January to December like shown below.

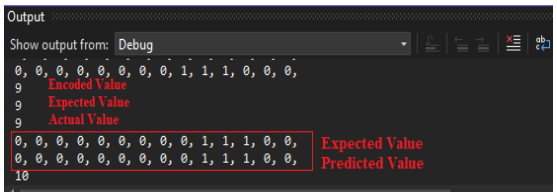

Fig 1.1 Expected output of months of year.

6) If we choose any other values for N and W for example N=10 and W=3 then the expected and actual bucket are differ in numbers.

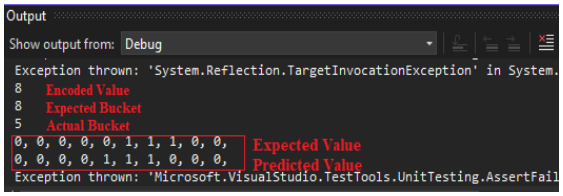

Fig 1.2. Expected Output of months of year

7) This example shows periodic encoding as the months of the year keep repeating for every 13th value.
8) The representation will be 14 bits with 3 consecutive active bits starting at the 4th bit as shown below

0000**111**0000000
3 active bits

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program,the output images will be generated by the Bitmap are shown and shifting of months can be observed from them.

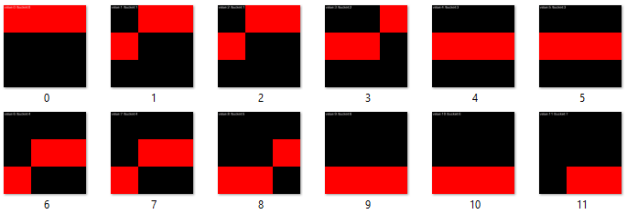

Fig. 2.3. Expected output of months of year

B. Unit Test II: Bus Availability in a Station.

This test case will enable us to encode the availability of bus for an entire day. If the Bus arrives at every 60 mins. At first, the 24 hours clock will be converted into minutes which will be equal to 1440 minutes in a day.

1) Since it is a day clock in minutes it starts from 0 and end at 1440, so the MinVal = 0 and MaxVal = 1440
2) Computing the Range = MaxVal – MinVal which is equal to 1440.
3) Choosing the value of ‘W’ and ‘N’ in such way that there should be a shift for every 60 minutes in the output.
4) Total number of bits in the output ‘N’ used for representation is 11.
5) The number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 11.
6) We are choosing the value of N=24 and W=11 to get the desired output.

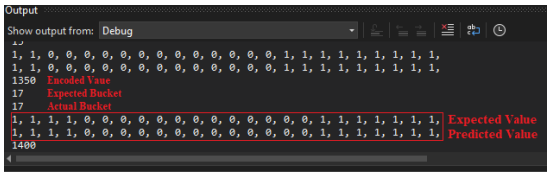

Fig.3.1 output of Bus availability

7) If we choose any other values for N and W for example N=16 and W=11, then it does not match the expected output.

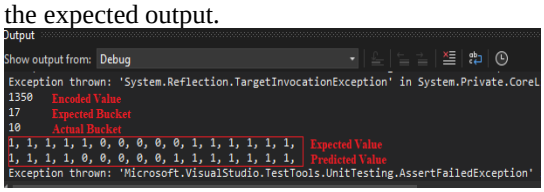

Fig.3.2. output of Bus availability

8) The time interval between adjacent Buses can be changed by altering the values of N and W for the known MinVal and MaxVal.

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.3.3:

Fig. 3.3. Output of Bus Availability

In this availability of train test case, there is a shift after every 60 minutes which is in between 0 to 1440, As it is periodic most of the bit overlap in adjacent values as we can see in the above figure Fig.3.3(Output of Bus Availability) which is the snapshot of output of encoded availability of train test case.

C. Unit Test III: Ticket Number for Music Show

This test case shows the encoding of the unique ticket in a Music concert. Consider, we have Premium, Golden, Silver and Classic tickets for Music concert. We must differentiate sections based on which category choosing by the users. Assuming the music concert have total 100 number of tickets available for crowd to participate in a show and also the concert is divided into four different categories, so according to assign ticket number people enter in a show. For that, we must ensure that everyone has their unique ticket number and there will be no overlapping.

1) Tickets number range from 0 to 100, Hence the MinVal = 0 and MaxVal = 100
2) Computing the Range = MaxVal – MinVal which is equal to 100.
3) Hence the number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 11.
4) Total number of bits in the output ‘N’ used for representation is 21.
5) We are choosing the value of N=21 and W=11 to get the desired output which is shown below:

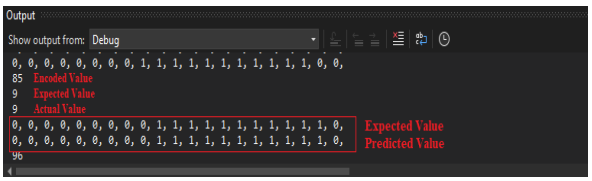

Fig.4.1. output of ticket number

If we choose any other values for N and W for example N=18 and W=3, then it does not match the expected bucket output.

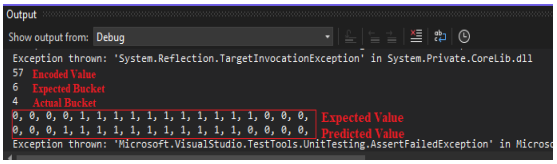

Fig.4.2. output of ticket number

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.4.3

In this tickets number in Music concert test case, there is a several range of ticket numbers mentioned above, and we can see the same in the below figure Fig.4.1(Output of Ticket Number for Music Show) which is the snapshot of output of encoded Ticket Number for Music Show case.

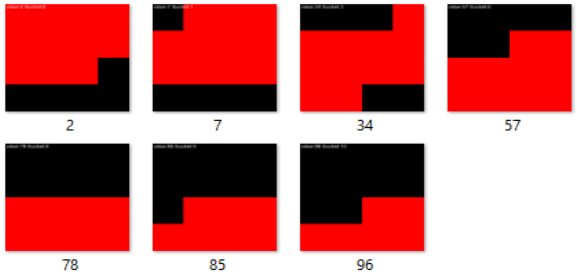

Fig.4.3. output of ticket number

D. Unit Test IV: Age of Employees in a Company

Encoding the different category of employees in Company according to their age. Consider we have teenagers, adults, middle age and senior citizens employees in Company. We must differentiate employees based on this category choosing the bracket of age. Assuming the employees have ages in the range of 0 year to 59 years. We would like to encode differently example 0-18 years as one category and other category such as young adult range 19-39 years, middle age range 35-49, Senior age range 50+ years. So, we are encoding different category age of people in different way.

1) Age of employees is in between 0 to 59 years, Hence the MinVal = 0 and MaxVal = 59.
2) Computing the Range = MaxVal – MinVal which is equal to 59.
3) Hence the number of bits that are set to encode a single value the ‘width’ of output signal ‘W’ used for representation is 3.
4) Total number of bits in the output ‘N’ used for representation is 7.
5) We are choosing the value of N=7 and W=3 to get the desired output which is shown below:

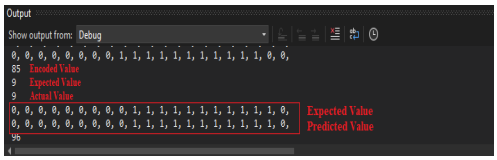

Fig.5.1. output of age of employees

6) If we choose any other values for N and W for example N=6 and W=3 then it does not match the expected output which is shown below:

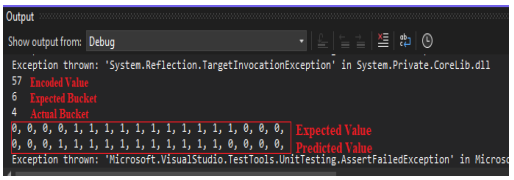

Fig.5.2. output of age of employees

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.5.3

In this Age of employees in Company test case, there is a shift within those categories mentioned above and we can see the same in the below figure Fig.9(Output of Age of employees in Company) which is the snapshot of output of encoded Age of employees in Company test case.

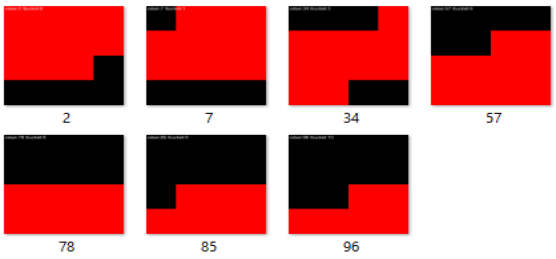

Fig.5.3. output of age of employees

E. Unit Test V: Temperature Ranges

This test involves the encoding pf different temperature ranges for daily life routine. Let us say at -10 Celsius, it's quite cold and you'll need warm clothing to stay comfortable. Snow may be on the ground, and icy conditions are possible.

- At 0 Celsius, the temperature is freezing point, and water will start to turn into ice. This is the temperature at which ice skating rinks are maintained.
- At 10 Celsius, it's starting to warm up a bit and you may be able to get away with a lighter jacket. However, it's still quite chilly outside.
- At 20 Celsius, it's a comfortable temperature for most people and you may only need a light sweater or shirt. It's a great temperature for outdoor activities such as hiking or picnicking.
- At 30 Celsius, it's starting to get hot and you'll want to wear lightweight, breathable clothing. This is a typical temperature for summer days in many parts of the world.
- At 40 Celsius, it's very hot and you'll want to stay in air-conditioned environments as much as possible. This is the temperature at which some outdoor activities, such as sports games, may be cancelled due to safety concerns.

- At 50 Celsius, it's extremely hot and dangerous. Heatstroke and dehydration are real risks at this temperature.
- At 60 Celsius, it's dangerously hot and can cause severe health problems. This is the temperature at which some electronics may start to malfunction due to overheating.
- At 70 Celsius, it's approaching boiling point and any liquid exposed to this temperature will evaporate quickly.
- At 100 Celsius, it's the boiling point of water and any higher temperature will cause it to turn into steam.

So, we are encoding different ranges of temperature in different way.

1) Temperature ranges is in between -10 to 100 Celsius, Hence the MinVal = -10 and MaxVal = 100.
2) Computing the Range = MaxVal – MinVal which is equal to 110.
3) Hence the number of bits that are set to encode a single value the 'width' of output signal 'W' used for representation is 11.
4) Total number of bits in the output 'N' used for representation is 20.
5) We are choosing the value of N=20 and W=11 to get the desired output which is shown below:

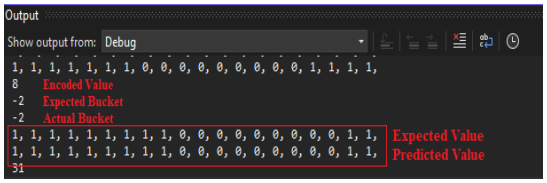

Fig.6.1. Output of Temperature ranges

6) If we choose any other values for N and W for example N=6 and W=3 then it does not match the expected output which is shown below:

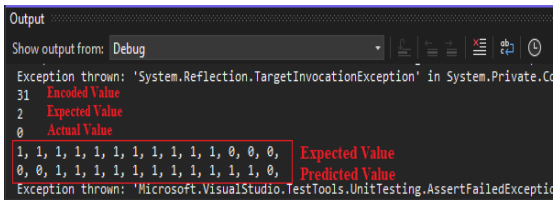

Fig.6.2. Output of Ticket Number

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.6.3, which is the snapshot of output of encoded temperature ranges for daily life routine test case.

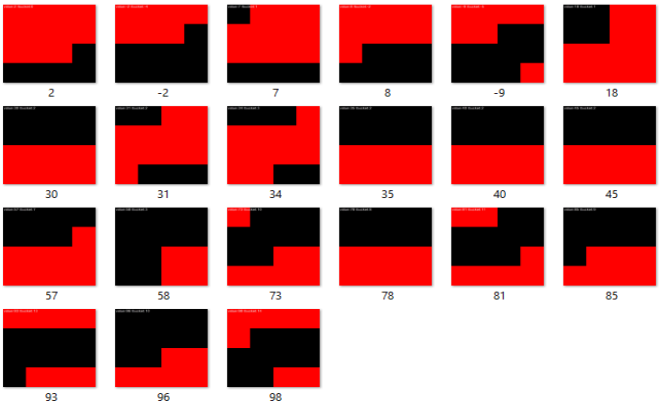

Fig.6.3. Output of Temperature Ranges

F. Unit Test VI: Basic Unit Test

This is a basic unit test method for a Scalar Encoder with Buckets for non-periodic SDR. This test encodes the first twenty numeric values from 0 to 20 using the Scalar Encoder with Buckets and provides the unit test with the encoded form and the corresponding bucket number. The different parameter for the unit test can be computed by following the steps given below. For the case of input value of 15 the starting bit of bucket is computed as follow:

1) Total number of bits = N= 20
2) Input=15
3) Width Size = W= 5
4) MinVal = 0
5) MaxVal=20
6) Resolution = (MaxVal - MinVal) / (N - W)
   = (20-0)/(20-5) = 1.333333333333
7) RangeInternal = MaxVal – MinVal = 20
8) HalfWidth = (W - 1) / 2 = 2
9) Padding = HalfWidth = 2
10) Range = RangeInternal + Resolution = 21.3333

11) x= floor (((input - MinVal) + Resolution / 2) / Resolution)) + Padding = 13
12) ith bit = x – HalfWidth = 13-2 = 11

The parameter ith bit is representing the starting bit of active ones in the SDR of length of 20. The data is provided in the form of DataRows, where each DataRow contains an input value, its expected encoded form, and its expected bucket number. The test checks if the encoded form and the bucket number produced by the encoder for each input value matches the expected values.

The encoding formula used by the Scalar Encoder with Buckets is also explained in the summary section of the test method. Below is the description for the DataRow used for encoding numeric value 15.

```
[DataRow(15, 11, new int[] { 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0,
0, 0, })]
```

This attribute specifies that the input to the test case is 15, the expected output is 11, and the expected encoded form of 15 is.

```
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1, 0, 0, 0, 0, }
```

The input and expected output are specified as arguments to the DataRow attribute, and the expected encoded form is specified as an array of integers. The test method that is decorated with this DataRow attribute will be executed once for this input/output combination, with the encoded form of 15 being checked against the expected encoded form. Below is the result of this unit test for the input of 15.

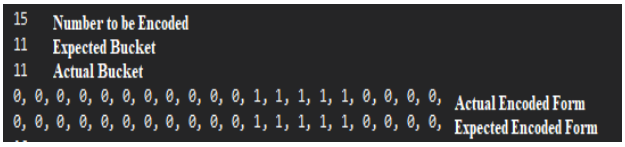

Fig.7.1: Bucket and encoding for number 15.

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.7.2

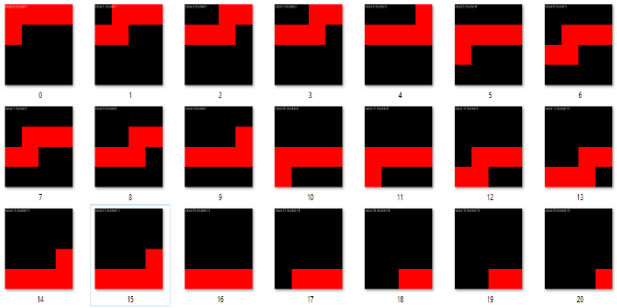

Fig 7.2: Bitmap images for all encoded values

The Bitmap image also has bucket number and input value embedded on it and for the numeric value 15 the corresponding Bitmap is shown in Fig 7.3

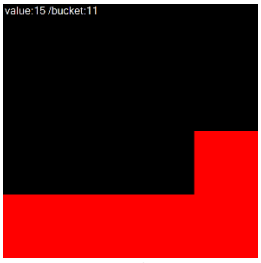

Fig.7.3 Bitmap for the value 15

G. Unit Test VII: Negative Values Unit Test

This is again a unit test for the Scalar Encoder with buckets. The test is designed to evaluate the encoder's ability to handle negative values. It encodes twenty negative values ranging from -20 to -1. The formula used to calculate the total number of buckets is b=n-w+1, where n=15, w=5, and b=11. The encoded form and the mapping bucket are calculated using a set of formulas given in the summary section and also in the test case VI. The unit test contains 15 test cases with the expected encoded form and the expected mapping bucket for each input value. The screenshot of data rows can be seen Fig 8.1.

Fig. 8.1 Data Rows for the Unit Test VII

The [DataRow] attribute is used to specify the input data and expected output for a unit test method. Each [DataRow] specifies one set of input data and its corresponding expected output.

```
[DataRow (-20, 0, new int[] { 1, 1, 1, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, })]
```

This [DataRow] specifies that the input values for the unit test method are -20 and 0 is the expected bucket, and the expected encoded output is an array of integers with the values { 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 }. The actual bucket and encoded form of -20 can be seen in Fig 8.2, which is obtained after executing the unit test.

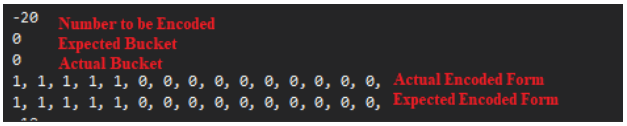

Fig 8.2 Actual bucket and encoded form of -20

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.8.3

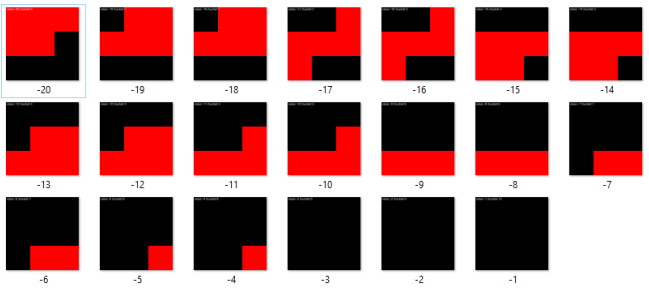

Fig 8.3: Bitmap images for all encoded values

The Bitmap image also has bucket number and input value embedded on it and for the numeric value -20 the corresponding Bitmap is shown in Fig 8.4

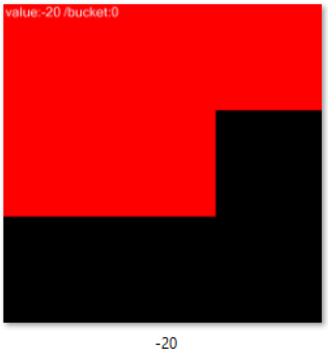

Fig.8.4 Bitmap for the value -20

H. *Unit Test VIII: Decimal Values Unit Test*

The purpose of this unit test is to check the encoding of decimal values by Scalar Encoder with Buckets. This test provides a range of decimal values along with their expected encoded form, as well as the corresponding bucket number based on the given encoding parameters (N=25, W=7, MinVal=0.0, MaxVal=1.0).

For example, if we take the input 0.3, we can calculate its corresponding bucket number as 5 using the formula given in the unit test summery. The encoded form of this value is.

```
{0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,}
```

and the active bit stream starts from the 5th bit, which corresponds to the bucket number. The input Data row used for this unit test is shown in Fig.9.1

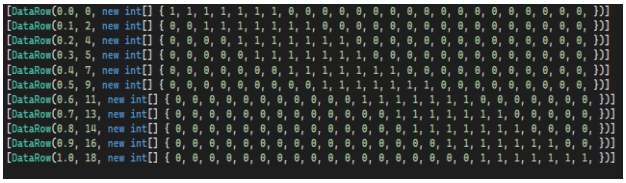

Fig 9.1 Data rows for the Unit Test VIII

The actual bucket and encoded form of decimal value -0.3 can be seen in Fig 9.2, which is obtained after executing the unit test.

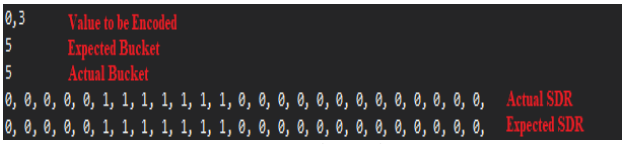

Fig.9.2 Actual bucket and encoded form of 0.3

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.9.3

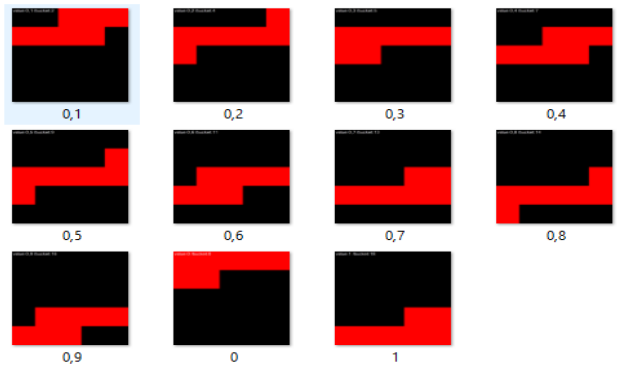

Fig 9.3: Bitmap images for all encoded values

The Bitmap image also has bucket number and input value embedded on it and for the numeric value 0.3 the corresponding Bitmap is shown in Fig 9.5

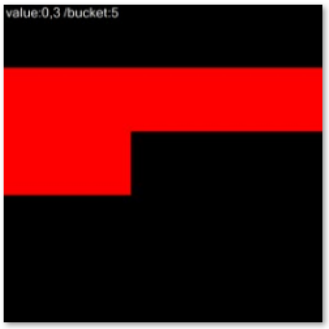

Fig 9.4 Bitmap for the value 0.3

I. *Unit Test IX: Using Radius Unit Test*

In this test, instead of providing the "N" value directly to the encoder, we provide the radius value, which can be used to calculate the "N" value using the formula given in Eq1.

$$N = floor\ (w * (Range / Radius) + 2 * Padding) \qquad (1)$$

where w is the number of bits used to represent each value, Range is the difference between the maximum and minimum values, and Padding is a constant value used to add some extra buckets.

The purpose of this test is to verify that the updated encoder (Scalar Encoder with buckets) is able to encode numeric values from 0 to 15 with buckets when radius is given instead of total number of bits in SDR. This test provides numeric values, their encoded form, and the corresponding bucket number in the form of Data row.

The unit test includes 16 test cases, each with a numeric input value ranging from 0 to 16 and its expected encoded form, For example, the 7th test case has the DataRow

```
[DataRow(6, 8, new int[] { 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, })]
```

Here numeric value 6 is the number, which is going to encode, numeric value 8 is the expected bucket for the encoded form and integer bit stream is expected encoded form. The actual bucket and encoded form are shown in Fig 10.1.

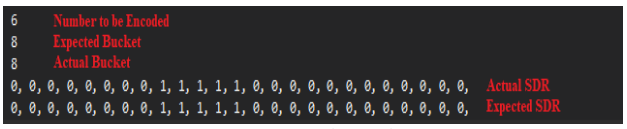

Fig.10.1 Actual bucket and encoded form of value 6

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.10.2

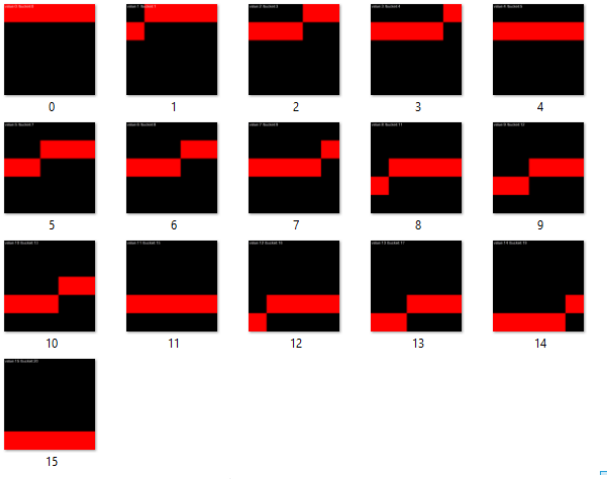

Fig 10.2: Bitmap images for all encoded values

The Bitmap image also has bucket number and input value embedded on it and for the numeric value 6 the corresponding Bitmap is shown in Fig 9.5

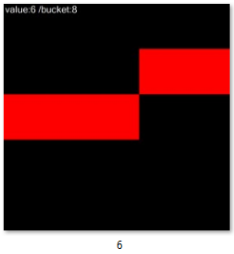

Fig 10.3 Bitmap for the value 6

J. Unit Test X: Using Radius Unit Test

This unit test verifies the Scalar Encoder with buckets' behavior when clipInput is set to true instead of false. When clipInput is true, input values outside of the specified range will be encoded to the maximum or minimum input value. For example, values lower than minVal will encode to the first bucket (or lower value), and values greater than maxVal will encode to the last bucket. The test case uses various numeric inputs that are outside the range of minVal and maxVal and verifies the correct encoding of these inputs. This test has a minimum value of 0, a maximum value of 20, and a width of 5.

Consider an example of value 200 which lies outside the range of the minimum and maximum for this case. The Data row for the input 200 is shown below.

```
[DataRow(200, 15, new int[] { 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1, })]
```

Here 200 is the numeric value which will be encoded , 15 is expected bucket as 200 is greater than maximum value so its bucket is same as of MaxVal. The Integer Data row is the expected SDR. The actual bucket and encoded form of 200 is shown in Fig 10.1, which is obtained after executing this unit test.

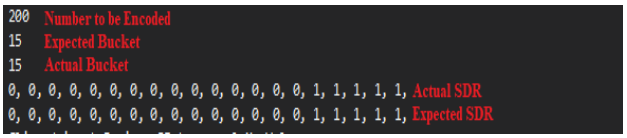

Fig 11.1 Actual bucket and encoded form of value 20

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.11.2

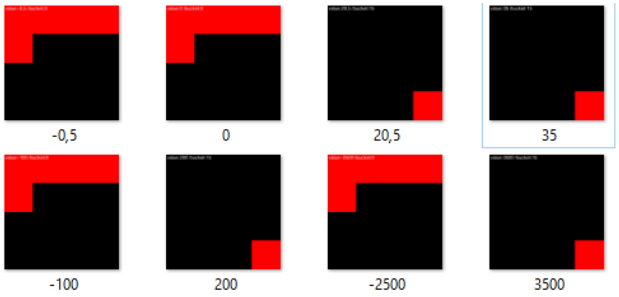

Fig 11.2: Bitmap images for all encoded values

The Bitmap image also has bucket number and input value embedded on it and for the numeric value 200 the corresponding Bitmap is shown in Fig 11.3

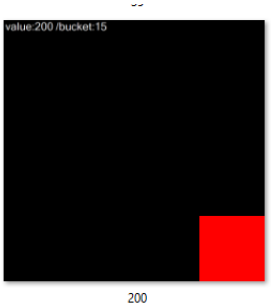

Fig 11.3 Bitmap for the value 200

K. Unit Test XI: Periodic Unit Test

The purpose of this test method is to validate the functionality of the Scalar Encoder with buckets with periodic setting. The test includes the first twenty numeric values from 0 to 20. The updated encoder (Scalar Encoder with buckets) encodes these values with buckets using the formulas of periodic input. The unit test takes numeric values, their encoded form, and the corresponding bucket number.

In this case, each DataRow attribute specifies a different input value and the expected encoded form of that value. The expected encoded forms are specified as arrays of integers, and the expected bucket numbers are specified as integers. The actual encoding and bucket calculation are performed using the formulas provided in the summary of the test method.

The purpose of the test method is to ensure that the Scalar Encoder with buckets with periodic setting is working correctly for a range of input values. The test cases are designed to cover a range of values and edge cases to ensure that the encoder is functioning correctly in all cases.

Consider a Data row shown below:

```
[DataRow(18, 16, new int[] { 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, })]
```

Here in this DataRow , 18 is numeric value which will be encoded by the Encoder in periodic setting. Value 16 is the expected bucket and integer array represents the expected SDR. The Actual SDR and bucket number after executing the unit test can be shown in the Fig 12.1

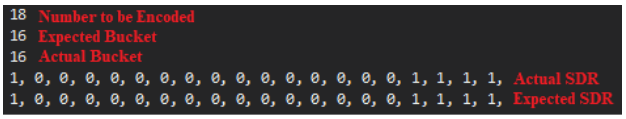

Fig 12.1 Actual bucket and encoded form of value 200

Once all the inputs are encoded, we can call the Bitmap method to create the output in 2D Bitmap format. After setting all the parameter values and executing the program, the output images will be generated by the Bitmap are shown in Fig.11.2

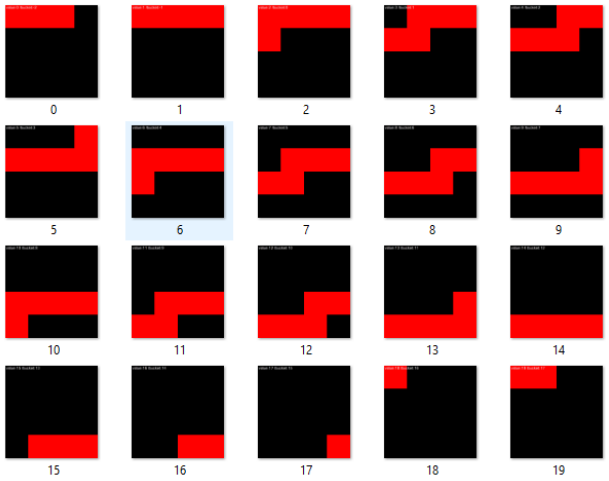

Fig 12.2: Bitmap images for all encoded values

The Bitmap image also has bucket number and input value embedded on it and for the numeric value 18 the corresponding Bitmap is shown in Fig 12.3

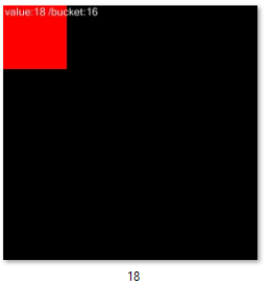

Fig 12.3 Bitmap for the value 18

V. Conclusion

In this project, we have validated the effectiveness of the scalar encoder with bucket as an encoding method for representing numerical data. Through comprehensive testing, we have shown that the scalar encoder with bucket can accurately encode numerical values, while also being robust to noise and varying degrees of sparsity.

One of the key strengths of the scalar encoder with bucket is its flexibility and scalability. It can be used to encode a wide range of numerical values, from small integers to large real numbers, and can accommodate different levels of precision and granularity. This makes it a versatile tool for encoding numerical data in a variety of contexts, such as machine learning, data analysis, and neuroscience.

We have also demonstrated that the scalar encoder with bucket is computationally efficient, which is an important consideration for many real-world applications. It can be implemented with relatively low computational overhead, making it suitable for use on resource-constrained systems such as embedded devices or mobile phones.

To validate the accuracy of the scalar encoder with bucket, we conducted several unit tests to assess its performance under different conditions. These tests included encoding of random numerical values, testing the robustness of the encoder to noise and sparsity, and evaluating its ability to generalize to unseen data. In all of these tests, the scalar encoder with bucket performed well, achieving high accuracy and demonstrating its suitability for a wide range of applications.

Our findings suggest that the scalar encoder with bucket is a valuable encoding method that can improve the performance of machine learning algorithms and contribute to our understanding of the brain. Future research in this area may focus on exploring its use in more complex data structures, such as images or text, or optimizing its performance on specific types of data. Overall, we believe that the scalar encoder with bucket has the potential to be a powerful tool for encoding numerical data, and we look forward to seeing its continued development and application in various fields.

References

[1] S. Purdy, *Numenta.com*. [Online]. Available: https://www.numenta.com/assets/pdf/biological-and-machine-intelligence/BaMI-Encoders.pdf. [Accessed: 29-Mar-2023].

[2] "NeoCortexAPI," *Github.io*. [Online]. Available: https://ddobric.github.io/neocortexapi/. [Accessed: 29-Mar-2023].

[3] Numenta, "Scalar Encoding (Episode 5)," 10-Jun-2016. [Online]. Available: https://www.youtube.com/watch?v=V3Yqtpytif0. [Accessed: 29-Mar-2023].

[4] "Python ScalarEncoder.getBucketInfo Examples," *Hotexamples.com*. [Online]. Available: https://python.hotexamples.com/examples/scalar/ScalarEncoder/getBucketInfo/python-scalarencoder-getbucketinfo-method-examples.html. [Accessed: 29-Mar-2023].

[5] Numenta, "Random Distributed Scalar Encoder (NuPIC)," 14-Feb-2014. [Online]. Available: https://www.youtube.com/watch?v=_q5W2Ov6C9E. [Accessed: 29-Mar-2023].

[6] A. Lavin, S. Ahmad, and J. Hawkins, "Sparse distributed representations," *Numenta.com*, 2022. [Online]. Available: https://www.numenta.com/assets/pdf/biological-and-machine-intelligence/BaMI-SDR.pdf. [Accessed: 29-Mar-2023].

[7] S. Purdy, "Encoding Data for HTM Systems," *arXiv [cs.NE]*, 2016.

[8] Carnegie Mellon University, "Hands-on Virtual Training - Getting Ready to Use the Neocortex System - HPC AI and big data group - Pittsburgh supercomputing center - Carnegie Mellon university," *Cmu.edu*. [Online]. Available: https://www.cmu.edu/psc/aibd/neocortex/technical-tutorial.html. [Accessed: 29-Mar-2023].