NTU

NATIONAL INSTITUTE OF TECHNOLOGY (KOSEN), KUMAMOTO COLLEGE

Indian Institute of Technology Bhubaneswar

Bio-inspired CODECs using steganography

Dr Pedro Machado, pedro.machado@ntu.ac.uk, Nottingham Trent University, Nottingham, UK

Dr Isibor Kennedy, Ihianle Isibor.ihianle@ntu.ac.uk, Nottingham Trent University, Nottingham, UK

Dr Andreas Oikonomou, andreas.oikonomou@ntu.ac.uk, Nottingham Trent University, Nottingham, UK

Dr Srinivas Boppu, srinivas@iitbbs.ac.in, Indian Insitute of Technology Bhubaneswar, Bhubaneswar – India

Dr Minoru Motoki, motoki@kumamoto-nct.ac.jp, National Institute of Technology (KOSEN) - Kumamoto College, Kumamoto – Japan

YouTube channel

Outline

- Overview
- Related work
- Methodology
- Results Analysis
- Conclusion and Future Work

Staff Profile

Hypothesis

- Can we transmit sound embedded in the image?

Why/How?

- Consider **underwater communication**. Wireless signals are limited there. Images could carry sound.
- Also, think about **raster plots**. They are hard to read. Embedding sound might make them clearer.

Overview

- Spiking Neural Networks (SNNs) are a type of artificial neural network that more closely mimics the way biological neurons in the brain communicate through discrete electrical pulses called spikes.
- Unlike traditional neural networks that process continuous values, SNNs leverage the timing of these spikes to encode and process information, making them potentially more power-efficient and adept at handling tasks involving temporal data.

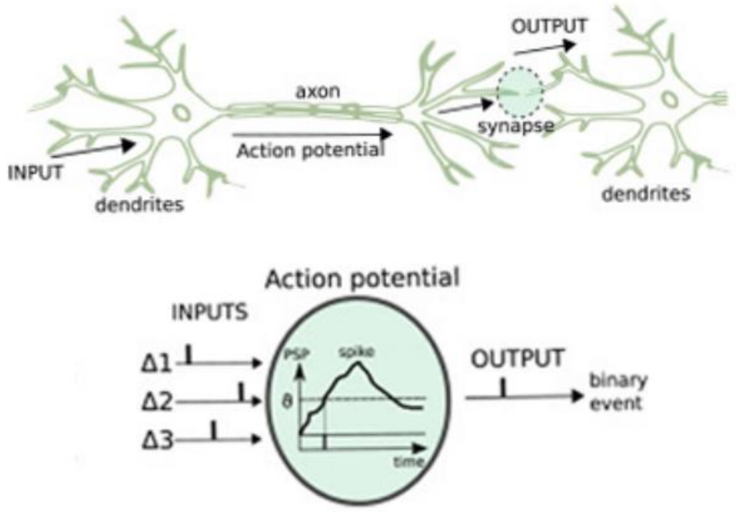


Fig. 3. Biological neuron and association with artificial spiking neuron

NTU

SNN Concepts

- **Event-Driven Processing:** Unlike traditional ANNs that process all data simultaneously, SNNs are event-driven. They only fire neurons when a significant change or "spike" occurs in the input, naturally leading to sparse data representation.
- **Spike-Timing Dependent Plasticity (STDP):** SNNs can utilise STDP, a biologically inspired learning rule, to learn relevant temporal patterns and prioritise information. Allows the network to become highly efficient at encoding only the most salient features.
- **Remote Supervised Method (ReSuMe):** upervised learning approach for SNNs that enables them to learn precise spatio-temporal spike patterns by integrating spike-based Hebbian learning with a novel remote supervision concept.
- **Efficient Information Representation:** Information is encoded in the precise timing of spikes, rather than continuous values. Allows for a much more compact representation of data, as a single spike or a few spikes can convey a significant amount of information.
- **Bio-Inspired Compression:** By mimicking the brain's energy-efficient and event-driven processing, SNNs offer a novel paradigm for data compression that inherently focuses on dynamic information, potentially leading to highly efficient encoding for media.

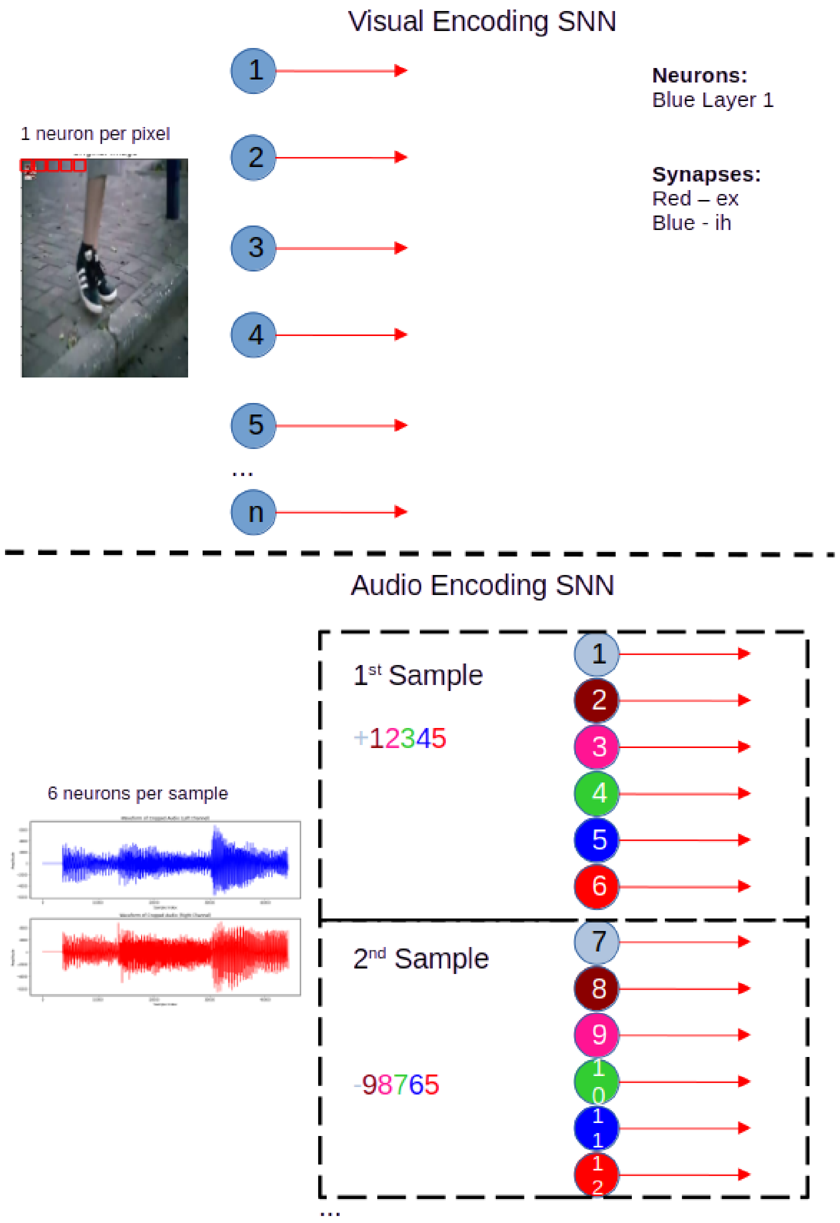
NTU

Methodology

- NEST simulator was selected for its comprehensive documentation, examples, and simulation capabilities, using Oracle Virtualbox and Python with Jupyter Notebook for implementation, and various NEST library packages for network creation and monitoring.
- Evaluation of the SNN's performance involved comparing the generated and desired spike trains using the Victor Purpura (VP) and Van Rossum (VR) spike metrics.
- LIF neurons were employed to build the network, and experiments were conducted to explore their behaviour based on threshold voltage and current configurations, with spike recorders attached to input and output neurons for comparison.

Methodology - Network Architectures

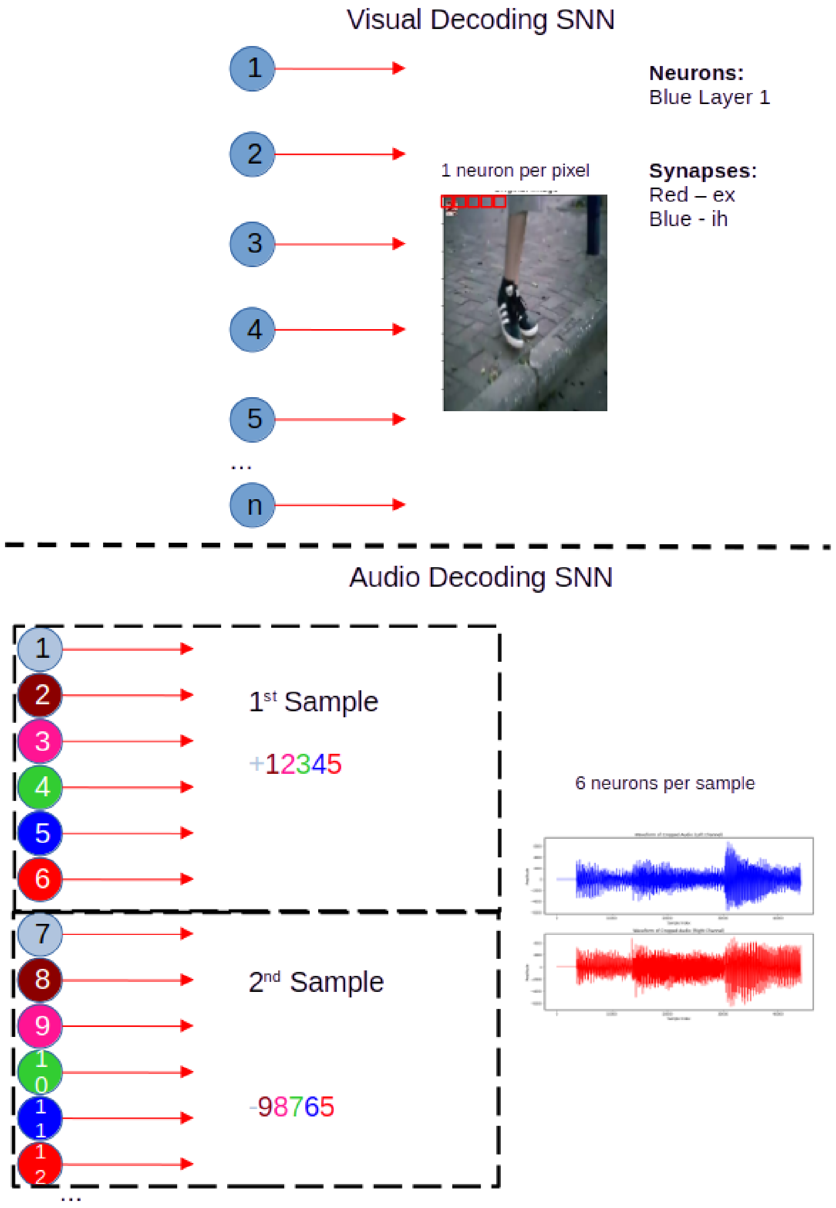
- Encoding SNN
  - Visual encoding: Neurons = n rows * m cols * 3 channels (Red, Green and Blue)
  - Audio encoding: Neurons = 6 neurons * num of sample * 2 channels (Left and Right)



NTU

Methodology - Network Architectures

- Decoding SNN
  - Visual encoding: Neurons = n rows * m cols * 3 channels
  - Audio encoding: Neurons = 6 neurons * num of sample * 2 channels

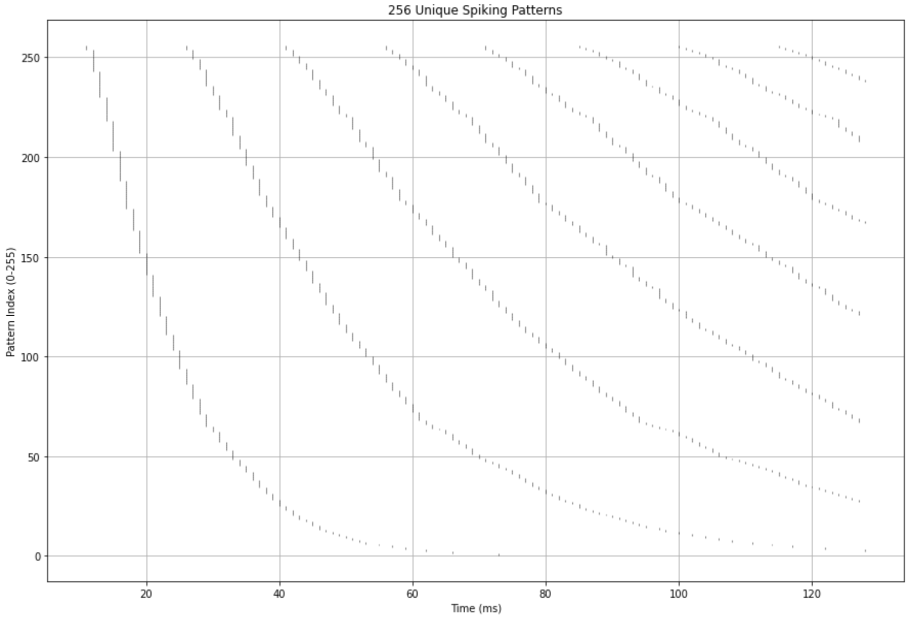


NTU

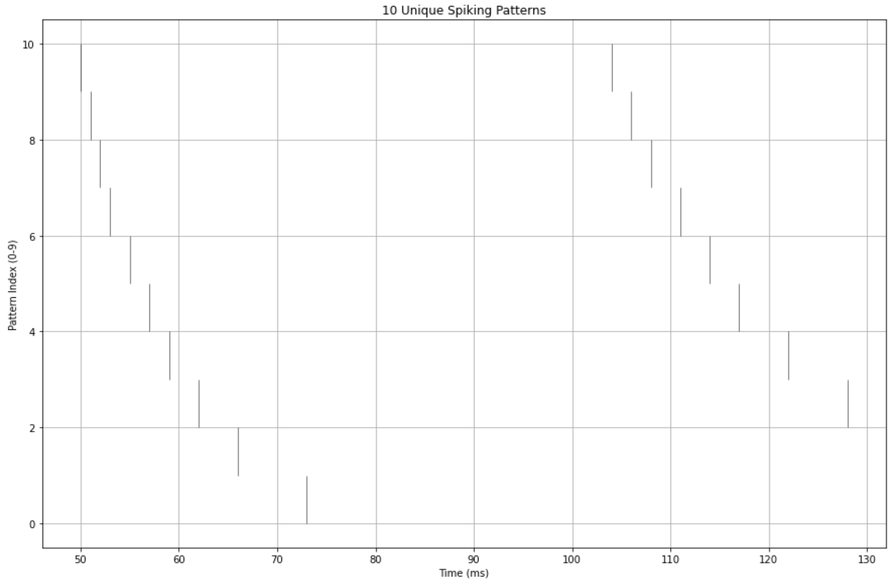
Temporal encoding

- 256 unique patterns to encode the pixel values (image encoding).
- 10 unique patterns to encode numbers from 0 to 9 (audio encoding)
- 2 out of the 10 unique patterns used to represent the signal (audio encoding)

Uint8 -> values in the range [0, 255]



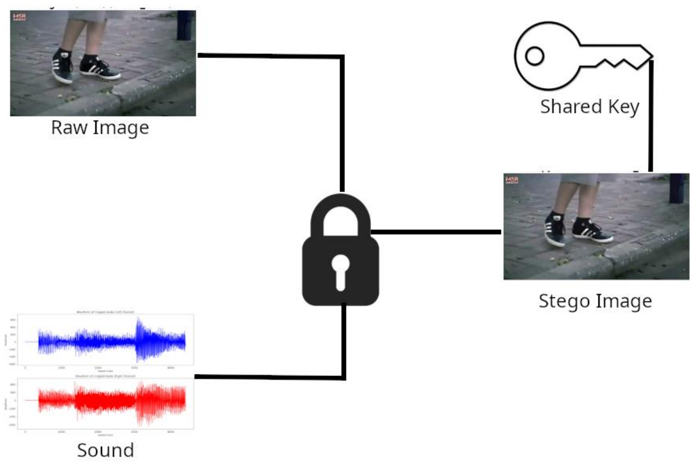
Sound samples Range [–99,999, 99,999]



NTU

Steganography

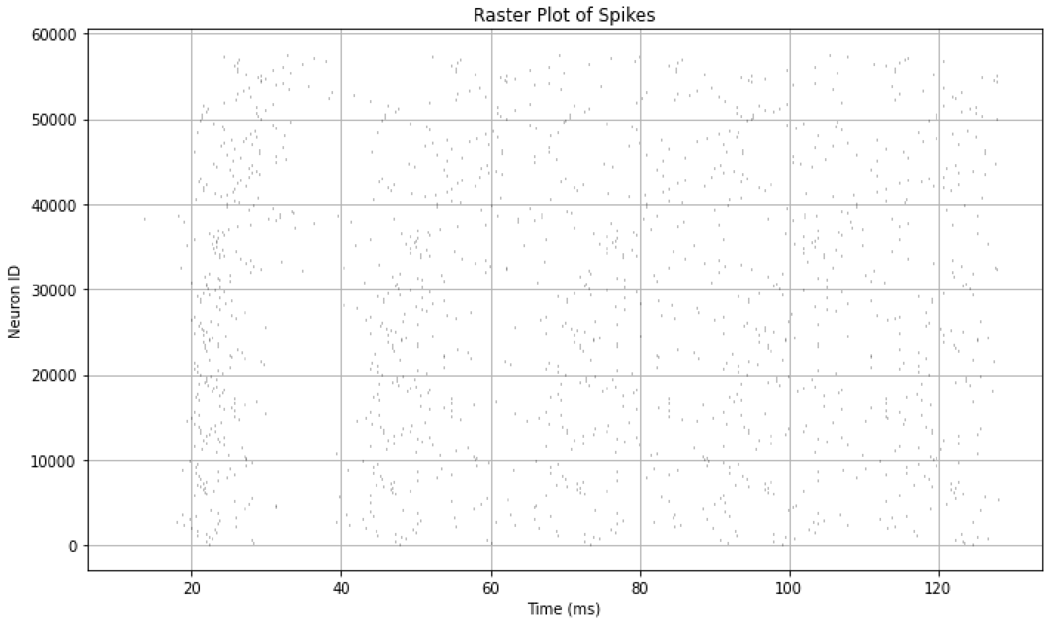
- **Concealing Information:** Steganography is the art and science of hiding one piece of information within another, in such a way that the existence of the hidden information is not readily apparent to an observer.
- **Invisible Communication:** Unlike cryptography, which scrambles data to make it unreadable without a key, steganography aims to make the very presence of the message undetectable.
- **Media as Cover:** Digital steganography often uses various forms of media, such as images, audio, or video files, as "cover" for the hidden data.
- **Leveraging "Dark" Pixel Values (0-9):** leveraging the darkest pixel values (0-9). These are often less perceptible to the human eye, making them ideal for subtle modifications.
- **Mapping Pixel Values (0-9 to 10):** applying a transformation where pixel values from 0 to 9 are mapped to a value of 10. Creates a specific range or set of values for encoding.
- **Sound Slices and Frame Distance:** The sound data is divided into slices, with each slice corresponding to the duration between two frames (e.g., for 48,000 samples/second and 30 frames/second, each image accounts for 1600 samples of sound).
- **Mapping Patterns to Samples (0-9):** Within each sound slice, values from 0 to 9 are used to map specific patterns, with each pattern corresponding to a sound sample. This implies a method of encoding sound data by subtly altering the "dark" pixel values based on these patterns.



NTU

Typical Visual output

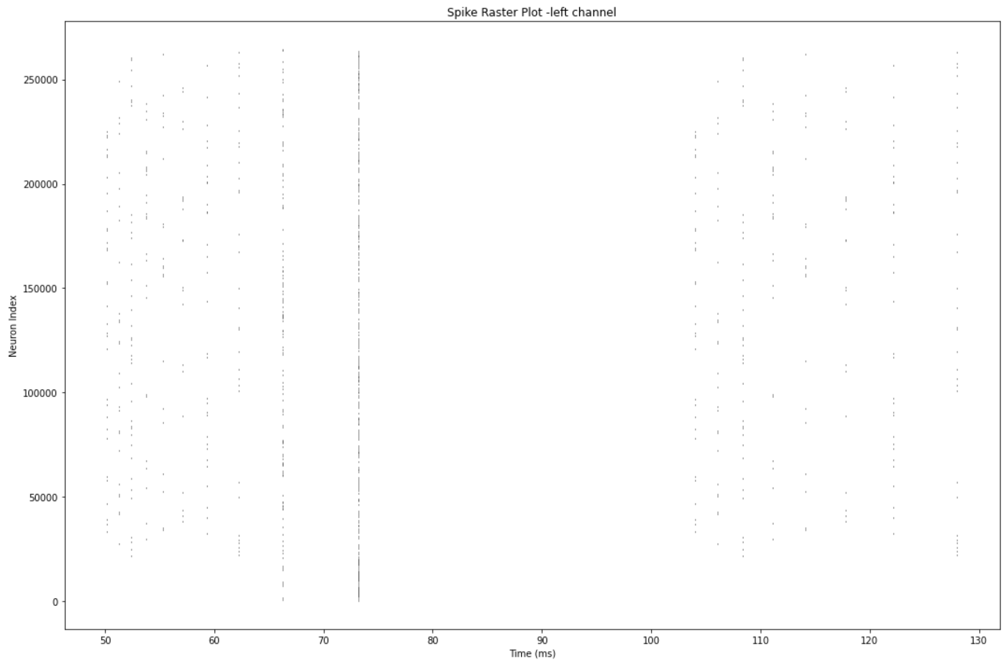
- A corresponding raster plot will be generated by each image
- To check for errors, we have calculated the absolute difference between the original and reconstructed images



NTU

Typical Audio output

- A corresponding raster plot will be generated by each time slice
- To check for errors, we have calculated the absolute difference between the original and reconstructed time slices



Steganography errors

- To check for errors, we have calculated the absolute difference between the original and reconstructed time slices
- As demonstrated below, while there are errors associated to the reconstruction process, these are not visible to the human eye.

Original (Color) (frame_0000)

Mapped (Color) (frame_0000)

Difference (Color) (frame_0000)

Binary Diff (Color) (frame_0000)

Target platforms

AMD Kria K26 SoM:

- Adaptive SoC (System-on-Module) optimised for edge AI applications.
- Integrated Arm Cortex-A53 processor with FPGA fabric.
- Supports OpenCL via Xilinx Vitis toolchain.

Sundance VCS-3:

- High-performance FPGA platform for computer vision and AI.
- Features Xilinx Zynq UltraScale+ MPSoC.
- Supports OpenCL acceleration for real-time processing.

Why Use OpenCL on These Platforms?

- Efficient parallel processing for AI and vision tasks.
- Hardware acceleration without extensive HDL coding.
- Compatibility in cross platforms (including FPGAs).

NTU

OpenCL FPGA Development Workflow

Host Code Development:

- Written in C/C++ and interacts with the FPGA kernel.

Kernel Development:

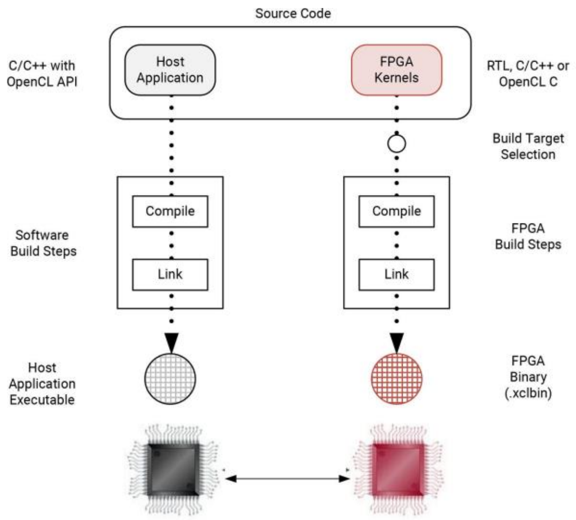
- Defined using OpenCL C and executed on FPGA hardware.

Compilation:

- OpenCL compiler translates the kernel into FPGA logic (bitstream).

Execution & Optimisation:

- Data transfer between host and FPGA, performance tuning.




```cpp
    {
        remove_all(dst_base_folder);
        create_directory(dst_base_folder);
    }
    path fn = dst_base_folder / "stats.csv";
    boost::filesystem::ofstream stats_file(fn,std::ofstream::out);
    stats_file<<"Category,# images,x,y,action_potential,fps_avg,total_time\n";
    stats_file.close();

    for (vector<path>::const_iterator it (folder_l1_list.begin()); it != folder_l1_list.end(); ++it)
    {
        path aux=*it;
        vector<path> folder_l2_list;
        if (is_directory(aux)){
            path aux_leaf=(path)aux.leaf();
            createOutputFolder(dst_base_folder,aux_leaf);
            getFolders(aux,folder_l2_list);
            for (vector<path>::const_iterator it1 (folder_l2_list.begin()); it1 != folder_l2_list.end(); ++it1)
            {
                path aux1=*it1;
                path aux_leafs=(path)aux.leaf();
                aux_leafs+=(path)"/";
                aux_leafs+=(path)aux1.leaf();
                if(is_directory(aux1))
                {
                    createOutputFolder(dst_base_folder,aux_leafs);
                    src_video_path=aux1.string();
                    vector<path> images_list;
                    getFilesInFolder(src_video_path,images_list);
                    hsmd_opencl(images_list,dst_base_folder,aux_leafs,DEBUG,RESIZE,DISPLAY,method_name,hsmdcl);
                }
            }
```

NTU

Writing an OpenCL Kernel for FPGA

Defining a Kernel:

__kernel void l1(__global const float *restrict input_f_pixel_values, __global int *restrict output_i_spike_count_l1)
{

Key Considerations:

- **Memory Access:** Use global/local memory efficiently.
- **Pipelining & Parallelism:** Optimise for FPGA hardware execution.
- **Optimisation Pragmas:** Use `#pragma unroll` and other techniques.

```c
#define C_m     10.0                        /*!<  pF - Capacity of the membrane */
#define R_m     1.0                         /*!<  MOhm - Membrane resistance */
#define V_reset 55.0                        /*!<  mV - Reset potential of the membrane */
#define V_min   55.                         /*!<  mV - Absolute lower value for the membrane potential */
#define V_th    70.0                        /*!<  mV - Voltage threshold */
#define tau_m   10.0                        /*!<  ms - Membrane time constant */
#define dt      0.1                         /*!<  ms - time step */
#define t_ref   2                           /*!< ms - Duration of refractory period */
#define PIXEL2CUR 8. //6.000000000000334 -> trajectories //4. -> CDnet
#define weight_std  1555.0                      /*!< weight */
#define coeficient  1/tau_m*dt
#define steps   10

#define PRAGMA_COEF1 16

__kernel void snn(__global const int *restrict input_f_pixel_values, __global int *restrict output_i_spike_count, const int num_neurons_per_layer)
{
    #pragma unroll PRAGMA_COEF1
// layer 1
    for(int index=0;index<num_neurons_per_layer;++index)
    {
        unsigned spike_cout_l3=0;
        if (input_f_pixel_values[index]>0.0)
        {
            bool spike_event_l1=false;
            bool spike_event_l2=false;
            bool spike_event_l3=false;
            unsigned t_rest_l1=0;
            unsigned t_rest_l2=0;
            unsigned t_rest_l3=0;
            float I_c_l2=0.0;
            float I_c_l3=0.0;
```

```cpp
    create_directory(dst_base_folder);
}
else
{
    remove_all(dst_base_folder);
    create_directory(dst_base_folder);
}

for (vector<path>::const_iterator it (folder_l1_list.begin()); it != folder_l1_list.end(); ++it)
{
    path aux=*it;
    vector<path> folder_l2_list;
    if (is_directory(aux)){
        path aux_leaf=(path)aux.leaf();
        createOutputFolder(dst_base_folder,aux_leaf);
        getFolders(aux,folder_l2_list);
        for (vector<path>::const_iterator it1 (folder_l2_list.begin()); it1 != folder_l2_list.end(); ++it1)
        {
            path aux1=*it1;
            path aux_leafs=(path)aux.leaf();
            aux_leafs+=(path)"/";
            aux_leafs+=(path)aux1.leaf();
            if(is_directory(aux1))
            {
                createOutputFolder(dst_base_folder,aux_leafs);
                src_video_path=aux1.string();
                vector<path> images_list;
                getFilesInFolder(src_video_path,images_list);
                hsmd_opencl(images_list,dst_base_folder,aux_leafs,DEBUG,RESIZE,DISPLAY,method_name);
            }
        }
    }
}
```

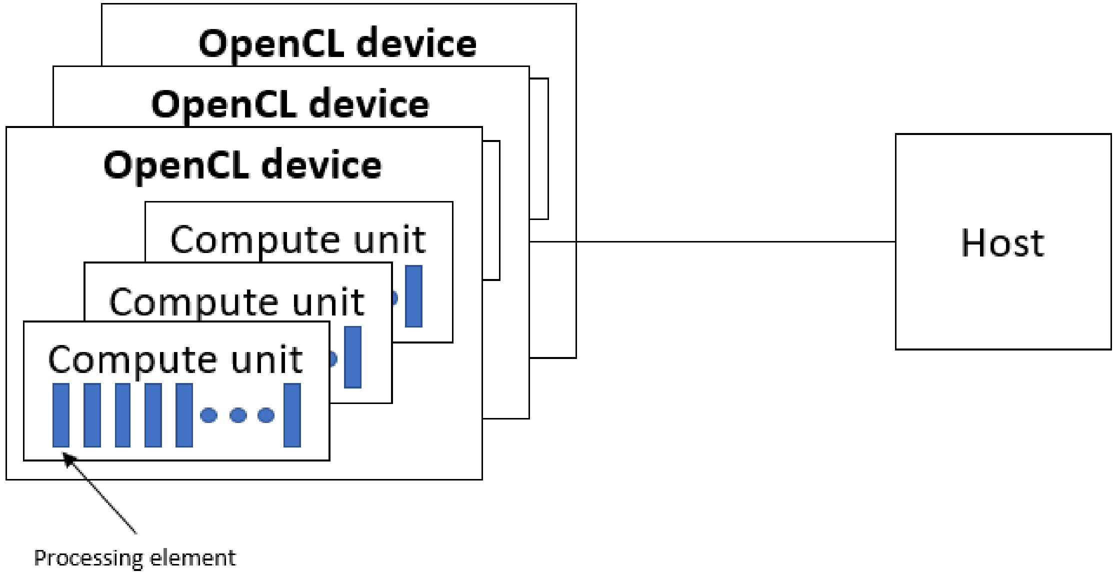
NTU

Implementing OpenCL on FPGA (Toolchains)

Use Xilinx Vitis for OpenCL-based acceleration.
Supports AI and computer vision pipelines.

Compilation Steps:

- Write OpenCL kernel and host code.
- Compile with FPGA-specific tools (v++ for Xilinx).
- Generate bitstream and execute on FPGA hardware.



NTU

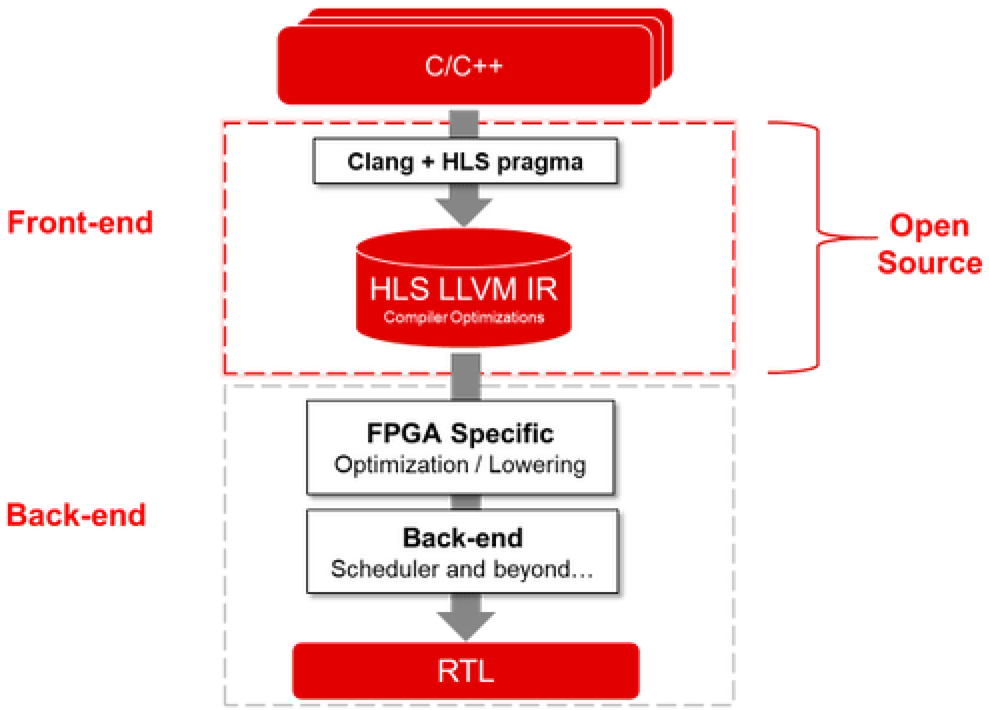
Challenges & Optimisations in OpenCL

Challenges:

- Long compilation times for bitstream generation.
- Data transfer bottlenecks between host and FPGA.
- Optimisation complexity for resource utilisation.

Optimisations:

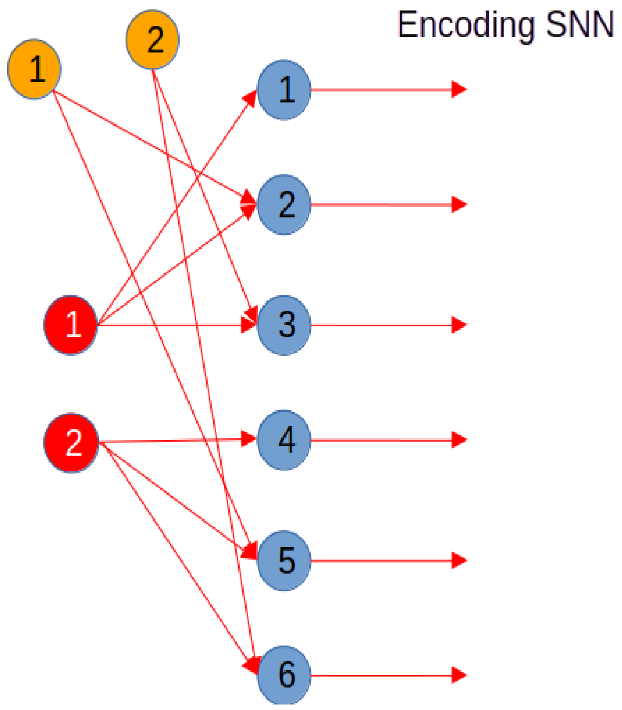
- **Loop Unrolling:** Improves performance by reducing overhead.
- **Memory Coalescing:** Aligns memory access patterns for efficiency.
- **Kernel Pipelining:** Enables continuous data flow processing.



NTU

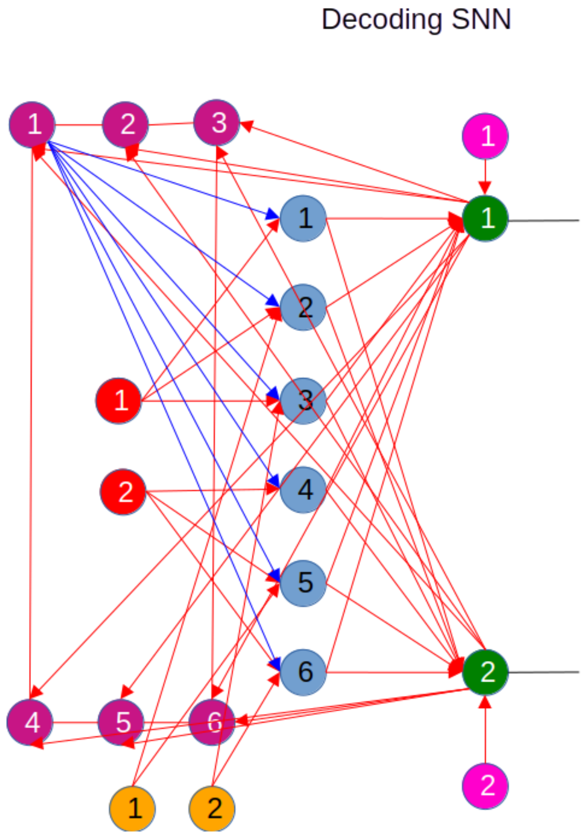
What next?- Encryption...

- We use noise to hide the original sound.
- The decoding SNN will use lateral inhibition and ReSuMe to remove the noise and recover the original audio samples.
- Using spike metrics to help training the decoding SNN.



Neurons:
Red Layer 1
Yellow Noise
Blue Layer 3

Synapses:
Red – ex
Blue - ih



Neurons:
Red Layer 1
Blue Layer 2
Green Layer 3
Purple Lateral ih
Yellow Noise
Pink Teachers

Synapses:
Red – ex
Blue – ih

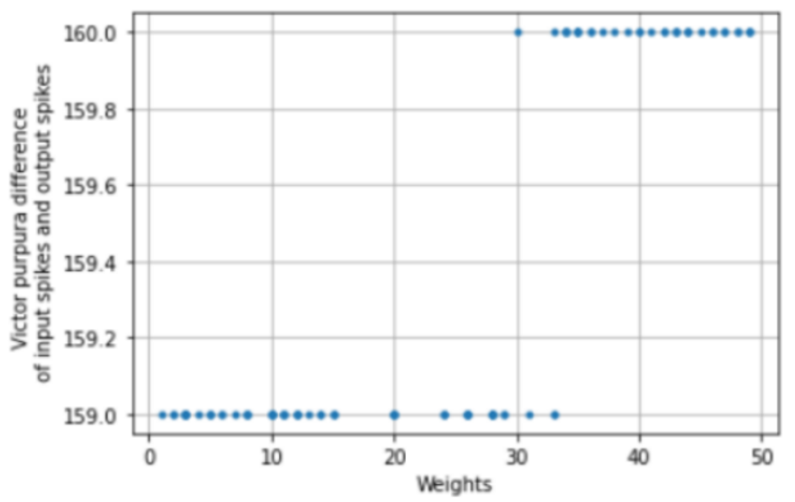
Connectivity:
L1 – L2 : 1:n
L2 – L3: n:m
NL – L2: 1:n
L3 - IL : m:n
TL – L3: 1:1
IL – IL : n:n
IL – L2 : n:m

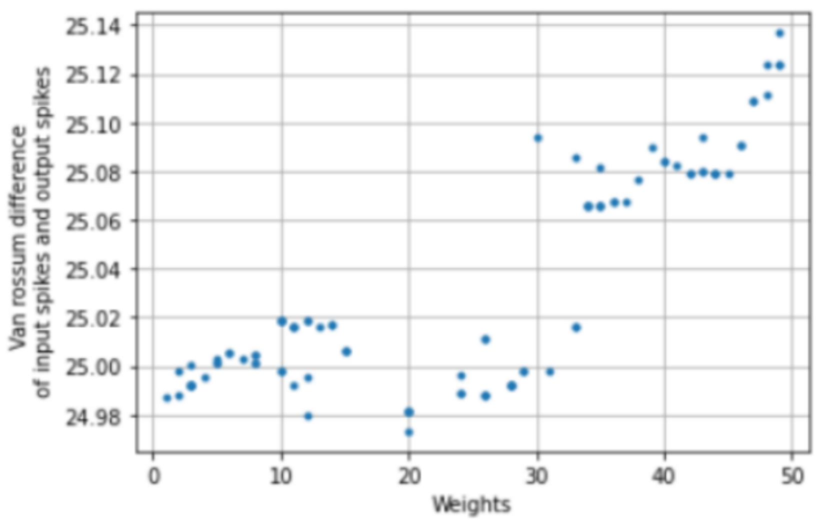
NTU

What next? - Use spike metrics...

- Utilise Victor Purpura (VP) and Van Rossum (VR) distance metrics to train a lateral inhibition layer to recovery the original signal. Spike metrics will help to decide when to stop training.

- Initial results shown that final decryption errors remained below 1.2%, validating the consistency of the SNN-based Steganography.





NTU

Takeaways

- The use of Spike metrics will help to assess when to stop training.
- The security key will be unique and could contain details on how to connect the SNN, weights and connectivity.
- Bio-inspired computing could be adopted more widely to reduce power consumption and increase the security.

NTU

Future work

- Future work should address the potential increase in noise in larger neuron populations, requiring the incorporation of noise reduction mechanisms to improve the research's applicability.
- Explore new architectures to enable the SNN to decide when to start/stop training leverage from spike metrics.
- The current approach can be expanded to consider various factors affecting the training optimisation process in LSM, including lateral inhibition, propagation delays of pre-synaptic neurons, and noise reduction techniques, especially in the context of larger neuron populations.

NTU

Partners

NATIONAL INSTITUTE OF TECHNOLOGY (KOSEN),
KUMAMOTO COLLEGE

ଭାରତୀୟ ପ୍ରଯୁକ୍ତି ପ୍ରତିଷ୍ଠାନ ଭୁବନେଶ୍ୱର
भारतीय प्रौद्योगिकी संस्थान भुवनेश्वर
Indian Institute of Technology Bhubaneswar

Ulster University

NTU
Department of Computer Science

altera™
An Intel Company

AMD
XILINX

SUNDANCE

NTU

Bio-inspired CODECs using steganography

Dr Pedro Machado, pedro.machado@ntu.ac.uk, Nottingham Trent University, Nottingham, UK

Dr Isibor Kennedy, Ihianle Isibor.ihianle@ntu.ac.uk, Nottingham Trent University, Nottingham, UK

Dr Andreas Oikonomou, andreas.oikonomou@ntu.ac.uk, Nottingham Trent University, Nottingham, UK

Dr Srinivas Boppu, srinivas@iitbbs.ac.in, Indian Insitute of Technology Bhubaneswar, Bhubaneswar – India

Dr Minoru Motoki, motoki@kumamoto-nct.ac.jp, National Institute of Technology (KOSEN) - Kumamoto College, Kumamoto – Japan

YouTube channel