Energy efficient spiking neural network neuromorphic processing to enable decentralised service workflow composition in support of multi-domain operations

Graham Bent[a], Christopher Simpkin[a], Yuhua Li[a], and Alun Preece[a]

[a]Crime and Security Research Institute, Cardiff University, Cardiff, UK

ABSTRACT

Future Multi-Domain Operations (MDO) will require the coordination of hundreds—even thousands—of devices and component services. This will demand the capability to rapidly discover the distributed devices/services and combine them into different workflow configurations, thereby creating the applications necessary to support changing mission needs. Motivated by neuromorphic processing models, in previous work it was shown that this can be achieved by using hyperdimensional symbolic semantic vector representations of the services/devices and workflows. Using a process of vector exchange the required services are dynamically discovered and inter-connected to achieve the required tasks. In network edge environments, the capability to perform these tasks with minimum energy consumption is critical. This paper describes how emerging spiking neural network (SNN) neuromorphic processing devices can be used to perform the required hyperdimensional vector computation (HDC) with significant energy savings compared to what can be achieved using traditional CMOS implementations.

Keywords: AI, Spiking Neural Networks, Service Discovery, Workflow Composition, Hyperdimensional Computing, Vector Symbolic Architecture

1. INTRODUCTION

Future Multi-Domain Operations (MDO) will require the coordination of hundreds—even thousands—of devices and component services in what has become known as the Internet of Battlefield Things (IoBT). This will demand the capability to rapidly discover the distributed devices/services and combine them into different workflow configurations, thereby creating the applications necessary to support changing mission needs. To meet these objectives, we envision a distributed Cognitive Computing System (CCS) that consists of humans and software that work together as a 'Distributed Federated Brain'.[1]

To address this challenge, our current research harnesses advancements from an emerging computing framework termed Vector Symbolic Architectures (VSA)[2] or Hyperdimensional Computing (HDC)[3] to develop light-weight algorithms for efficiently performing cognitive processing at the network edge. Vector Symbolic Architectures (VSAs) are a family of bio-inspired methods originally inspired by Hinton[4] for representing and manipulating concepts and their meanings using fixed size vector representations in a high-dimensional vector space. Eliasmith[5] has shown how these vector representations can be used to perform 'brain like' neuromorphic cognitive processing and coined the phrase 'semantic pointer' for such a vector since it acts as both a 'semantic' description of the concept and a 'pointer' to the concept. As such, they are said to be semantically self-describing. VSAs are capable of supporting a large range of cognitive tasks such as: Semantic composition and matching[6,7] ; Analogical mapping[2] ; and Logical reasoning[6,8] . Consequentially they have been used in natural language processing[6] , and cognitive modelling[5] . VSAs use vectors of very high dimensionality (D), i.e., hypervectors (HVs). For example, Plate's Holographic Reduced Representations (HRR)[9] use real-number vectors typically having ($512 \leq D < 2048$). Whereas Kanerva's Binary Spatter Codes (BSC)[10] are bit-string hypervectors (HV), typically having $D \approx 10,000$.

In a previous paper presented at SPIE 2021[11] , a VSA was described that used an hierarchical BSC binding and bundling scheme[12–14] . This new approach to workflow orchestration was used to demonstrate how MDO sensor and service descriptions could be represented as HVs and how these HVs could in turn be bound and bundled into higher-level HVs that represent sensor-service workflows. Using these workflow HV representations, the possibility to perform efficient distributed service discovery and workflow orchestration in communications and energy constrained environments that are typical of IoBT scenarios was described. An example MDO communications re-planning task was used to demonstrate the approach can be used in a simulated typical TacCIS environment with limited communications bandwidth. Using a VSA architecture to perform these types of operations is already computationally efficient, since the operations that need to be performed to discover services only requires simple vector comparison operations (e.g., Hamming Distance using logical XOR operations on binary HVs). However, in future IoBT environments energy efficiency is crucially important and so to demonstrate that some of these matching operations could be performed at ultra-low energies an ‘In Memory’ matching process, based on an experimental Phase Change Memory (PCM) device[15] ,was described. It was demonstrated that this type of device could perform the required matching operations on up to 150 10kbit HVs with an efficiency $> 100x$ that of an equivalent CMOS device.

Although PCM devices can efficiently perform the vector matching they are not able to perform the other required VSA processing operations, namely binding, bundling and unbinding that are required to implement a full VSA solution. The motivation for this paper was to demonstrate the possibility to perform all of the VSA operations required to implement the hierarchical VSA scheme presented in the SPIE21[11] paper using an alternative technology based on emerging energy efficient SNN neuromorphic processing devices such as the IBM TrueNorth[16] and Intel Loihi 2[17] processors. Eliasmith[5] has demonstrated how spike-rate encoded SNN circuits can implement an HRR based VSA scheme for cognitive processing in order to perform the required convolution and deconvolution operations. Whilst this is an elegant solution, it requires large numbers of spikes to be processed and since the energy efficiency of neuromorphic processors is typically a function of the number of spikes required[16] this is not an energy-efficient mechanism. Similarly, although BSC operators are much simpler to perform than the HRR equivalent, representing 10,000 bit HVs as spike sequences is also not energy efficient. An alternative to SNN spike rate encoding is time-to-spike encoding which can significantly reduce the number of spikes required. To use time-to-spike encoding requires a different type of HV representation which we term a sparse hypervector (SHV). An SHV encoding model was proposed by Laiho et.al.[18] , which is based on a slot encoding mechanism with and SHV encoded as M slots with B possible bit positions per slot but with only one bit set in each slot. A recent paper by Frady et.al.[19] , compared various alternative SHV models and concluded that this slot encoding has the all desired properties for VSA manipulations and outperformed the other methods evaluated.

This paper is structured as follows. In Section 2 the basic VSA operations are described and using the Laiho SHV model the equivalence between HV and SHV bundling capacity is presented. Section 3 describes how the Lahio model can be mapped to a time-to-spike SNN model and how all the basic VSA operations can be represented as SNN circuits. Section 4 summarises the MDO communications re-planning scenario presented in the SPIE21[11] paper. This scenario is used to illustrate how the basic SNN circuits can be combined to perform the more complex VSA functions required and compares the estimated energy requirements for SNN devices with those obtained for PCM devices performing similar functions. Conclusions and plans for future work are presented in Section 5.

2. VSA OPERATIONS

Unlike classical computing, which operates on bits through logical operations and the four arithmetic operations of addition, subtraction, multiplication and division, VSAs deal with HVs through three operations, *bundling* (a superposition operator denoted ‘$+$’), *binding* and *unbinding* (permutation/multiplication operators denoted ‘$\odot$’ and ‘ $\oslash$ ’ respectively) and a HV *normalisation* operator. VSA vectors built using these operations can then be compared for similarity and have their sub-vector contents probed using vector comparison operators such as *cosine similarity* and *Hamming Distance/Similarity*.

Binding permutes HVs to a different part of the HV space making them orthogonal to all other vectors with high probability. In this paper two types of binding are used. *roll-filler* and *cyclic-shift* binding. *roll-filler* binding is a multiplicative permutation between *roll* (R) and *filler* (F) HVs such that if $V = R \odot F$ then V is orthogonal to both R and F with high probability. Binding is also commutative and so $V \oslash R = F$ and $V \oslash F = R$. *Cyclic-shift* binding, denoted ‘ρ’ is a unary version for binding/unbinding an HV. *Cyclic shift* binding cyclically shifts the HV to the right right. *Cyclic shift* unbinding cyclically shifts the vector to the left. Again the *cyclic shift* binding produce an HV the is orthogonal to the original HV. Binding/unbinding and cyclic-shift are also associative and distribute over *bundling*. For many VSAs, including BSCs and Laiho’s SHVs, binding and unbinding are also invertible.

Bundling combines orthogonal HVs using a sperposition operation, such as majority sum addition, into a single same sized compound HV that bears similarity to each of its roll-filler constituents. The resulting HV therefore represents the group of sub-HVs of which it is composed, analogous to a set or data record, while simultaneously storing each sub-HV within its HV elements. Consider the bundled set of roll-filler pairs in Eq. 1

$$Z_v = V_r \odot A_v + W_r \odot B_v + X_r \odot C_v \tag{1}$$

To recover the sub-HVs from Z_v it must be unbound with the appropriate roll HV, for example to extract A_v perform:

$$V_r \oslash Z_v = A_v + V_r \oslash W_r \odot B_v + V_r \oslash X_r \odot C_v \tag{2}$$
$$= A_v + noise \tag{3}$$

By designating the roll HVs in Eq. 1 as '*position*' HVs (e.g., $V_r = P_0, W_r = P_1, X_r = P_2$) it is easy to see that Eq. 1 can be used to represent sequences where the filler vector at any position can be extracted by unbinding V_r with the position vector. An alternate approach for the creation of sequences is to combine bundling with the cyclic-shift binding operator, for example

$$Z_v = \rho^1(A_v) + \rho^2(B_v) + \rho^3(C_v) \tag{4}$$

The exponentiation operator applied to ρ indicates the number and direction of the cyclic-shift steps. Each bundled sub-HV can be retrieved in order by applying ρ in the opposite direction.

$$A_v \approx \rho^{-1}(Z_v) = A_v + \rho^1(B_v) + \rho^2(C_v) \tag{5}$$

Bundled VSA HVs preserve vector dimension regardless of the number of sub-attribute HVs being combined. However, as shown in Section 2.2 the ability to decode sub-HVs within a bundled vector is limited by the vector dimension (D), the number possible vectors (L), which we define as the vector library size, and the acceptable unbinding error rate (typically 1.0E-06) If greater bundling capacity is required then it becomes necessary to use a hierarchical bundling scheme. In[12, 13, 20] an hierarchical binding/bundling scheme is described that employs both permutation and cyclic-shift binding to enable the creation of practically unlimited hierarchically nested sequences overcoming the limitations that occur when using Eq. 1 or Eq. 4. The encoding scheme is given in Equation 6 and is capable of encoding many thousands of sub-feature vectors even when there are repetitions and similarities between sub-features:

$$Z_x = \sum_{i=1}^{cx} \left(\rho^i(Z_i) \odot \prod_{j=0}^{i-1} p_j \right) + StopVec \odot \prod_{j=0}^{cx} p_j \tag{6}$$

Omitting $StopVec$ for readability, this expands to,

$$Z_x = p_0 \odot Z_1^1 + p_0 \odot p_1 \odot Z_2^2 + p_0 \odot p_1 \odot p_2 \odot Z_3^3 + \ldots \tag{7}$$

Where

- Z_x is the next highest semantic vector containing a *superposition* of x sub-feature vectors;
- $\{Z_1, Z_2, Z_3, \ldots Z_n\}$ are the sub-feature vectors being combined for the individual nodes of Figure 1. Each Z_n itself can be a compound vector representing a sub-workflow or a complex vector description for an individual service step, built using the methods described by Simpkin et al;[21]
- $p_0, p_1, p_2, \ldots$ are a set of known *atomic* role vectors used to define the current position or step in the workflow.
- cx is the size of bundled vector Z_x, i.e., the number of sub-feature vectors being combined; and
- $StopVec$ is a role vector owned by each Z_x that enables it to detected when all of the steps in its (sub)workflow have been executed.

The key to understanding the power of Equation 6 as a hierarchical binding scheme results from the fact that the cyclic_shift operator distributes over the $\odot$ operator and we stipulate that cyclic_shift takes precedence over $\odot$. As illustrated in Equation 7, this means that if a compound vector such as Z_x is used as a sub-feature vector for a higher level concept then the P vectors associated with each sub-feature vector Z_n are automatically promoted to a new value by the cyclic_shift. This is illustrated in Figure 1 where we have replaced the $\odot$ operator with a 'dot' symbol and the ρ operator by adding an exponent to the vector in order to de-clutter the diagram. These representations are interchangeable throughout the rest of this paper.

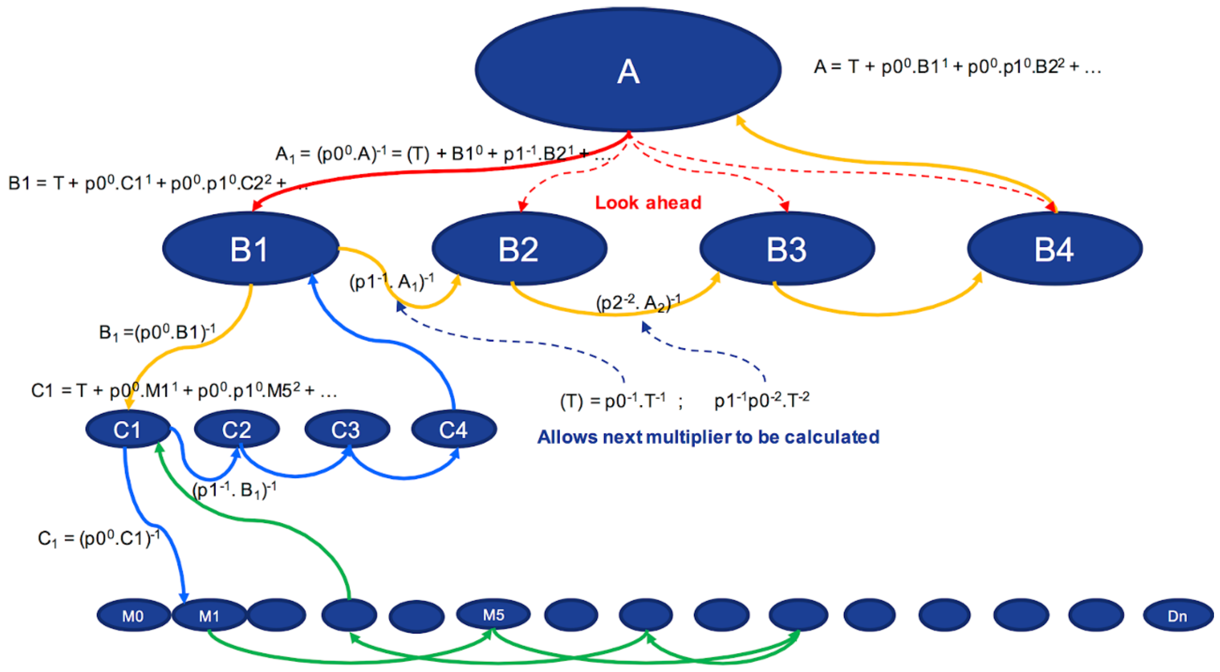

A
A = T + p0⁰.B1¹ + p0⁰.p1⁰.B2² + …
A₁ = (p0⁰.A)⁻¹ = (T) + B1⁰ + p1⁻¹.B2¹ + …
B1 = T + p0⁰.C1¹ + p0⁰.p1⁰.C2² + …
Look ahead
B1
B2
B3
B4
(p1⁻¹. A₁)⁻¹
(p2⁻². A₂)⁻¹
B₁ =(p0⁰.B1)⁻¹
C1 = T + p0⁰.M1¹ + p0⁰.p1⁰.M5² + …
(T) = p0⁻¹.T⁻¹ ; p1⁻¹p0⁻².T⁻²
Allows next multiplier to be calculated
C1
C2
C3
C4
(p1⁻¹. B₁)⁻¹
C₁ = (p0⁰.C1)⁻¹
M0
M1
M5
Dn

Figure 1. Schematic representation of the hierarchical recursive binding scheme showing how vectors compose into higher level vectors which can then be recursively un-bound from the higher level to lower level vectors.

As shown in Section 4, Equation 6 can be used to construct service description vectors and in turn, using the same recursive formulation, these vectors can be combined into service workflow vectors at the next higher semantic level. In this case the vectors, $Z_1 \ldots Z_{cx}$ represent the individual services and service workflow compositions and the vectors $p_o \ldots p_{cx-1}$ are a set of random vectors chosen to represent the position of the component services in the workflow. In this paradigm workflow HVs and SHVs are vectors of vectors.

To extract the sequence of services from a workflow HV the vector can be recursively unbound using the $\oslash$ operator and performing a left cyclic shift on the resultant vector as follows:

$$\rho^{-1}(\,Z_x \oslash p_0)\;=Z_1+\rho^{-1}(p_1)\oslash\rho^{1}(Z_2)+\ldots=Z_1+noise \tag{8}$$

In this case the resulting vector is a noisy version of the vector Z_1 and to identify that this vector is the requested vector Z_1, it must be compared with a list of possible vectors and verified to be Z_1 by virtue of having the highest similarity (e.g., lowest *Hdist*). This process is often referred to as a 'clean-up' memory operation.

2.1 BSC vs SHV operators

In the Kanerva BSC HV representation the left or right cyclic shift operations are performed by a rotation of all D bits of the HV and both the *role/filler* binding and unbinding operations are performed using the logical XOR operation between all of the bits of the respective role and filler vectors. The superposition bundling operation in this case is a majority sum operation. In the Laiho SHV slot encoding model the left and right cyclic-shift operations are achieved by a similar right-cyclic-shift of the M slot positions. *Role-filler* binding is however achieved by adding the bit positions between the corresponding slots of the role SHV (e.g., bit position b_1) and filler SHV (e.g., bit position b_2) and then computing the sum

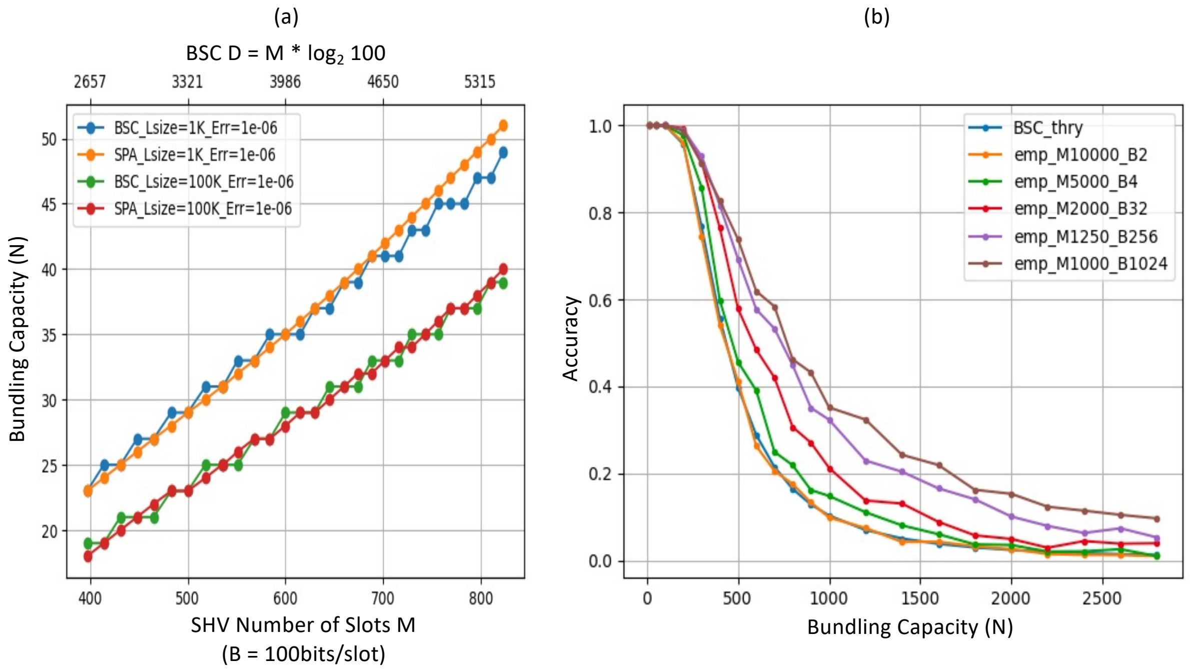


Figure 2. Bundling Capacity of BSC HVs and SHVs. (a) shows the equivalence between the two encoding approaches for low unbinding error rates. (b) shows how the unbinding accuracy is degraded for higher bundling capacity using 10,000 bit BSC HVs and SHVs with the same information capacity

of the two bit positions modulus the number of possible bit positions B (i.e., $b_n = (b_1 + b_2) mod\, B$). The bit at this position (b_n) is then set in the slot of the bound SHV. Unbinding is a modulus subtraction operation (i.e., $b_1 = (b_n - b_2) mod\, B$). The SHV bundling operation is performed by slotwise addition operation by counting the number of bits that occur in the same position within each slot from all of the SHVs that are bundled. Sparsity is preserved by computing for each slot the bit position corresponding to the maximum number of co-occurring bits (i.e., the argmax) and setting the bit at this position in the final bundled SHV. If there are multiple bit positions with the same number of co-occurring bits then the bit at one of these positions is selected either randomly or by a well defined algorithm (e.g., lowest bit index). The resulting bundled SHV therefore has the same sparse structure as the component SHVs of which it is composed.

2.2 BSC vs SHV Bundling Capacity

The bundling capacity of BSC HVs is well established both theoretically and empirically and depends upon the vector dimension (D), the number of symbolic vectors in the clean-up memory library (L) and the error rate that can be accepted when unbinding. When using HVs to represent service workflows a high degree of unbinding accuracy is required (i.e., typical error rates of $P\_error <= 10^{-6}$). While this limits the bundling capacity of any single HV this can be compensated for by using the hierarchical binding scheme described above.

In the SPIE21[11] paper it was demonstrated that HVs can be used to perform IoBT service workflow orchestration in typical MDO TacCIS environments. In these environments D is limited by the capability to efficiently exchange HVs over relatively low bandwidth communications networks. It was demonstrated that this can be performed efficiently using 10,000 bit BSC HVs. The vector library size in these type of applications depends on the number of IoBT services in the network with each service comparing its own service description vector(s) with the unbound workflow vector to determine if they are candidates to perform the next action required[13,14,20,21] . This vector comparison is performed in parallel by all the IoBT services and so the clean-up memory library is essentially distributed across the communications network. This requires the capability to handle large clean-up memory library sizes, typically in the range 1,000 to 100,000 vectors. As a result the typical bundling capacity is limited to 90 sub-HVs in a single HV. This has been confirmed both theoretically and empirically.

The bundling capacity for the Laiho slot encoding model[18] can also be determined both theoretically and empirically and Figure 2 (a) compares the bundling capacity of BSC HVs of dimension (D) with SHV encoding for the same information capacity (i.e., $D = M Log_2(B)$) for library sizes (L) of 1,000 and 100,000 SHVs respectively and an unbinding error rate of $P\_error <= 10^{-6}$. In the example the SHV bits per slot was fixed at $B = 100$ while the number of slots M was varied. The results show that for the same information capacity the bundling capacity of the two representations are equivalent over a wide range of values of M with a 20% reduction in capacity for a 100 times increase in library size. This result is significant since it shows that the cost of transmitting SHVs and HVs over a communications network is the same for this level of unbinding accuracy.

In the VSA literature, bundling capacity is often quoted in the range of hundreds or even thousands of HVs. Reference[22] for example, presents results for a range of VSA vector binding and bundling schemes. However in their results the vector library size is limited to a small number of symbol vectors (L=27) and large bundling capacity is only achieved at the expense of lower unbinding accuracy. Figure 2 (b) presents empirical results for SHV bundling capacity using the Lahio model and a library size L = 10,000. The figure shows the unbinding accuracy as a function of bundling capacity for larger bundling capacity and for SHV vectors with the same information capacity, i.e., M_log_2B. This is compared to the theoretical BSC capacity (BSC_)thry). The empirical results show that the slot encoded SHVs perform better than the BSC HVs and that the accuracy improves as the number of slots (M) is reduced with a corresponding increase bit positions (B) to maintain the same information capacity. A comparison of the bundling accuracy of the Lahio model with results obtained in Reference[22] and for the same library size (L=27) shows that the slot encoding with 1 bit per slot also has similar bundling capacity to other non binary encoding schemes such as HRR and FHRR.

It should be noted that whilst bundling capacity is important, VSA bundling is not a data compression technique and judging performance based purely on sequence indexing capacity is not the main criteria for judging the value of any VSA approach. The main criteria should be the simplicity with which the vector binding, bundling and unbinding can be performed, and crucially the efficiency with which the required clean-up memory operations can be performed. For example,[23] found the HRR solution to be intractable; reporting issues with computation time and storage capacity and noting the computational complexity of HRR solutions to be $O(D.log(D))$ compared to $O(D)$ for BSC.

The following sections show how all the required VSA operations operations can be efficiently performed using the sparse vector slot encoding model using 1 bit per slot and how these operations can be implemented in time-to-spike SNN circuits.

3. MAPPING SPARSE BINARY VSA OPERATORS ONTO SNN CIRCUITS

Mapping of the sparse slot encoding vector model into an SNN representation is performed by treating each slot as a neuron and the time-to-spike as the bit position. In the following sections a symbolic representation of an SNN circuit is used as illustrated in Figure 3(a). In the circuit representation neurons are shown as triangles, each with an input dendrite and an output axon. The axon of any neuron can connect to the dendrite of one or more other neurons via synaptic connections. This representation reflects the structure of a typical neuromorphic processor, for example the IBM experimental Truenorth architecture[16] . The major difference between this architecture and the Truenorth architecture is that each synaptic connection between an axon and a dendrite can introduce a time delay to the incoming spike before it stimulates the neuron associated with the dendrite. The way in which the neuron responds to the stimulus depends on the neuron model used. The sections below describe how SNN circuits of this type can perform all of the required VSA operations using relatively simple neuron models and a time-to-spike encoding strategy.

All of the circuit components and models described in this paper have been simulated using the Brian 2 SNN simulation tool[24] . The Brian2 simulator seeks to simulate the behaviour of biological neurons but is equally applicable for simulating neuromorphic processing devices. In this paper Brian 2 model terminology is used throughout to describe the neuron model behaviour. All the neuron models are therefore defined by systems of differential equations that determine how the internal neuron voltage changes. This is controlled by user defined parameters. Each neuron model can specify one or more spike

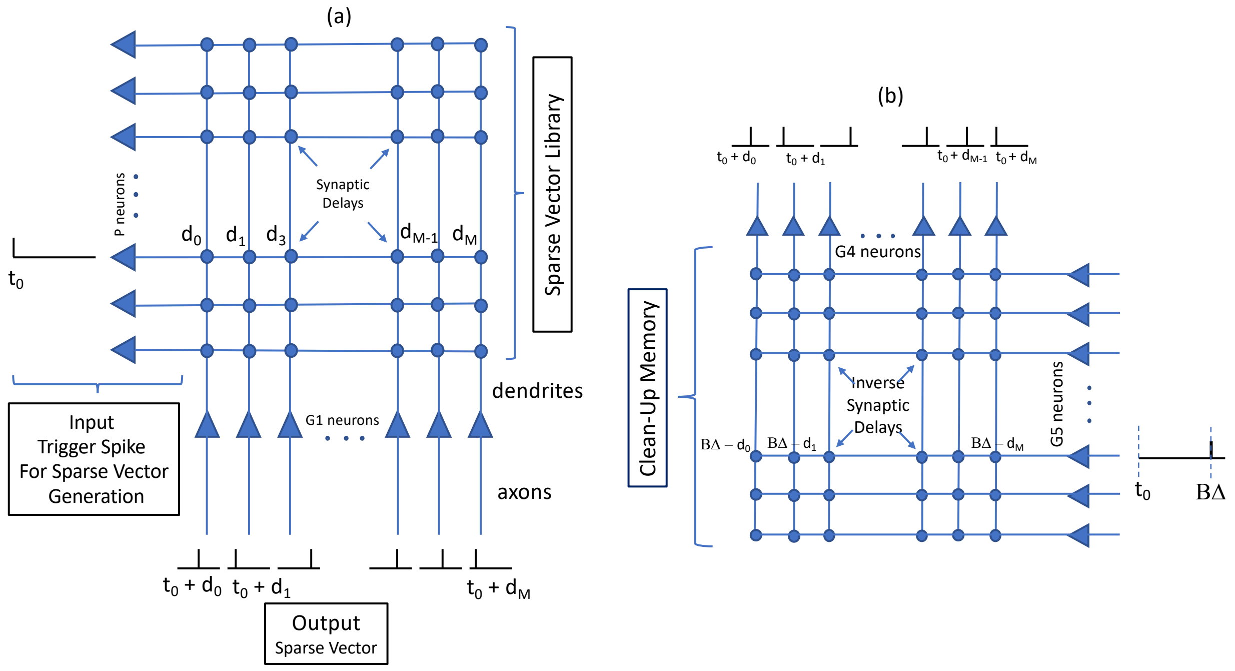

(a)
Sparse Vector Library
P neurons
Synaptic Delays
d_0 d_1 d_3 d_{M-1} d_M
t_0
Input Trigger Spike For Sparse Vector Generation
G1 neurons
dendrites
axons
$t_0 + d_0$ $t_0 + d_1$ $t_0 + d_M$
Output Sparse Vector
(b)
$t_0 + d_0$ $t_0 + d_1$ $t_0 + d_{M-1}$ $t_0 + d_M$
G4 neurons
Clean-Up Memory
Inverse Synaptic Delays
G5 neurons
$B\Delta - d_0$ $B\Delta - d_1$ $B\Delta - d_M$
t_0 $B\Delta$

Figure 3. Example SNN VSA circuits. (a) SHV Generation and (b) Clean-Up Memory

threshold voltage levels with the result that the neuron producing one or more output spikes on its axon whenever a spike threshold voltage is exceeded. The Brian 2 neurons also allow the user to specify the neuron voltage reset values following a spike and importantly to specify refractory properties such that after the neuron fires it is prevented from further spiking during the refractory period.

3.1 VSA SNN Components

A typical VSA comprises five main components: Hypervector generation; Binding; Bundling; Unbinding and Clean-Up memory. The following sections show how each of these components can be mapped into SNN circuits which can then be combined to perform more complex VSA operations. In Section 4.4, an example complex VSA SNN circuit that can perform hierarchical vector unbinding is described.

3.2 Sparse Hypervector Generation

SHV generation is the component that provides a mechanism for selecting the SHV's that are to be processed by the other components. SHVs with M slots can be generated such that the bit position in each slot is random or where the bit position is chosen to represent some specific vector properties e.g., a semantic vector. Each generated SHV can be stored into a neuromorphic SHV library using synaptic time delays (d_i) corresponding to the bit position. The circuit is shown in Figure 3(a). The circuit comprises two neuron groups designated P and G1. Each of the G1 neurons corresponds to one slot in the Laiho model and so there are $M * G1$ neurons. There is one P neuron for each SHV stored in the SHV library and so there are L P-neurons. The P-neuron axons are connected in a matrix structure to the G1 dendrites via synapses as shown in Figure 3(a) and so there are $M * L$ synaptic connections. Each P neuron axon connects to each of the G1 neuron dendrites through the corresponding synaptic connections. The firing of a single P-neuron at time t_0 is the trigger for the generation of a single SHV. The corresponding spike stimulates all of the synapses on the P-neuron axon and after a time delay of d_j

the associated j^{th} G1 neuron is stimulated and its neuron voltage is increased by V_s at time $t_0 + d_j$. All the G1 neurons in this case have a simple leaky integrate and fire (LIF) type neuron model and a threshold voltage of V_t ($V_t \leq V_s$). The corresponding G1 neuron immediately fires and generates an output spike. The resulting spike firing pattern of all the M G1 neurons represents the required SHV time-to-spike encoding.

3.3 Clean-Up Memory Circuit

Before considering how sequences of SHVs can be generated and subsequently bound and bundled, it instructive to understand how the clean-up memory component operates in relation to the SHV generation circuit.

A typical clean-up memory circuit is illustrated in Figure 3.3(b). In this example the circuit is required to mirror the action of the SHV generation circuit such that if the SHV generated by a spike from the j^{th} P-neuron (P_j) at time t_0 were input then the corresponding j^{th} G5 output ($G5_j$) neuron would be stimulated and would fire at time $t_0 + B\Delta$. To achieve this objective the circuit comprises M input neurons in a neuron group designated as G4 and L output neurons designated as neuron group G5. In this case the axon of each G4 neuron connects to all L dendrites of the G5 neurons via synaptic connections. The synaptic delays are calculated such that any SHV input to the circuit is additionally delayed at each of the synapses on the G5 dendrites i.e., on the $G5_j$ neuron the additional time delay would be $B\Delta - d_j$. Thus at time $t_0 + B\Delta$ all of the delayed spikes will simultaneously stimulate the dendrite of the $G4_j$ neuron resulting in a stimulus of $M * V_s$ volts being received. If this voltage exceeds the neuron threshold voltage then the neuron will fire at this time. Conversely on all other dendrites the stimulus will arrive with a random time delay and so the voltage on these neurons will not exceed the spike threshold.

When a number of generated SHVs have been bound and bundled, following an unbinding operation the received SHV will be a noisy version of one of the SHVs in the sense that only a subset of the M bits will match. The expected number of matching bits (m) depends on the values of M, B and the number of bundled SHVs N. This can be determined theoretically and has been empirically verified. In this case only m synaptic time delays will match and the the resulting stimulus at time $t_0 + B\Delta$ will be $m * V_s$. Similarly the expected mean number of matching bits for all other $L - 1$ neurons that will align randomly can be determined. The spike threshold V_t of the G5 neurons is therefore chosen to ensure that only those neurons that match will fire with high probability. An important feature of this clean-up memory circuit is that matching against all L library SHVs is performed in parallel and is only limited by the number of synaptic connections on the dendrite. The results from an example Brian2 simulation of a clean-up memory circuit is given in Section **??**.

3.4 Cyclic Shift Binding and Bundling

Right Cyclic Shift binding of the spike encoded SHV can be performed by connecting the output axon of each of the M neurons to the dendrite of the next neighbouring neuron in the group via further set of synaptic connections. Figure 4 (a) illustrates this for a neuron group designated G2 where the output of neuron $G2_j$ is wired to the input of neuron $G2_{j+1}$ and the output of the last neuron, $G2_{M-1}$, is wired to the input of the first neuron, $G2_0$. The synaptic delay on these connections is $B\Delta - \delta$ seconds where δ is the minimum neuron processing delay between a stimulus event on the dendrite and the output spike being generated. The circuit operates as follows. At time t_0 a P-neuron fires to select the first of the SHVs to be bundled. This SHV is then cyclically shifted and arrives at the input of the G2 neuron group with all spikes delayed by $B\Delta$ seconds. At time $t_0 + B\Delta$ a second P-neuron is fired which selects the next SHV to be bundled. The input to each G2 neuron is then the shifted spike from the first SHV and the spike from the current un-shifted SHV. To perform the bundling operation on these two SHVs each neuron selects fires on receipt of the earliest of these spikes and then uses the refractory property of the neuron to suppress any later spikes occurring in the remainder of the $B\Delta$ time period. This operation maintains the sparsity of the resulting output SHV at one spike per neuron. The process continues until all N SHVs that are to be cyclically shifted and bundled into a new SHV with only one spike per neuron.

Figure 4 (b to d) shows the results from the Brian2 simulation of this process. The number of SHVs being bundled in this example is limited to just five SHVs with M= 100 neurons, B = 100 bit positions, $\Delta = 0.001$ and $\delta = 0.0001$ so that the actions can be clearly seen. In Figure 4(b) the sequence of P-neuron firing is shown beginning with neuron P_4 down to neuron P_0 with a 100ms separation between spikes. In Figure 4 (c) the neuron voltage from a single G2 neuron ($G2_j$) is shown. In the first 100ms there the neuron is stimulated at $t_0 + 80ms$ and since this is the first stimulus the neuron fires. This is indicated by the fact that there is no exponential decay of the neuron voltage since on firing the neuron voltage is reset to zero. The pattern of firing of all 100 G2 neurons is shown in Figure 4 (d) in the first 100ms, the spike pattern is for the un-shifted SHV that was generated by the triggering of the P_4 neuron. In the next 100ms the spike from the adjacent

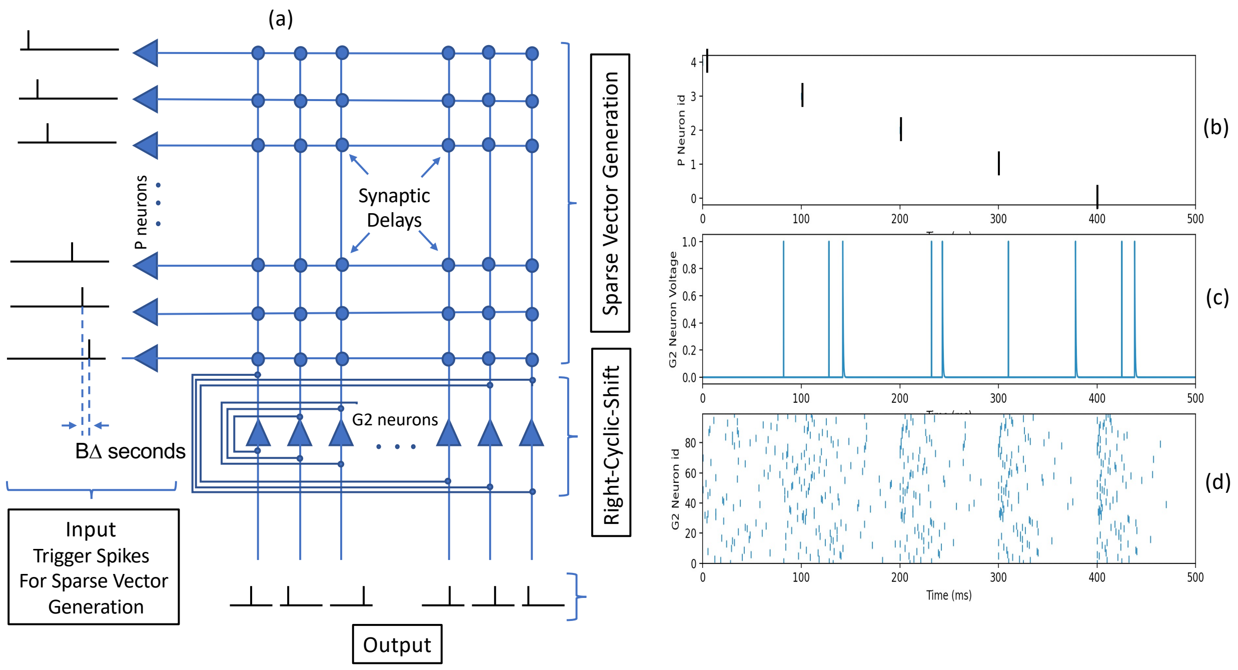


Figure 4. Cyclic Shift Binding. (a) SNN Circuit. (b)-(d) Brian2 simulation results binding 5 vectors with M = 100, B = 100 and $\Delta = 1ms$

neuron $G2_{j-1}$ and the new spike generated from the firing of the P_3 neuron stimulate the neuron as shown in Figure 4 (c). The first of these stimulus events causes the neuron to fire but the second stimulus event does not result in a spike. The output from all the G2 neurons now represents the cyclically shifted bundled SHV of the first two selected SHV's. The process continues until all five SHVs have been bundled, and the resultant cyclically shifted bundled SHV is as shown in the time period between 400ms and 500ms in Figure 4 (d).

3.5 Cyclic Shift Unbinding and Clean-Up

To unbind this SHV it is necessary to perform a left-cyclic shift operation. The circuit required to do this is shown in Figure 5(a) where the neuron group G2 that performs the cyclic shift is wired in the reverse order to the previous circuit. In this case the output of neuron $G2_j$ is wired to the input of neuron $G2_{j-1}$ with the output of neuron G_0 wired to the input of neuron G_{M-1}. The time delay on the cyclic shift connections is again $B\Delta - \delta$ seconds. The bundled SHV is injected into the circuit at time t_0 and therefore the unbinding of each bundled SHV occurs every $B\Delta$ seconds. This is shown in the Brian2 simulation results in Figure 5(b) where the SHV in each 100ms time period is a left-shifted version of the SHV from the previous 100ms.

The output SHV from the G2 neurons is a noisy version of each of the original bundled SHVs in the sense that only a subset of the bit positions will match the corresponding 'clean' SHV. This noisy SHV is injected into the clean-up memory circuit as described in Section 3.3 and the resulting neuron voltages from all of the G3 output neurons is shown in Figure 5(c). The effect of the clean-up memory operation can be clearly seen with the neuron voltage of all neurons other than the neuron corresponding to the noisy unbound SHV being random but for the matching neuron the voltage peaks at the end of each unbinding cycle. In the example, and for illustration only, the spike stimulus event on the neurons dendrite is set at 1.0 volts and so the voltage is a measure of the number of matching bits. In this example the average number of matching bits is 19 which conforms to the theoretical expected value. Each neuron has a threshold voltage that is determined by the

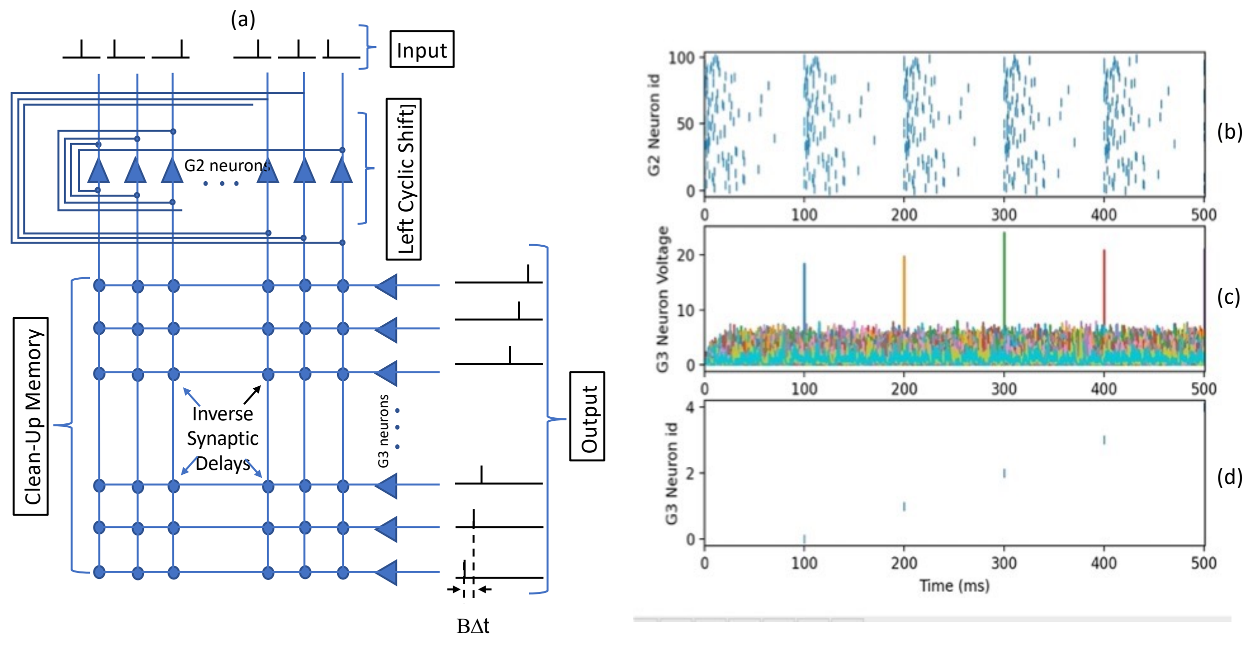


Figure 5. Cyclic Shift Unbinding and Clean-Up Memory. (a) SNN Circuit. (b)-(d) Brian2 simulation results unbinding 5 vectors with M = 100, B = 100 and $\Delta = 1ms$

maximum number of SHVs that can be bundled and the neuron spikes when this threshold is exceeded. In this case the pattern of neuron firing is shown in Figure 5(d) which shows a sequence ordering from $G2_0$ to $G2_4$, which is the reverse of the order in which the SHVs were bundled as expected. The neuron firing order is essentially an index of the matching SHV and if the actual matching SHV is required then this output can be used as the input triggers to the SHV generation circuit.

3.6 Role-Filler Binding and Bundling Circuit

Role-filler binding that conforms to the Laiho slot encoding model requires a neuron model that can perform a modulus addition between the times of arrival of two input spikes and produce a single output spike at that position. A neuron model that can perform this operation is illustrate in Figure 6(a) where the synaptic connections between the two input neurons and the output neuron introduce no delay. The neuron model uses a linearly increasing neuron voltage that is then held constant on the arrival of the first spike and then linearly decays on the arrival of the second spike. The neuron has two spike thresholds voltages V_0 and V_t and the time constant of the linear increase or decrease is $(V_t - V_0)/B\Delta$. If the neuron voltage falls below the lower threshold V_0 the neuron fires and the neuron voltage reverts to the linear increase. If the upper threshold is exceeded the neuron fires and the neuron voltage is reset to V_0. If two spike times (e.g., t_1, t_2) sum to less than $B\Delta$ the lower threshold is crossed at time $t_1 + t_2$ and the neuron fires and the neuron voltage reverts to a linear increase which reaches the upper threshold at time $B\Delta + t_1 + t_2$. This is equivalent to the required modulus addition in the next cycle. If the two spike times (e.g., t_3, t_4) sum to more than $B\Delta$ then the lower threshold is reached at time $B\Delta + t_3 + t_4$ which is the required modulus addition in the next cycle time window and the neuron fires. The neuron voltage then reverts to a linear increase but this gets reset at time $2B\Delta$ and the process can repeat.The neuron model also operates on a cyclical basis and resets every $2B\Delta$ seconds.

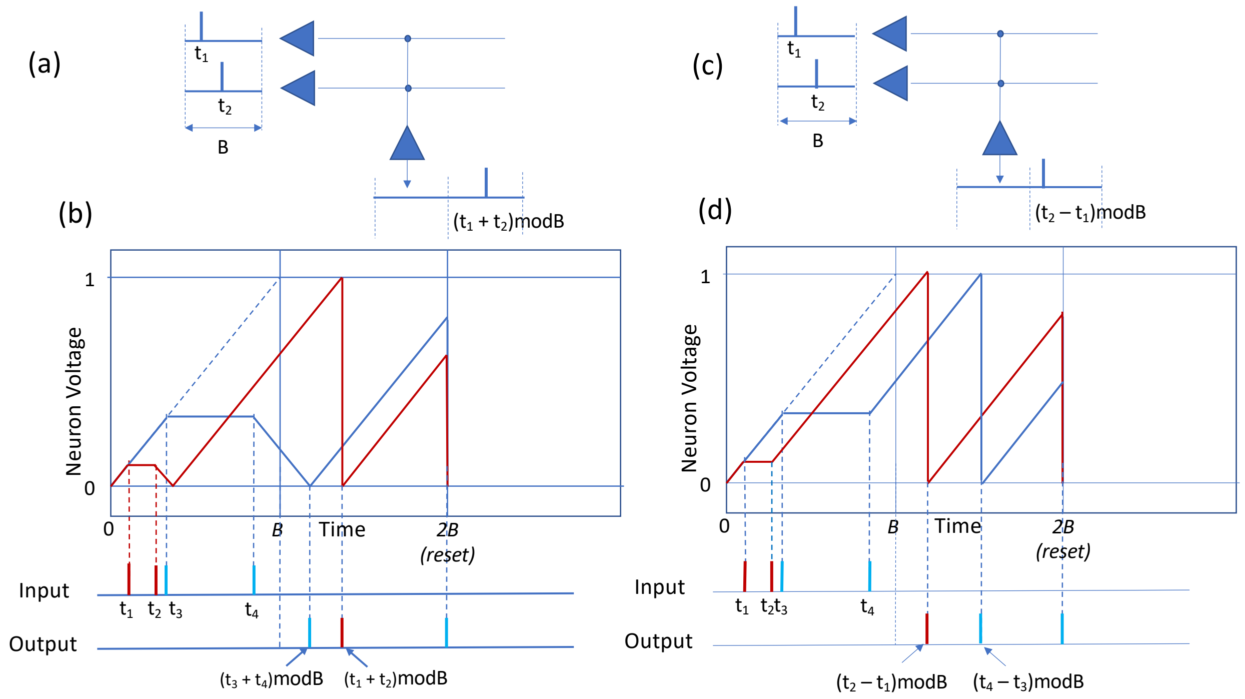


Figure 6. Role-Filler Binding and Unbinding Neuron Model. (a) Binding using modulus addition. (b) Unbinding using modulus subtraction

The Role-filler binding and bundling circuit that performs this operation is shown in Figure 7(a). The circuit uses the the same SHV generation circuit as the cyclic shift binding and bundling circuit, but now the role and filler SHVs are generated concurrently with each pair being generated after a time delay of $2B\Delta$ seconds. To aid understanding of how this circuit operates Figure 7 (b) to (f) shows the corresponding Brian2 simulation results. The circuit comprises two neuron groups. Group G1 performs the binding using the neuron addition model and Group G2 performs the bundling using the same model as the cyclic shift neurons to select the earliest spike. The G2 neuron group could have been implemented as a recurrent connection where each G2 neuron feeds back to itself. However, a simpler neuron model was used in which neuron fires in its current cycle at the same time as in the previous cycle the unless an earlier stimulus is received from the next vector to be bundled.

In Figure 7(b) the P-neurons are shown to fire every $2 * B\Delta = 200ms$ with five pairs of SHVs being generated. Each pair are the role and filler SHVs that are to be bound in each case. Figure 7(c) shows the neuron voltage of one of the G1 neurons which illustrates the addition bundling operation. Whilst the Laiho model specifies a modulus operation, all that is actually required is a reversible operation for binding and unbinding. A simplification to the model was therefore used which performs the addition but does not apply the extra cycle required for the modulus. As will be seen the reverse subtraction operation then also does not perform the modulus cycle. The output spikes from the single corresponding neuron are shown in Figure 7(d) and these are fed into the bundling neuron group G2 with Figure 7(e) showing the neuron voltage of the corresponding G2 neuron. The spike pattern for all 100 G2 neurons after each binding and bundling cycle is shown in Figure 7(f).

3.7 Role-Filler Unbinding and Clean-Up Memory Circuit

To illustrate how the Role-Filler unbinding operates the time delays of the spikes from the bundled SHV are used to create an additional entry in the SHV Generation memory that stores the bundled vector. The unbinding and clean-up memory

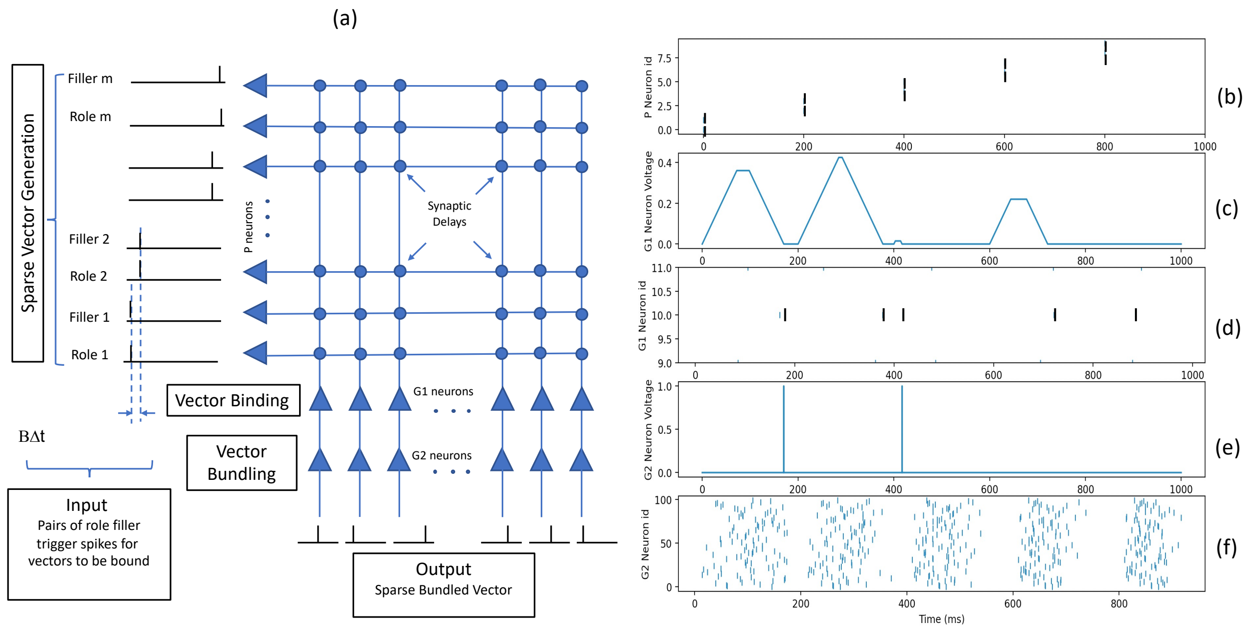


Figure 7. Role-Filler Binding and Bundling. (a) SNN Circuit. (b)-(f) Brian2 simulation results unbinding 5 vectors with M = 100, B = 100 and $\Delta = 1ms$

circuit is shown in Figure 8(a). The unbinding circuit comprises a singe neuron group (G4) which implements the subtraction neuron model shown in Figure 7(b). Subtraction is performed in a similar way to the addition but on receipt of the second spike the neuron voltage is linearly increased and a spike is generated at when this voltage exceeds an upper spike threshold.

The unbind operation uses the P-neurons to simultaneously generate a role SHV and the stored bundled SHV. This action is required to unbinfd the corresponding filler SHV. To unbind all the filler SHVs this process is repeated every $B\Delta$ ms cycling through the role vectors. This is illustrated in Figure 8(b) where roles P_0, P_2...P_8 are sequentially selected to unbind the bundled SHV. The corresponding voltage of one of the unbinding G4 neurons is shown in Figure 8(c) and the resulting unbound SHV in each 200ms unbinding cycle is shown in Figure 8(d). These SHVs are injected into the Clean-Up Memory and the resulting G5 neuron voltages are shown in Figure 8(e). Because the cycle time is now 200ms the noise varies as shown but as for the cyclic shift clean-up operation the neuron voltage on the neuron corresponding to the matching SHV adds at the end of each cycle to reflect the number of matching bits which again has an average of 19 bits. The sequence of G5 neuron firing is shown in Figure 8(f) which are the required corresponding filler SHV indices.

4. USING SHVS AND SNN CIRCUITS IN AN MDO SCENARIO

To demonstrate how a VSA approach can be used to perform typical MDO tasks the SPIE21[11] paper described a communications re-planning task being performed in a typical MDO TacCIS environment where agile rapid communications re-planning is required. The demonstration showed how a radios and communications plans can be represented as VSA HVs and how by exchanging the HVs over a number of radio networks, the HVs can be used to discover radio's operating under a current communications plan and instruct them to switch to a required target communications plan. The SPIE21[11] paper also explained how an experimental 'In Memory' processing 'Phase Change Memory' (PCM) device could be used to perform some of the required vector processing, specifically the clean-up memory processing, at ultra low energy.

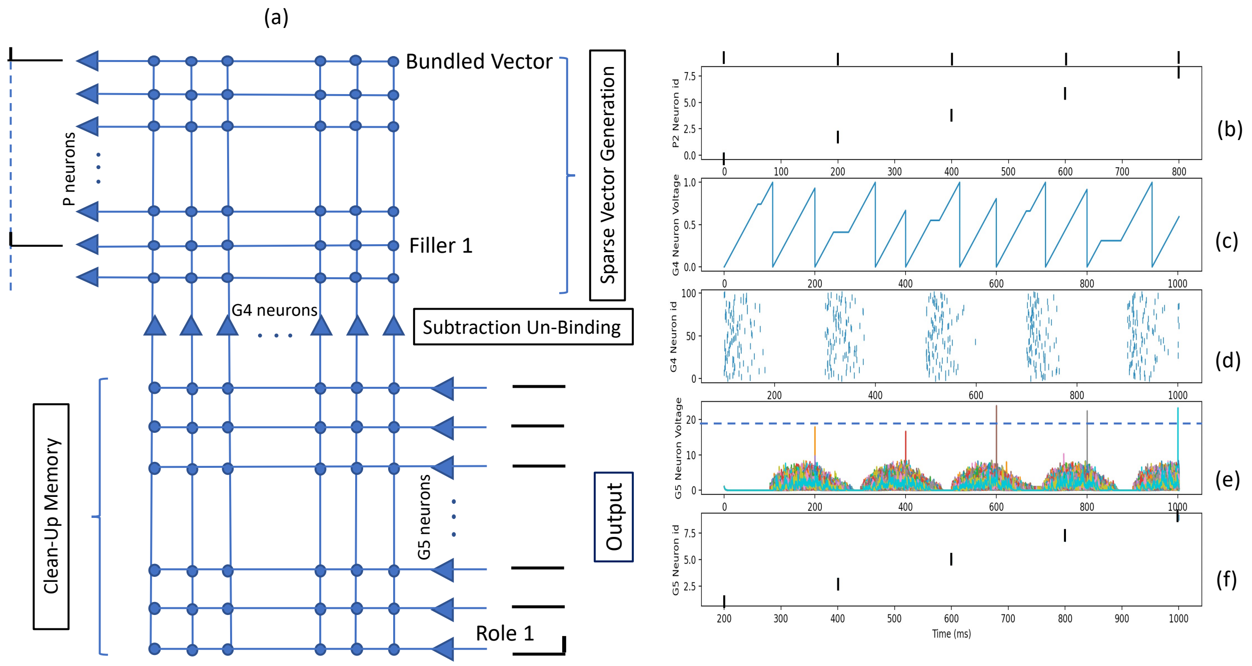


Figure 8. Role-Filler Unbinding and Clean-Up Memory.(a) SNN Circuit. (b)-(f) Brian2 simulation results unbinding 5 vectors with M = 100, B = 100 and $\Delta = 1ms$

This section describes how an SNN neuromorphic processor can be used to perform the required VSA operations and specifically how an SNN circuit can perform the clean-up memory operation on hierarchical SHVs.

4.1 The Scenario

The scenario used in the SPIE21[11] paper is shown in Figure 9. At the start of the scenario different types of platform and their associated radio's are operating on the different radio channels indicated by the three planes (CH1 - CH3) in the diagram. Some of these radio's are assumed to be on a fixed channel, some can operate on any two channels and a minority can operate on all three channels. Whilst three channels were used to demonstrate the principle, increasing the number of available channels and expanding the capabilities of the radios to use multiple channels is not a limitation. A commander wishes to request assets under their control to be used in different configurations operating on different radio channels. Figure 9, for illustration, shows the initial and target configurations of the radios and their associated platform (e.g., personal radio's are assumed to be associated with an individual user).

4.2 Representing Radio's as VSA SHVs

To represent radio's as VSA SHVs, the SHV binding scheme described by Equations 6 and 7 in Section 2is used. In the scenario, an individual radio is specified by key-value pairs represented in a json description file as illustrated in Figure 10. The json description defines the various attributes of the radio and its associated roles and platform characteristics. The keys (i.e., roles) are associated with corresponding values (i.e., fillers). This json structure is used for the purpose of our scenario but any field descriptions and corresponding values can be used. The json file is automatically parsed and converted into SHVs using a similar approach to that described in Reference [13] for BSC HVs. The approach uses random SHVs to describe each letter of the alphabet and any required additional symbols (e.g., punctuation, from which the various

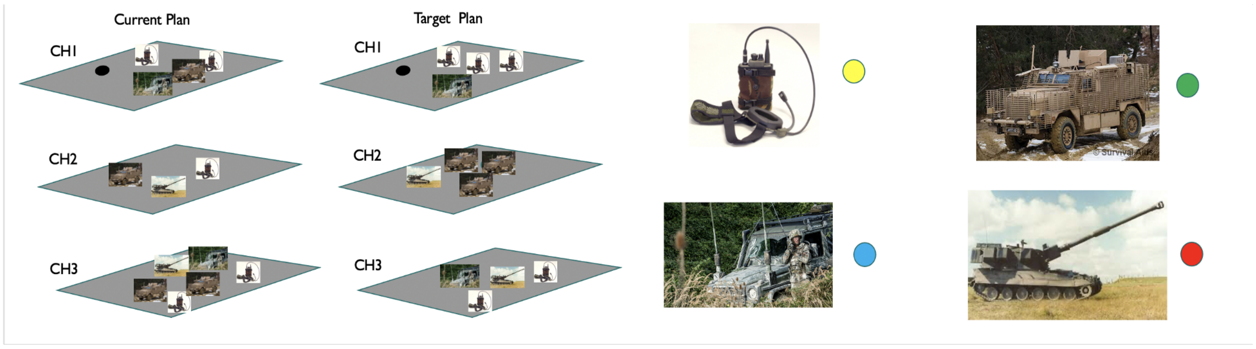


Figure 9. Schematic of the scenario showing the current and target communication plans in three channel layers. The colour coding of the different radio/platforms is used to illustrate the vector unbinding

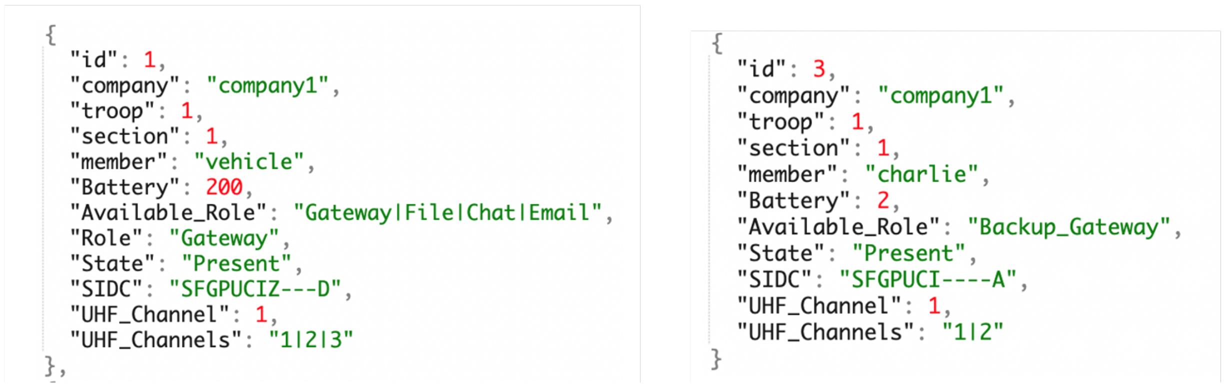
keys can be constructed. For example, the key ‘company’ is constructed from random SHVs for each letter of the word as follows:

$$Z_{key-company} = c^0 \odot o^1 \odot m^2 \odot \ldots \odot y^6 \tag{9}$$

In this case the SHV binding produces a unique role SHV for each key.

The ’value’ SHVs are constructed in a similar way but to allow for similar values to be compared the value SHVs use both binding and bundling operations. For example the value ’alpha’ is constructed as follows:

$$Z_{value-alpha} = a^0 \ + l^1 + p^2 \odot + \ldots + a^4 \tag{10}$$



```
{
  "id": 1,
  "company": "company1",
  "troop": 1,
  "section": 1,
  "member": "vehicle",
  "Battery": 200,
  "Available_Role": "Gateway|File|Chat|Email",
  "Role": "Gateway",
  "State": "Present",
  "SIDC": "SFGPUCIZ---D",
  "UHF_Channel": 1,
  "UHF_Channels": "1|2|3"
},
```

```
{
  "id": 3,
  "company": "company1",
  "troop": 1,
  "section": 1,
  "member": "charlie",
  "Battery": 2,
  "Available_Role": "Backup_Gateway",
  "State": "Present",
  "SIDC": "SFGPUCI----A",
  "UHF_Channel": 1,
  "UHF_Channels": "1|2"
}
```

Figure 10. json representation of two radios each with different configurations

VSA SHVs for both role and filler SHVs that can also be generated from a semantic vector space, for example Word2Vec,[25] via the method of randomised binary projection.[26] This enables semantically similar descriptions of radio’s to be created allowing the discovery of similar services from different MDO partners.

The resulting key and value SHVs are bound together to construct key-value (i.e., role-filler) SHVs and these SHVs are bundled to create a description SHV for each radio as follows:

$$Z_{radio-1} = Z_{key-id} \odot Z_{value-1} + Z_{key-company} \odot Z_{value-company1} + \ldots + Z_{key-SIDC} \odot Z_{value-SFGPUCI---A} \tag{11}$$

In our scenario, since we want to instruct a radio to switch to a specific channel, we separately construct SHVs for each of the allowable channels on which the associated radio can operate. For example:

$$Z_{UHFChannel1} = Z_{key-UHFChannel} \odot Z_{value-1} \tag{12}$$

and

$$Z_{VHFChannel1} = Z_{key-VHFChannel} \odot Z_{value-1} \tag{13}$$

And so the first radio in our example would also have SHVs for VHF Channel 1 and UHF channels 1 - 3 together with bundled with an SHV that described its key-value (i.e., role-filler) description pairs.

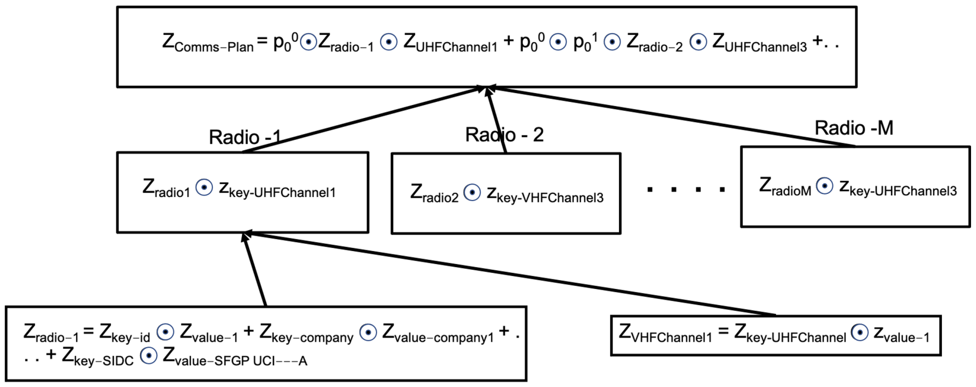
This approach provides the flexibility for a radio to have a number of accredited functions and radio channels on which it can operate with each allowable configuration being stored on the radio device as a separate SHV.

4.2.1 Representing Communications Plans as VSA SHVs

To represent a communications plan (comms-plan) as a VSA SHV each radio can be represented as a service and the comms-plan itself as a workflow. The workflow needs to record the order in which the radio services are discovered and so the hierarchical binding scheme described in Section 2 is used to achieve this. In this case the service SHVs are constructed as an SHV that combines the required radio description SHV and the channel SHV that the plan requires that radio to be on, for example:

$$Z_{Comms-Plan} = p_0^0 \odot Z_{radio-1} \odot Z_{UHFChannel1} + p_0^0 \odot p_1^0 \odot Z_{radio-1} \odot Z_{UHFChannel1} + \dots \tag{14}$$

The construction of a comms-plan SHV is shown schematically in Figure 11 where it can be see that the comms-plan SHV is a hierarchical *vector of vectors* which, as a result of the binding and bundling operations, is the same dimension as the original component SHVs.



$Z_{Comms-Plan} = p_0{}^0 \odot Z_{radio-1} \odot Z_{UHFChannel1} + p_0{}^0 \odot p_0{}^1 \odot Z_{radio-2} \odot Z_{UHFChannel3} + . .$
Radio -1
Radio - 2
Radio -M
$Z_{radio1} \odot z_{key-UHFChannel1}$
$Z_{radio2} \odot z_{key-VHFChannel3}$
. . . .
$Z_{radioM} \odot z_{key-UHFChannel3}$
$Z_{radio-1} = Z_{key-id} \odot Z_{value-1} + Z_{key-company} \odot Z_{value-company1} + . . . + Z_{key-SIDC} \odot Z_{value-SFGP UCI---A}$
$Z_{VHFChannel1} = Z_{key-UHFChannel} \odot z_{value-1}$

Figure 11. Hierarchical construction of the Comms-Plan SHV as a vector of vectors

4.3 Logical Operations Performed

When a communications plan change is to be activated, the commander radio assumes the role of a requesting node. The node packs the SHV into a wireless communications packet and multicasts the packet on one of the available communications channels. The sequence of logical operations that are then performed by each node are identical to those described in detail in the SPIE21.[11] It has been shown in Figure 2 (a) that the bundling capacity of the SHVs is equivalent to the type of BSC HVs used in the SPIE21[11] paper when $D = Mlog_2B$. So in the scenario the equivalent SHVs have a $M = 1000$ neurons and $B = 1024$ possible bit positions with a library size $L = 10000$. Since vectors of this size require $Mlog_2B$ bits to communicate the vector the wireless communication requirements are identical to the reported results in the SPIE21[11] paper.

The logical sequence of operations is shown as a flowchart in Figure 12 where each node is in one of two states. It is either acting in a ‘Listening State’ or in a ‘Requesting State’. In Figure 12 the ‘Listening State’ workflow is connected by orange flow and is un-shaded. In the ‘Requesting State’ the flow is connected by blue lines and the actions are shaded. The red dashed lines indicate message transmission/reception. Details of all these operations are given in the SPIE21[11] paper.

Essentially when a comms-plan SHV is transmitted on any radio channel all the VSA enabled radios are in a listening state and respond to any VSA messages on their current channel. On receipt of a VSA message nodes unpack the SHV and compare it with their own SHV library of allowable SHV descriptions. The received vector will always be a noisy representation and so this action must be performed using a ’clean-up’ memory operation. Since each radio can potentially have a large number of possible configurations the size of the local clean-up memory can itself be relatively large but because all radio nodes are performing this action in parallel the actual vector library size (L) used to determine the clean-up memory match threshold is effectively the size of all the allowable configurations across all participating nodes. Each node therefore compares this noisy unbound SHV with its allowable channel SHVs. If there is an above threshold match then this radio is a candidate to perform the requested operation in this step of the plan.

In the current architecture each candidate node performs a set vector exchanges with the current requesting node to determine if it is the best match and if selected it becomes the current requesting node. The node then performs the required service action, unbinds the received vector, packs the SHV into a wireless communications packet and multicasts the packet on one of the available communications channels. The detailed description of these operations are again given in SPIE21[11] paper.

4.4 Integration points for the SNN circuits

In the SPIE21[11] paper a PCM device was described that could perform the required clean-up memory operations for up to 150 10kbit BSC HVs with a significant performance and energy efficiency gain i.e., $over 100x$ compared to performing the same operations using standard CMOS circuits. In this section an SNN circuit is described that can perform the same types of operation but on more SHVs. The circuit can also perform the hierarchical unbinding operation which was not possible using the PCM device.

Generating a hierarchical SHV using the component SNN circuits can readily be achieved using a combination of Right Cyclic Shift and Role-Filler binding and bundling and so the following discussion assumes that the hierarchical workflow vector has been generated. In the logical flow described in Section 4.3 on receipt of a workflow vector the first operation is to compare the noisy unbound workflow SHV to determine if there is a matching configuration. An SNN circuit that can perform this operation is illustrated in Figure 13. The circuit is a combination of the sparse SHV generation circuit, the role-filler and left cyclic-shift unbinding circuits and the clean-up memory circuit. However the cyclic shift connections now encompass the role-filler binding as shown. The function of the circuit is to perform the vector comparisons but also to determine the current unbound state of the workflow vector and using a recurrent connection between the clean-up memory and the vector generation components to select the appropriate position vector to perform the next unbinding operation.

The structure of an SHV workflow vector is described by Equation 7. Each component of the vector represents a step in the workflow with cyclically shifted node description vectors, Z_i^n, being bound to position SHVs e.g., $P_0 \odot P_1 \ldots \odot P_n$, where n is the position of the service in the workflow. The operation to unbind the workflow vector is described by Equation 8 which requires a role-filler unbinding operation between a cyclically shifted position SHV e.g., $P_0^0 .\ P_1^1$ and the Workflow SHV. The resulting vector is then left cyclically shifted to expose the required noisy version of the required service vector.

The challenge is that when a vector is received it is necessary to determine how many times it has been previously unbound. To do this an extra SHV, the T-vector, is bundled with the unbound workflow vector. As the vector is unbound the T-vector is also operated on and so after n unbinding operations this vector will have been modified to:

$$p_1^{-n} \odot P_2^{-n} \odot \ldots P_n^{-n} \ldots T^{-n} \tag{15}$$

The vector generation component stores the shifted position vectors $P_0^{-}1 \ldots P_n^{n-1}$ together with a null SHV where all time delays are set to zero and on receipt the received workflow SHV is also stored in the memory.

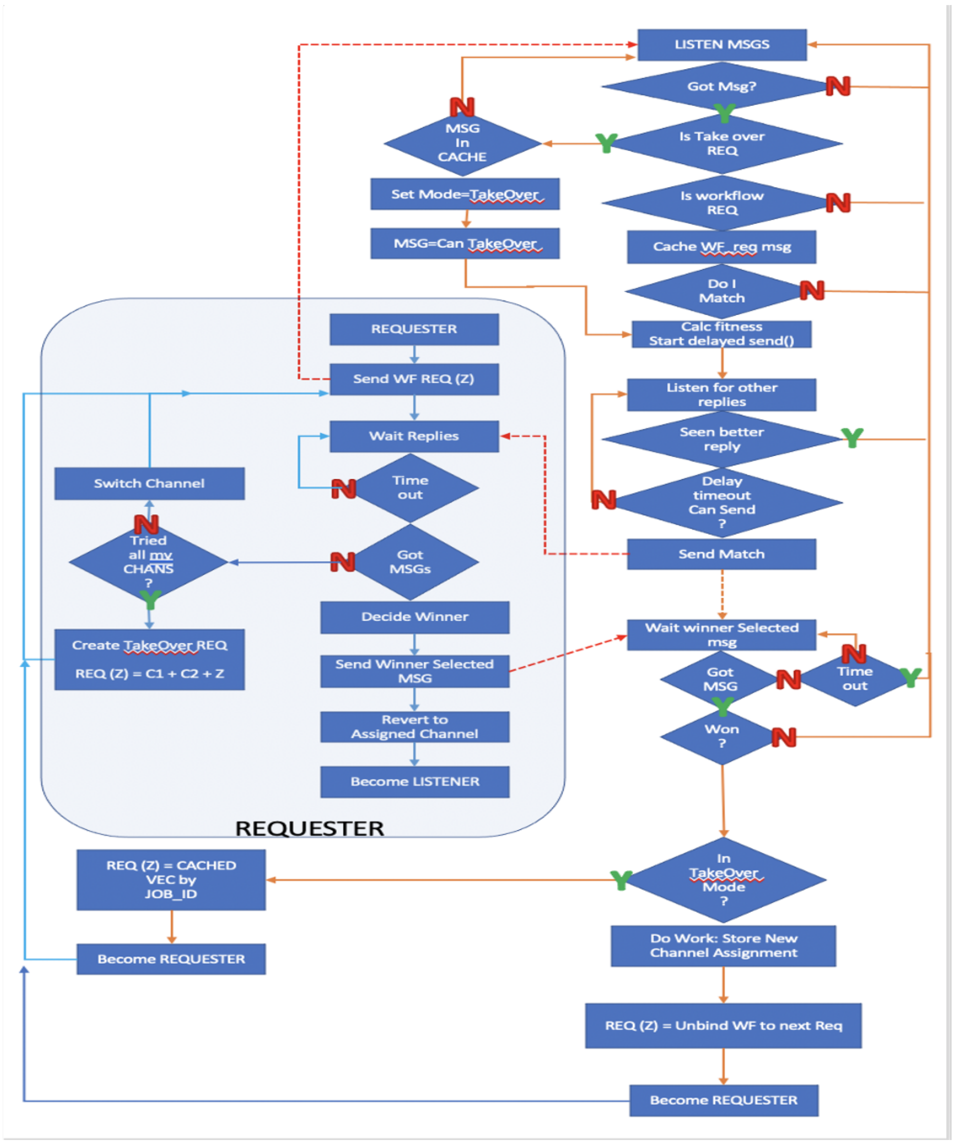


Figure 12. Flow diagram showing the logical sequence of operations that each radio node performs to achieve the required sequence of operations

To initiate the process the null vector and workflow vector are triggered and the subtraction unbinding layer and left cyclic shift neurons simply passes through the workflow vector with no change. Depending on the number of previous unbinding operations this vector will be:

$$Z_x = Z_n + p_1^{-n} \odot P_2^{-n} \odot \ldots P_n^{-n} \ldots T^{-n} + noise \tag{16}$$

The clean-up memory stores vectors representing all of the different possible n states of the T vector together with all of the configuration SHVs Z_k that the node can perform.

When an unbound SHV is injected into the clean-up memory this will result in the excitation of any service SHV that matches Z_n and the corresponding modified T-vector. This will result in the two corresponding neurons firing as shown at time $t_0 + B\Delta$. The firing of the neuron corresponding to Z_n is the indication that there is a match on one of the configurations that the service can perform and so it is a candidate to perform this step of the workflow. The output of the matching modified T-vectors is fed back to the position vector that is required to unbind the next step with a delay of $B\Delta - 2 * \delta$ seconds. Since the feedback delay of the left-cyclic shift is $B\Delta - \delta$seconds the cyclically shifted vector Zx^{-1} and the triggered position vector are injected into the unbinding circuit and the next step in the workflow is revealed. This circuit can be used to perform a complete unbinding of a workflow vector if required - or in the case of a service the cycle only has to be completed once to determine if there is a match and to obtain the unbound vector for transmission as the new requester.

This circuit therefor performs a similar clean-up memory operation to that described in the SPIE21[11] paper using the PCM processor plus the unbinding operation which had to be performed in near-memory alongside the PCM device.

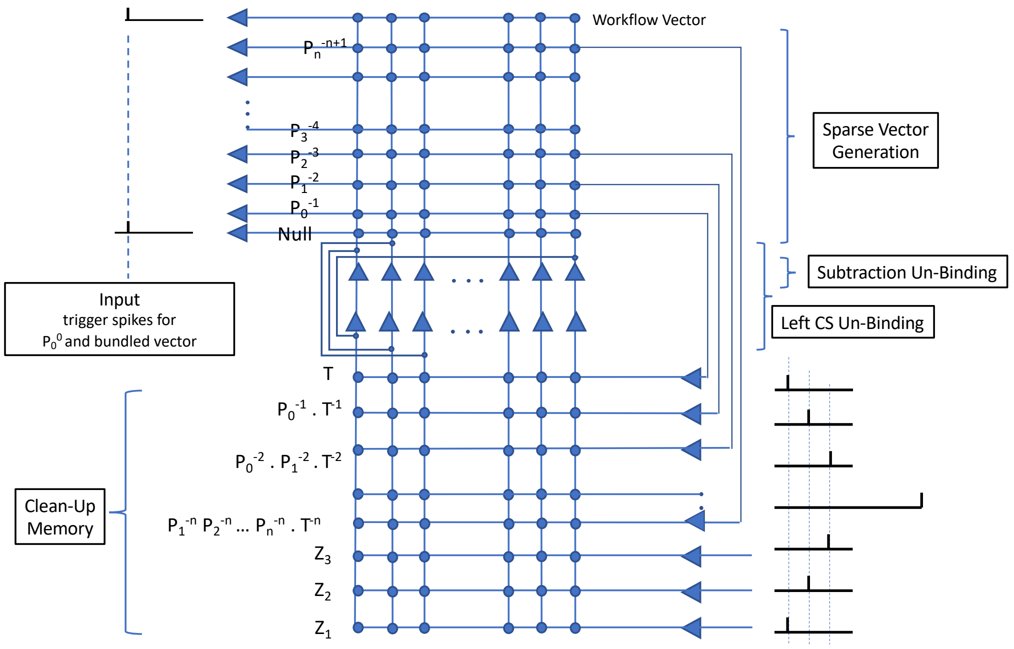


Figure 13. Hierarchical SNN Unbinding Circuit

4.5 Estimated Energy Efficiency of the SNN Circuits Vs PCM

The results from the Brian 2 simulations have demonstrated that it is possible to implement VSA operations using energy efficient SNN processing devices. The amount of energy consumed by a neuromorphic processing device is a function of the number of spikes required to perform the computation. Using a sparse vector encoding and a *time-to-spike* SNN processing and maintaining sparcity as the SHV binding and bundling operations are performed minimises the number of spikes required to perform these operations.

An estimate of the energy required can be made using the TrueNorth experimental SNN processor as a guide. This processor consumes an estimated 45pJ per spike.[16] An SHV cyclic-shift unbinding circuit using $M = 1000$ neurons and $B = 1024$, for each unbinding operation the energy cost is in the region of 45nJ and would take $B\Delta$ seconds to complete. Similar estimates can be made for the binding and bundling circuits. In contrast, estimates for a CMOS cyclic-shift circuit is typically in the region of 100pJ per bit,.[27] Thus, the cyclic shift operation on a equivalent $D = 10000$ bit BSC HV (i.e.,

$D = M \log_2 B$) would require in the region of 1 µJ. Hence, the SNN process would potentially be approximately 20x more energy efficient, however there is a clear trade-off to be made between energy efficiency gain and processing time latency.

In the case of the clean-up memory circuit the cost of performing the matching operations will depend on the cost of the synaptic delays rather than simply the number of spikes. Current SNN devices such as TrueNorth do not support time delays on each synaptic connection and so the energy cost of performing the clean-up memory operations will therefore depend on the efficiency with which future SNN neuromorphic devices can perform the required time delay synaptic processing. In terms of a baseline comparison an experimental Phase Change Memory (PCM) device has demonstrated the capability to perform a 300 vector cleanup memory for 10,000 bit HVs using just 9.44 nJ per query an improvement in total energy efficiency of x 117.5 compared to the equivalent CMOS implementation,.[15] These figures give some indication of the energy efficiencies that would be required for synaptic time delay processing for an SNN processor that could perform an equivalent SHV clean-up memory. However this requirement would need to be balanced against the fact that the SNN processor could also perform all the other required VSA SHV operations.

5. CONCLUSION AND FUTURE DIRECTIONS

Work presented at SPIE21[11] showed how a Vector Symbolic Architecture (VSA) approach, using densely encoded binary hypervectors (HV), could be used to discover services distributed across low bandwidth communications networks and how distributed service workflows could be orchestrated. This was demonstrated in the context of a communications replanning MDO scenario. The motivation for this paper was to show that by using a different sparse hypervector representation(SHV), based on a slot encoding model, that the same VSA operations could be performed and that this type of encoding could be mapped into energy efficient spiking neural network circuits to perform the VSA processing.

Theoretical and empirical comparison of the VSA binding and bundling capacity of dense encoded binary HVs (i.e., HVs with 10,000 bits used in the previous work) with VSA SHVs encoded using a sparse slot encoding model proposed by[18] are presented. It is shown that the two representations are essentially equivalent, in terms of bundling capacity,specifically when low unbinding error rates ($1in10_{-6}$ and large vector libraries are required such as in the example workflow orchestration scenario. Under these conditions SHV unbinding $D = Mlog2B$, where M are the number of slots. B the number of bits per slot and D the dimension of the BSC HV. This result is significant since it shows that the communications cost involved in the vector exchange is identical between the two representations. Interestingly if lower unbinding accuracy is acceptable then communicating SHVs is more efficient.

Using the slot encoding formulation the paper has demonstrate how the VSA operations can be mapped into SNN circuits. The resulting circuits can perform all of the basic VSA functions including cyclic-shift and role-filler binding and bundling, cyclic-shift and role-filler unbinding and the required clean-up memory. Results using Brian 2 simulations of these circuits show that they perform close to the theoretical predictions and that, using various approximations to the[18] model, highly efficient SNN circuits minimise the number of neurons and spikes required to perform the VSA operations were created.

Energy estimates are provided for performing typical VSA functions and energy efficiencies of x20 are estimated compared to performing the same operations using standard CMOS operation on 10,000 bit BSC HVs. Further energy savings can be made by reducing the number of neurons but this is at the cost of extending the required processing time for each VSA operations. Recommendations for future SNN neuromprphic processor design to support VSA processing, particularly in relation to supporting synaptic time delays have been made.

Current work is focused on extending the VSA circuits to perform hierarchical binding and bundling which requires cyclic-shift and role-filler operations to be combined into a single circuit. An alternative adaptive approach to determining the spike threshold voltage for the output neurons of the clean-up memory is also being investigated. Future work will investigate how SNN circuits can be developed to perform typical VSA processing using semantic SHVs to perform analogical mapping, logical reasoning with a specific focus on abductive reasoning as a form of cognitive processing. The full capabilities that can be enabled by VSA operations in SNN circuits are still to be explored but this paper shows that it offers the potential to develop a new generation of energy efficient artificial intelligence capabilities.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] Verma, D., Bent, G., and Taylor, I., "Towards a distributed federated brain architecture using cognitive iot devices," in [*9th International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 17)*], (2017).

[2] Gayler, R. W., "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059* (2004).

[3] Kanerva, P., "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation* **1**(2), 139–159 (2009).

[4] Hinton, G. E., "Mapping part-whole hierarchies into connectionist networks," *Artificial Intelligence* **46**(1-2), 47–75 (1990).

[5] Eliasmith, C., "How to build a brain: from function to implementation," *Synthese* **159**(3), 373–388 (2007).

[6] Widdows, D. and Cohen, T., "Reasoning with vectors: A continuous model for fast robust inference," *Logic Journal of the IGPL* **23**, 141–173 (11 2014).

[7] Kleyko, D., *Pattern Recognition with Vector Symbolic Architectures*, PhD thesis, Luleå tekniska universitet (2016).

[8] Levy, S. and Gayler, R., "Vector symbolic architectures: A new building material for artificial general intelligence," in [*Frontiers in Artificial Intelligence and Applications*], **171** (Jan. 2008).

[9] Plate, T. A., [*Distributed representations and nested compositional structure*], University of Toronto, Department of Computer Science (1994).

[10] Kanerva, P., "Binary spatter-coding of ordered k-tuples," in [*Artificial Neural Networks — ICANN 96*], von der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., eds., 869–873, Springer Berlin Heidelberg, Berlin, Heidelberg (1996).

[11] Bent, G., Simpkin, C., Taylor, I., Rahimi, A., Karunaratne, G., Sebastian, A., Millar, D., Martens, A., and Roy, K., "Energy efficient 'in memory' computing to enable decentralised service workflow composition in support of multi-domain operations," in [*Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*], Pham, T. and Solomon, L., eds., **11746**, 414 – 435, International Society for Optics and Photonics, SPIE (2021).

[12] Simpkin, C., Taylor, I., Bent, G. A., de Mel, G., and Ganti, R. K., "A scalable vector symbolic architecture approach for decentralized workflows," in [*COLLA 2018 The Eighth International Conference on Advanced Collaborative Networks, Systems and Applications*], 21–27, IARIA (2018).

[13] Simpkin, C., Taylor, I., Bent, G. A., de Mel, G., Rallapalli, S., Ma, L., and Srivatsa, M., "Efficient orchestration of node-red iot workflows using a vector symbolic architecture," *Future Generation Computer Systems* **100**, 70–85 (2019).

[14] Simpkin, C., Taylor, I., Bent, G. A., Harbourne, D., Preece, A., and Ganti, R., "Constructing distributed time-critical applications using cognitive enabled services," *Future Generation Computer Systems* **111**, 117–131 (2020).

[15] Karunaratne, G., Le Gallo, M., Cherubini, G., and et al., "In-memory hyperdimensional computing," *Nature Electronics* **3**, 327–337 (2020).

[16] Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., and Modha, D. S., "A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm," in [*2011 IEEE Custom Integrated Circuits Conference (CICC)*], 1–4 (2011).

[17] Orchard, G., Frady, E. P., Rubin, D. B. D., Sanborn, S., Shrestha, S. B., Sommer, F. T., and Davies, M., "Efficient neuromorphic signal processing with loihi 2," *CoRR* **abs/2111.03746** (2021).

[18] Laiho, M., Poikonen, J. H., Kanerva, P., and Lehtonen, E., "High-dimensional computing with sparse vectors," in [*2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*], 1–4 (2015).

[19] Frady, E. P., Kleyko, D., and Sommer, F. T., "Variable binding for sparse distributed representations: Theory and applications," *CoRR* **abs/2009.06734** (2020).

[20] Simpkin, C., *A Vector Symbolic Approach for Cognitive Services and Decentralized Workflows*, PhD thesis, COMSC (2021). unpublished thesis.

[21] Simpkin, C., Taylor, I., Bent, G. A., De Mel, G., and Rallapalli, S., "Decentralized microservice workflows for coalition environments," in [*2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*], IEEE (2017).

[22] Frady, E. P., Kleyko, D., and Sommer, F. T., "A theory of sequence indexing and working memory in recurrent neural networks," *CoRR* **abs/1803.00412** (2018).

[23] Recchia, G., Sahlgren, M., Kanerva, P., and Jones, M. N., "Encoding sequential information in semantic space models: comparing holographic reduced representation and random permutation," *Computational intelligence and neuroscience* **2015**, 58 (2015).

[24] Stimberg, M., Brette, R., and Goodman, D. F. M., "Brian 2: an intuitive and efficient neural simulator," *bioRxiv* (2019).

[25] Mikolov, T., Chen, K., Corrado, G., and Dean, J., "Efficient estimation of word representations in vector space," *CoRR* **abs/1301.3781** (2013).

[26] Rachkovskij, D., "Estimation of vectors similarity by their randomized binary projections," *Cybernetics and Systems Analysis* **51**(5), 808–818 (2015).

[27] Joshi, N. and Jangir, R., "Design of low power and high speed shift register," (2019).