A demonstration of vector symbolic architecture as an effective integrated technology for AI at the network edge

Graham Bent[a], Cai Davies[b], Marc Roig Vilamala[b], Yuhua Li[b], Alun Preece[b], Gaetano Di Caterina[c], Alex Vicente Sola[c], Paul Kirkland[c], Gavin Pearson[d], and Benomy Tutcher[e]

[a]Neurosynapse Ltd, UK
[b]Security, Crime and Intelligence Institute, Cardiff University, UK
[c]Neuromorphic Sensor Signal Processing Lab, EEE Dept, University of Strathclyde, UK
[d]Defence Science and Technology Laboratory, Porton Down, UK
[e]Frazer-Nash Consultancy Ltd, Bristol, UK

ABSTRACT

Vector Symbolic Architecture (VSA), a.k.a. Hyperdimensional Computing has transformative potential for advancing cognitive processing capabilities at the network edge. This paper presents a technology integration experiment, demonstrating how the VSA paradigm offers robust solutions for generation-after-next AI deployment at the network edge. Specifically, we show how VSA effectively models and integrates the cognitive processes required to perform intelligence, surveillance, and reconnaissance (ISR). The experiment integrates functions across the observe, orientate, decide and act (OODA) loop, including the processing of sensed data via both a neuromorphic event-based camera and a standard CMOS frame-rate camera; declarative knowledge-based reasoning in a semantic vector space; action planning using VSA cognitive maps; access to procedural knowledge via large language models (LLMs); and efficient communication between agents via highly-compact binary vector representations. In contrast to previous 'point solutions' showing the effectiveness of VSA for individual OODA tasks, this work takes a 'whole system' approach, demonstrating the power of VSA as a uniform integration technology.

Keywords: Vector Symbolic Architectures, Hyperdimensional Computing, Artificial Intelligence, Cognition, OODA, ISR

1. INTRODUCTION

The enduring goal of artificial intelligence (AI) is to address all of the key processes in human cognition, including perception, attention, memory, decision-making, problem-solving and communication: these underpin the Observe, Orientate, Decide and Act (OODA) loop. Additionally, defence needs an AI solution able to operate in a distributed manner at the edge, with contested communications and data, and limited compute and power.

In a companion paper[1] we have shown how Vector Symbolic Architecture (VSA), also known as Hyperdimensional Computing (HDC), has a unique set of attributes, in comparison with other AI approaches (including symbolic, sub-symbolic, or hybrid), making it ideal for providing such distributed AI at the point of need. The integrated demonstration seeks to show that VSA can perform the set of necessary and sufficient cognitive processing functions associated with the OODA loop and, moreover, that VSA is effective, efficient, noise robust, data-frugal, and can integrate other AI approaches and integrate distributed nodes in a resilient manner.

In this context, the integrated demonstration emphasises interoperability:

- Across AI-supported functions (perception, attention, memory, decision-making, problem-solving and communication);

Further author information: (Send correspondence to Alun Preece) E-mail: PreeceAD@cardiff.ac.uk, Telephone: +44 29 2087 4653 or Graham Bent E-mail:graham.bent@neurosynapse.co.uk.

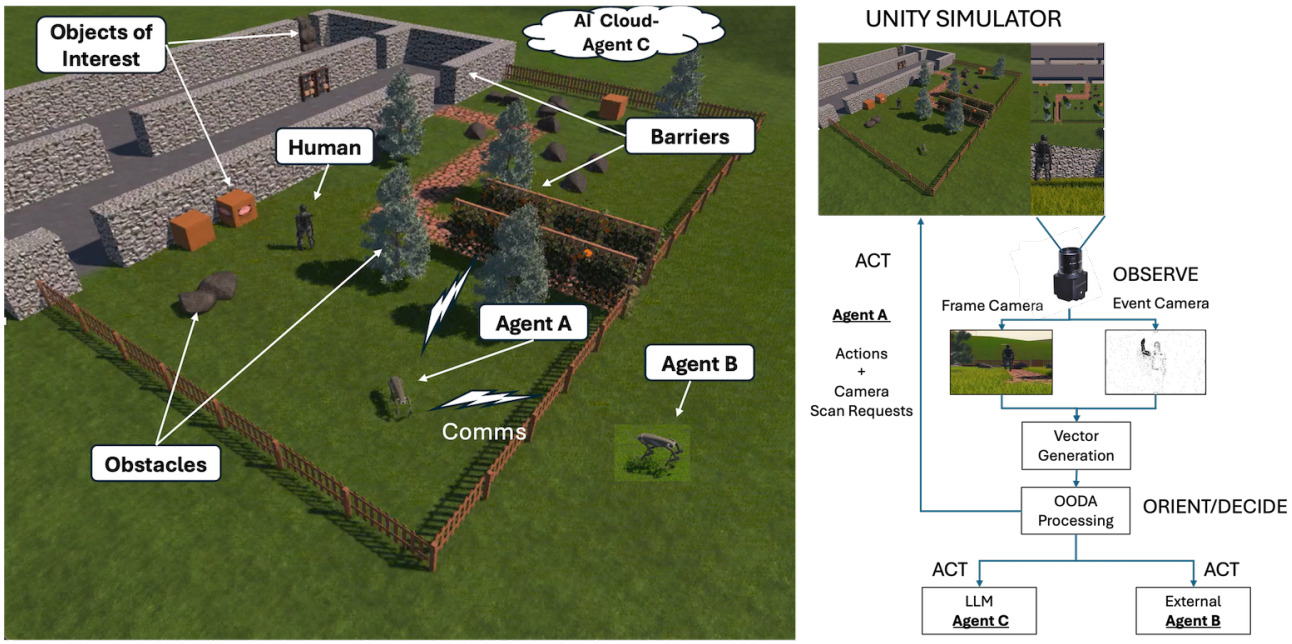

Figure 1. Demonstration environment showing the major elements in the Unity simulation environment and the control feedback loop to the simulated Agent A.

- Between other agents at the edge and in the cloud; and between different fundamental models of AI (symbolic and sub-symbolic).

The demonstration is situated in a human-robot teaming scenario featuring:

1. Integration with low size, weight, power and cost (SWaP-C) sensing to demonstrate compatibility between VSA and other edge AI technologies: specifically, a spiking neural network (SNN) based event-based camera for gesture recognition;
2. Efficient inter-agent communication in a contested setting: specifically, the exchange of local knowledge among VSA-based agents at the edge;
3. Uniform VSA approach to perception, attention, memory, planning, and communication; and
4. Integration with a high-resource AI, specifically a large language model (LLM) equipped agent to access 'open world' knowledge and reasoning.

Note that the applicability of this work is not limited to the human-robot teaming domain; this scenario is just an illustration of one of many sets of integrated functionalities.

An overview of the demonstration environment, realised using the Unity engine* is shown in Figure 1. The demonstration is centred on the behaviours of a simulated robot (Agent A) equipped with Low SWaP-C sensing (frame-rate and event-based cameras) and uniform integrated VSA vector processing. Agent A interacts with a simulated human soldier who commands it with gestures that it needs to understand and carry out corresponding courses of action using objects in the environment whilst avoiding obstacles and barriers. At times, Agent A will need to communicate (exchange knowledge) with other machine agents including peers at the edge (Agent B) and more powerful AI agents (Agent C) nearer the centre via the cloud; the agents exchange rich knowledge using efficient VSA semantic vector encoding.

*https://unity.com

To provide the sensor input to the VSA processing we process output from each of two cameras that are physically viewing the simulated agent view. The first camera is a spiking neural network (SNN) based event-based camera which responds to moving objects, and the second is a standard frame rate camera (FRC). What the cameras observe, together with the position and pointing direction of the Agent, are converted into VSA vectors that provide the input for all the VSA processing.

Agent A is a simulated robot dog with a fixed forward-looking view. To change the agent view, instructions must be given as to which direction to 'look' and the robot must rotate to face that direction. Similarly, instructions have to be given to instruct the agent which direction it should move and in this simplified simulation this is restricted to movement in one of eight possible directions. The simulator also models kinetic effects and so Agent A also has a simulated proximity detector which it also uses to plan a course of action (COA) that achieves different goals whilst avoiding collisions with the objects and barriers.

Everything Agent A does is performed using 'Near-Memory' and 'In-Memory' VSA vector processing[2] and demonstrates the cognitive functions of perception, attention, memory, decision-making, problem-solving and communication that are all required to continuously perform an OODA loop. We also demonstrate how Agent A can use VSA vectors to efficiently communicate with other agents (Agents B and C) to obtain knowledge that it does not have and to then act in response to new knowledge that it gains.

The next following sections describe the key technical elements of the demonstration in detail. Section 2 describes the two sensors used for the demonstration and how VSA camera vectors for both gestures and objects are generated; Section 3 describes the core VSA OODA loop performed by Agent A and the associated cognitive processing; Section 4 describes the various VSA vectors that are used and Section 5 explains how the vector manipulations are performed using Near-Memory and In-Memory Processing. Finally, Section 6 presents the conclusions and future work. A link to a video showing the output screen of the Unity simulation as the agent performs the sequence of tasks described in Section 3 can be found at Figure 3.

2. AGENT SENSOR INPUTS

In this section we describe the two cameras used in the demonstration and how the different camera signals are processed to produce VSA hypervectors for the different gestures and objects that the agent observes.

2.1 Gesture Recognition Using the Event-Based Camera

Event-based cameras, also known as neuromorphic vision sensors (NVS), represent a paradigm shift in computer vision. By mimicking the change detection mechanisms of biological retinas, they offer a unique, event-driven approach to visual sensing. The combination of event-based cameras and spiking neural networks (SNNs) offers an effective and efficient method for edge-based spatio-temporal processing that can interface with the VSA OODA loop. Event-based cameras operate by detecting changes in light intensity at the pixel level, rather than capturing entire frames at fixed intervals. When a pixel detects a significant change, it generates an "event", a data packet containing the pixel's coordinates (x, y), the time of the change (t), and the polarity of the change (p, indicating whether the brightness increased or decreased) to form a $[x, y, t, p]$ data packet. This event-driven approach offers high temporal resolution, reduced data rate, and lower power consumption compared with traditional cameras. Within this demonstration environment, the event-based camera provides a simpler method to detect hand field signals being gestured by the human towards the agent. As in the otherwise static scene, these gestures are the only salient information captured by the sensor, streamlining the visual scene understanding task.

2.2 Spiking Convolutional Neural Networks (SCNNs)

Spiking Convolutional Neural Networks (SCNNs) are a type of artificial neural network inspired by the spiking behaviour of neurons in the biological brain. They are designed to efficiently process the asynchronous, event-based output of event-based cameras. By utilising spikes as the fundamental unit of information transmission, SCNNs offer energy efficiency, real-time processing capabilities, and the ability to effectively handle spatio-temporal data. Within this example, the SCNN learns to recognise gestures in an unsupervised manner using Spike Timing-Dependent Plasticity (STDP), meaning it uses the co-location of spikes through time (and space

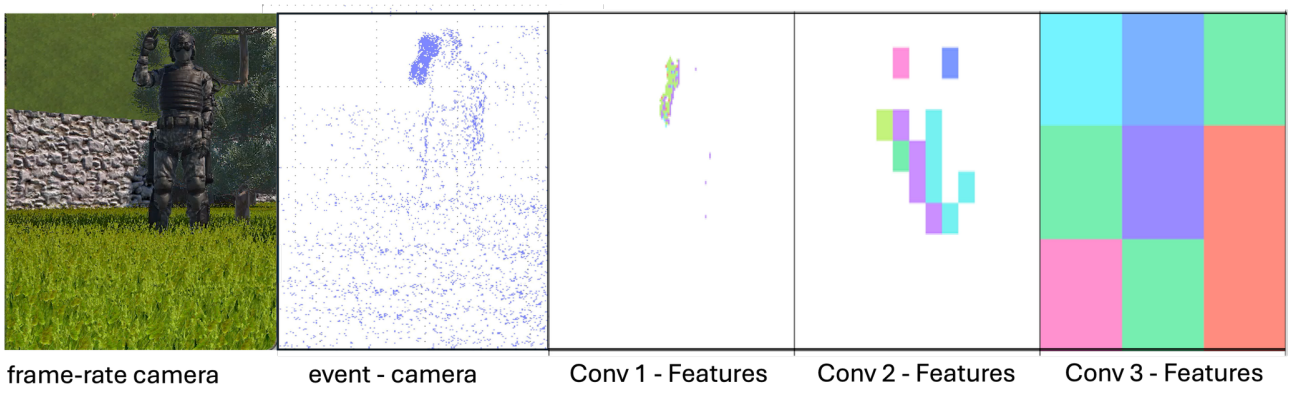

Figure 2. Example of event camera processing showing simulated agent view, the corresponding event camera output and neuron firing patterns in each of the three SCNN convolution layers (colour coding reflects the neuron firing rate).

thanks to the convolution kernels) to learn salient features. This creates a sparsely activated network, where neurons learn independent spatio-temporal features. Training in this manner facilitates a simple method in which the SCNNs can also be used to create feature vectors to pass to the VSA system, by collecting the spiking activity of the neurons throughout the processing of the visual scene. A unique latent space spiking feature vector is created for each of the individual gestures following what spatio-temporal features were activated in the network.

2.3 Gesture Recognition with Event-Based Cameras and SCNNs

Figure 2 illustrates how event-based cameras excel at capturing the subtle changes in light patterns caused by hand movements, making them ideal for this gesture recognition task. The figure shows the simulated agent view, the corresponding event camera output and neuron firing patterns in each of the three SCNN convolution layers.

2.3.1 Gesture Representation as Hypervectors

The representation of the observed gesture as a VSA hypervector is achieved by using the SCNN to processes the sensor's event stream from the second convolutional layer which produces a 40x16 matrix representing the spiking activity of 40 neurons over 16 temporal steps. This matrix serves as the input for further gesture classification and analysis. The simple approach used to convert this representation into a VSA hypervector is to flatten the matrix into a 640-dimension vector, normalise to the range [0,1], and to project into the hypervector binary representation. Projection is done by multiplication with a random (but constant) binary matrix with summation and thresholding applied.[3] In the demonstration all of the VSA processing was performed using 2kbit hypervectors. Where the same gesture is observed from multiple directions the resulting vectors are bundled to create exemplar vectors for each gesture type.

2.3.2 Exemplar Gesture Hypervectors

In addition to obtaining gesture vector representations from the event camera, using the binding and bundling operations that VSA provides, we can construct new exemplar vectors that combine vectors representing other attributes, such as the semantic meaning of the gesture as well as the sequence of actions that the agent should perform in response to the gesture (i.e., we can represent everything we know about the observation in a single exemplar vector). To do this we create a compound VSA vector of the gesture vector, a semantic description vector and a workflow vector using role-filler binding to distinguish the different components. The semantic description vector component is used to describe the meaning of the gesture, for example 'Follow Me', and is obtained using a language model such as BERT[4] and projecting the BERT vector embedding into the VSA vector space.[3] This allows for a soft matching with similar commands such as 'Go Ahead' and 'Move Forward'. Similarly the workflow vector is itself a compound vector of vectors that represents a semantic embedding of the actions to take, i.e., it represents the workflow associated with the command. Exemplar gesture vectors were

generated for four different gestures 'wave', 'salute', 'T-shape' and 'point', representing commands to 'follow me', 'follow', 'move-forward' and 'a problem!' respectively. Further details of how these exemplar vectors were constructed for the demonstration are presented later in Section 4.1

2.4 Object Recognition with Frame Rate Cameras and Transformer Neural Networks (TNNs)

Object recognition using the frame rate camera takes much the same approach as gesture recognition with the key difference being that object detection is performed by using a combination of a regular frame rate camera and transformer-based neural network (TNN) models for zero-shot object detection and vector generation. For demonstration purposes it was recognised that the pixels from the simulated agents view were sufficiently accurate representation of a real frame rate camera observing the simulator output and therefore representative vectors for the objects seen could be generated directly from the simulation.

2.4.1 Object Representation as Hypervectors

Firstly, we need to detect objects within the scene. To do this, we use a zero-shot vision-transformer model called Grounding DINO[5] [†], which extends a closed-set object detection model with a text encoder, allowing us to use zero-shot labels to detect objects; i.e., the input is an image (of the scene) and a list of object labels to detect, which allows for a generic object detector. The aim is to detect objects and create a bounding box to crop to. A segmentation model that is trained on the specific domain would potentially perform significantly better but the current approach proved to be sufficient to demonstrate the process of representing detected objects as VSA hypervectors.

Given the subset of pixels from the bounding box of each detected object, we create a corresponding embedding vector using a vision-transformer model, such as Google's ViT-base[6] [‡], which creates a 768-dimension real-valued vector as the features of the image. As before, we can project this into a 2kbit binary vector, which is then our observation VSA hypervector for that particular object.

For each object or gesture class, we collect N observations, which is to say we collect N binary vectors that are generated as described above. We use a k-means clustering algorithm to generate M clusters for each class. For each cluster, we then bundle the constituent vectors to create M compound vectors. The size M is dependent on memory constraints, and given X classes, we have $X \times M$ vectors to keep in memory.

The demonstration had a small object class list of [Shelf, Rocks, Soldier, Tree, Crowbar, Box/Crate] with 7,695 different observations of all classes through 90 seconds of the demonstration video. With $M = 10$ this produced 60 exemplar vectors. An accuracy of 98.4% was achieved when matching all observations to these representative vectors. Note that this accuracy isn't for unseen observations, it merely demonstrates how much we can compress all individual observations and still maintain a representation of the classes. The resulting representational binary hypervectors therefore accurately represent the different objects and similarity can be determined using Hamming similarity and a simple threshold since dissimilar objects were demonstrated to be quasi-orthogonal which it should be noted, is an important property of VSA vectors.

2.4.2 Exemplar Object Hypervectors

Similar to the exemplar gesture vectors we can construct exemplar vectors for the detectable objects. Again, the exemplar vectors combine semantic vectors representing the the object type (e.g., a crowbar) but the workflow component represents the sequence of actions the agent can perform on the object or with the object (e.g., 'pick-up', 'carry', 'put down'). Again further details are presented in Section 4.1.

[†] https://huggingface.co/IDEA-Research/grounding-dino-base

[‡] https://huggingface.co/google/vit-base-patch16-224

3. CORE OODA LOOP PROCESSING

This section describes the core of the OODA loop processing performed in the demonstration scenario by Agent A. This subsystem is responsible for instructing simulated Agent A to perform a sequence of tasks in response to gestures made by the simulated human. The simulated Agent A can be instructed to move to a specific location, expressed in terms of X and Y co-ordinates of the simulated world grid, and to instruct Agent A to 'Look' in a particular direction expressed as an angle relative to grid orientation (i.e., zero degrees is in the direction of the positive Y-axis and ninety degrees in the direction of the positive X-axis). The agent is restricted to move only to any of the eight adjacent grid squares in any one move. Each time an instruction is issued the simulated camera view from the agent is updated and vectors are generated for each object in Agent A's field of view (FOV) as described in Section 2.4.

The objective of the demonstration is not primarily human-robot interaction but rather to demonstrate that all the cognitive functions required to perform the OODA loop processing can be performed using 'Near-Memory' and 'In-Memory' VSA vector operations. The input to the OODA loop performing subsystem at each time-step are vectors that represent the agents current position, the agent's orientation and hence the camera 'Look Angle' and a compound vector that represents the objects observed in the agents camera FOV. Further details of these vectors and other vectors derived from them are described in Section 4.

Agent behaviour is driven by a goal of navigating to the location of various goal objects. These objects can be at fixed locations or can move around the simulated environment as exemplified by the simulated human. The sequence in which goal objects are reached is determined by a compound vector termed the current workflow vector. All of the vectors used have a common VSA representation and are stored in the memory of Agent A. Since we are using binary VSA hypervectors, sometimes referred to as binary spatter codes or BSC, the agent's memory forms a sparsely populated self-organising hypercube where each vector is a node in the hypercube and similar vectors are close together. The agent memory is therefore self organising in the sense that vectors that represent things that are semantically similar are close together in the hypercube, where closeness is measured using the Hamming similarity. Things that are dissimilar are mapped into different areas of the memory and are essentially orthogonal to other vectors. This property is exploited to advantage to facilitate much of the required cognitive processing.

Agent A is also assumed to have a proximity detector which is simulated by identifying grid squares into which the agent cannot move since they are occupied by a barrier or other obstacle.

The output from the OODA loop performing subsystem is limited to instructions to the simulator to update the agent's position and look angle. All the OODA processing is therefore driven simply from the angle of observation of identified objects in Agent A's FOV. If an object or gesture is seen that Agent A cannot identity, then it can seek assistance from other agents (e.g., Agent B) to obtain additional information that can be used to perform its various goals or through interaction with Agent C to demonstrate the cognitive task of problem solving.

Importantly, all the processing performed by Agent A is done using the VSA vector operations of bundling, binding and interactions with a vector memory using vector similarity (i.e., Hamming distance) to construct and use various cognitive maps and to determine the next action(s) that Agent A must perform to achieve its goal. Utilising the properties of VSA vectors, all the OODA loop performing subsystem operations are performed using 'In-Memory' or 'Near-Memory' vector processing which emulate the neuromorphic processing operations from which VSA obtains its 'brain' inspired functions. These are described in further detail in Section 5.

3.1 Logical Processing Sequence

The logical sequence of operations that are performed by Agent A is shown in Figure 4. In the scenario all instructions to Agent A are determined by interpreting the gestures made by the simulated Human. The human is treated as an object that can be detected by the Frame Rate Camera (FRC)and, when a human is in the FOV of the FRC, then any vectors generated by the event camera (EC) are assumed to be possible gesture vectors. These vectors are compared to known gesture exemplar vectors and if there is a match the corresponding workflow becomes the new 'Current Workflow'. If the camera gesture vector is unknown, then this vector is broadcast to determine if any other agent (e.g., Agent B) recognises the vector. If Agent B does recognise the gesture vector,

Figure 3. Video 1 Showing the output of the Unity simulator as Agent A performs the range of cognitive operations including perception, attention, decision making, problem solving, planning and learning. The video recording runs at 5x real time. http://dx.doi.org/doi.number.goes.here

then it returns a corresponding exemplar vector which is added to the memory of Agent A. One important nuance here is that if Agent A does not have the same capabilities as Agent B it might not be capable of implementing the workflow returned by Agent B. In this case it compares the semantic meaning of the exemplar vector (e.g., 'follow') with other exemplar gesture vectors that it holds in memory. If there is a match on the meaning it can infer that the unrecognised gesture is equivalent and so can implement the appropriate actions.

The initial workflow is therefore the single FRC vector that represents a human which is formed by bundling the vectors representing a human from the camera processing subsystem as described in Section 2.4. A video showing the output screen of the Unity simulator showing Agent A performing the following range of cognitive tasks can be found from the link in Figure 3.

3.1.1 Observation and Perception

To demonstrate the cognitive functions of Observation and Perception, the demonstration begins with Agent A having to locate the human by scanning the environment, building a cognitive map of what is perceived at the different scan angle and, when a human is detected, focusing attention on the human. The FOV of the camera is 10 degrees and the scan angle range is initialised to a maximum of +/- 180 degrees in alternating 10-degree steps (e.g., 0, -10, 10, -20, 20, ..., degrees). Figure 5 shows the FRC view at representative angles of -20, 0, +20 and +40 degrees from location(1,1). At each scan angle any object in the FOV results in the generation of a corresponding FRC vector and these are in turn bundled into two vectors at each location. One of the vectors represent all objects observed at the current location and the other vector records the objects and the scan angle at which they were observed. Further details are given below in Section 4.5.

The scan continues until either the current goal object is detected, or the maximum scan angle is reached. If the goal object is a human and a human is in the FOV at some scan angle then a check is made to determine if the human is also gesturing (i.e., the event camera produces a gesture vector at the same angle). If the gesture vector corresponds to a known gesture, then the workflow corresponding to the gesture vector becomes the current workflow and the cycle repeats as shown in the logical sequence Figure 4.

3.1.2 Attention and Decision Making

To demonstrate the cognitive functions of attention and decision making, the demonstration requires that when Agent A observes a human it then focuses attention on the human to determine if the human performs a gesture

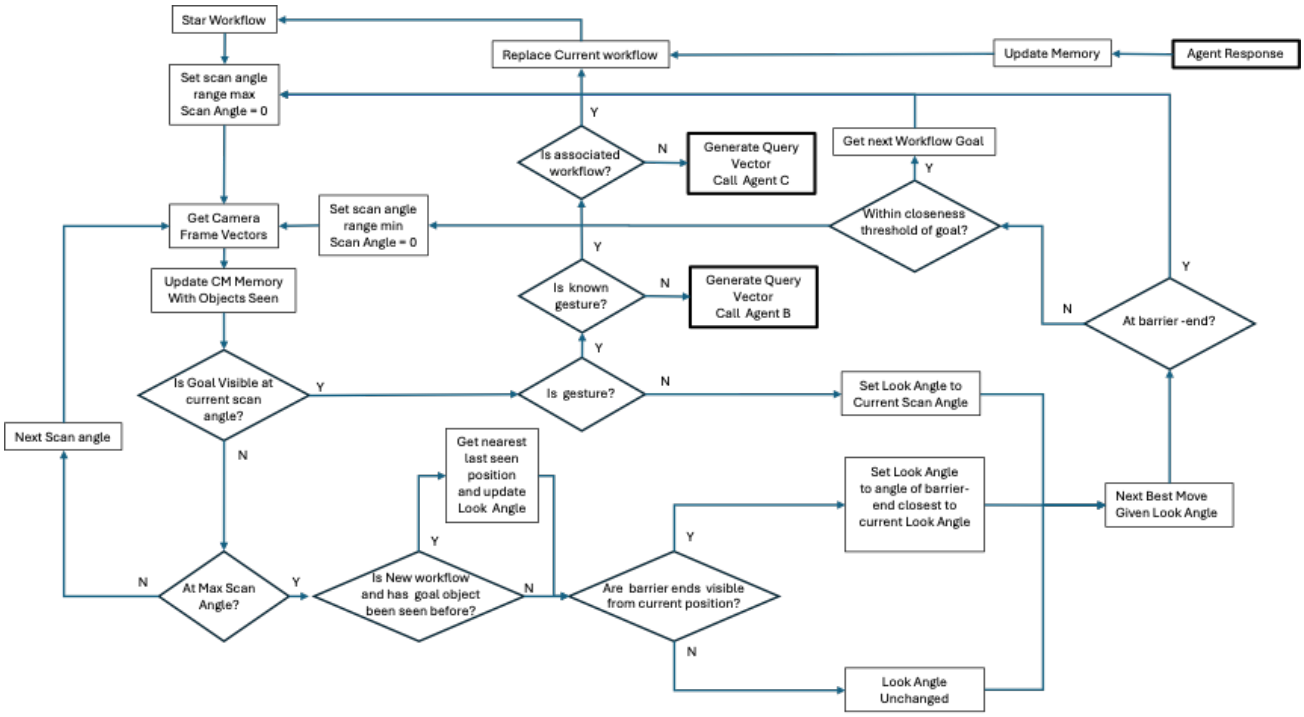

Figure 4. Logical Sequence of Agent A operations.

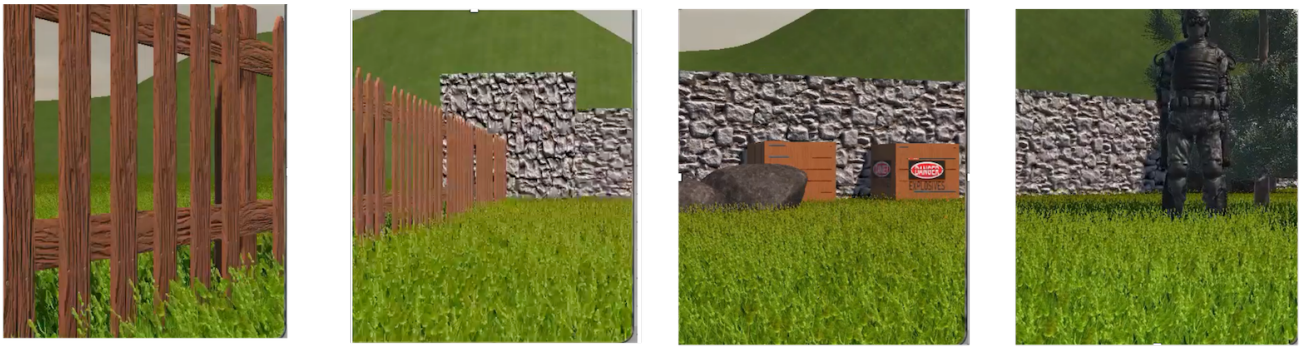

Figure 5. Example FRC views at the different scan angles from the initial position of Agent A; the human is visible at a scan angle of +40 degrees.

that will instruct Agent A as to what its next goal or sequence of goals it needs to perform. This requires the agent to control its scan angle to keep the human in its FOV and then to determine the next best move that is required to achieve the current goal whilst avoiding obstacles and barriers.

In the demonstration, the initial gesture is recognised as a command to 'follow me' and the corresponding workflow comprises a single FRC goal vector for the Human (i.e., no change to the original workflow is required) and so the objective for Agent A is to move towards the human and to orientate such that the human is kept within the FRC's FOV. As both the human and Agent A are moving, this usually means that the Human moves out of the FRC's FOV and this requires the processing cycle to instruct Agent A to perform a limited scan to re-locate the Human and then again to make the next best move in the direction of the human that avoids any barriers or other obstacles. These steps are shown in the video.

If the human cannot be detected at any of the limited scan angles (+/- 30-degrees in the demonstration), then the assumption is that they have moved behind a barrier or passed through a gap in a barrier. In the

demonstration Agent A is provided with the location of each barrier-end and if the human or goal object is not detected then the agent re-plans a course that takes it to the barrier-end that is closest in angle to the angle at which the goal object was last in the FRC's FOV. As we show in Sub-section 5 this is achieved using a simple VSA vector operation. This action continues until Agent A reaches the barrier-end or the goal object is again seen.

If Agent A reaches a barrier-end, then to give the best chance that the goal object will be detected again, the agent increases the scan angle range back to the maximum scan angle. In the demonstration video the logical flow described so far results in Agent A following the human to the position illustrated in Figure 6 where the human encounters a problem that they require the agent to help solve.

3.1.3 Problem Solving by Exploiting LLMs

To illustrate the cognitive process of problem-solving Agent A is required to solve a problem encountered by the human. In the demonstration, the human performs a pointing gesture, which indicates to Agent A that the problem is the pile of rocks (i.e., the nearest object of interest). Whilst Agent A recognises this as a known object vector, it does not have any associated workflow that can be implemented to solve the problem. Agent A must invoke the help of an external agent (Agent C) which has access to knowledge from external sources, such as Large Language Models (LLMs). We explain how Agent A can construct a VSA vector to query Agent C, who is then able to generate and return a new plan to solve this previously unseen problem. Agent A is then able to follow this new plan to solve the problem.

Encoding Messages In order to communicate with the Agent C using VSA, we need to be able to encode text into vectors and vice versa. This is because the LLMs use natural language as both input and output, and they are unable to deal with our vectors directly. In Section 2.3.2 we explained how the different exemplar vectors include a semantic description vectors that are obtained from the BERT vector embedding model projected into binary vectors. With encodings produced using this method, words with similar meanings will tend to have similar encodings. As a result, even if the memory that is being used for decoding does not contain one of the words used in the message, it will most likely match another word with a very similar meaning. For instance, in the demonstration, one of the memories does not contain the more specific word *dynamite*. Despite that, it decodes the noisy vector representing dynamite as the more generic word *explosive*. This makes it possible to preserve most of the meaning of a message even if the receiver does not know the vectors for all the words used by the sender. Agent A therefore only needs to hold in memory the semantic vectors for each object and a small set of word vectors used to define the problem and understand the steps in the plan it receives.

Once the vectors representing each word have been defined, they can be combined to form sentences, much in the way actual words can be combined. Of course, one way to create sentences using vectors would be to append the vector representing each word to each other in order. However, VSA allows us to bundle the vectors, making it possible to compress this information and significantly reducing the need for communications bandwidth.

Prompting the LLM Agent C decodes the message received from Agent A, which gives it information about the problem and the potentially useful known objects. Using this, Agent C constructs a prompt that asks the LLM to generate a plan as a graph of connected nodes. In the resulting graph, any path between the nodes labelled *start* and *end* should go through the steps required to accomplish the given objective, using the appropriate known objects where needed. In the prompt, we ask the LLM to define the new plan using the graphviz[7] language. This allows the LLM to define different approaches based on the available objects.

The resulting plan for the demonstration suggested two possible paths of action: (i) removing a pile of rocks using a crowbar, or (ii) blowing it up using explosives and a detonator. Llama3-8b[8] was the LLM used to generate this plan, but we have also successfully tested Llama2-7b.[9] Once the LLM has generated the new plan, the graph defining this plan is encoded and bundled into a single vector, which is returned to Agent A who can then choose one of the paths of action suggested by the LLM based on the current situation.

In the demonstration Agent A performs a sequence of actions that results in seeking out the location of the different objects (Detonator, Explosives Crate, Crowbar) required to solve the problem and to report these

locations and the shortest routes to each object. The following section describes how this was achieved and the planning and decision making required.

3.1.4 Planning and Decision-Making

Fundamental to our capability to perform cognitive processing using VSA is the concept of a Cognitive Map Memory. To demonstrate the cognitive processes of Planning and Decision-Making, Agent A must construct and then utilises cognitive maps to plan a course of action that enables the agent to locate the objects that are required to achieve the current workflow goals. The following sub-sections describe how the maps are constructed and then how they are used in the planning and decision-making process.

Cognitive Map Construction As was explained in the companion paper,[1] cognitive maps are vector representations of the 'real world' that represent relationships between different things. This can for example be the positional relationship between different objects using vectors that bind an object description vector with its position vector. It could also be relationship between entities in a knowledge graph. These maps can be constructed through a learning process (Cognitive Map Learners, CML) such as that described in the literature[10,11] or, as in this case, they can be engineered for specific tasks.

In the demonstration system, as Agent A moves through the simulated environment it serendipitously observes different objects in its FRC (e.g., rocks, trees, objects of interest). At each location the Agent constructs a new vector (ZO) by bundling the FCR vectors for each object observed at that location, and a second vector (Zoa) that bundles the observed FCR vectors bound to the angle at which they were observed. These vectors are then each bound to the position vector for the current location (Zp) e.g., ZO * Zp, and the resulting vectors are added to the agent's memory. Again, since the agent memory is a self-organising hypercube, these vectors can be used to generate cognitive maps for the different object types that have been seen as discussed below.

Planning Using Cognitive Maps As explained in Section 5 the vector processing required to extract information from a cognitive map is simple to implement. A vector query that binds the vector representing an object of interest to the current position vector of the agent (or any other reference location) can be used to determine the locations where Agent A has observed objects of this type. This is further explained in Section 5. The Hamming similarity is a measure of distance from the reference location. This is illustrated in Figure 6. The top row shows a map of the simulated world with the location of Agent A and the Human indicated. Also shown is a spatial cognitive map of the grid squares that Agent A can view from its current location. The remaining cognitive maps illustrate the locations where Agent A has observed each of the four objects during its journey. The colour coding represents the measure of Hamming similarity and the dark colouring shows the locations that have not been visited.

Figure 4 includes the logical flow for the section of the simulation that makes use of these cognitive maps and the video shows the corresponding movements of Agent A. When Agent A adds the workflow proposed by Agent C to its memory, Agent A is required to locate each of the objects that can be used to solve the problem of the pile of rocks obstacle. Since this is a new workflow the scan angle range is increased to the maximum of +/-180 degrees and the agent performs a scan for any of the required objects (i.e., Explosives Crate, Detonator, Crowbar). Since none of these objects is visible from Agent A's current location the next step is to query the agent memory for the nearest location at which any of the objects were observed. The logic being that if the agent can navigate to this location, then the object should be visible, and Agent A would then be able to navigate directly to the object to get its actual location. The nearest seen location therefore becomes the current goal for Agent A to navigate to and if it cannot be 'seen' then Agent A navigates to the nearest barrier-end whose angle from the current location is closest to the angle of the current goal. In this case the nearest location at which any of the objects were seen is that for the detonator plunger at location(19,17) and Agent A moves towards the gap in the barrier. On passing through the gap the Detonator becomes visible and so Agent A can now navigate directly to it without having to get to the last seen position. Other planning strategies can be implemented, including the equivalent of an A* planning strategy.[1]

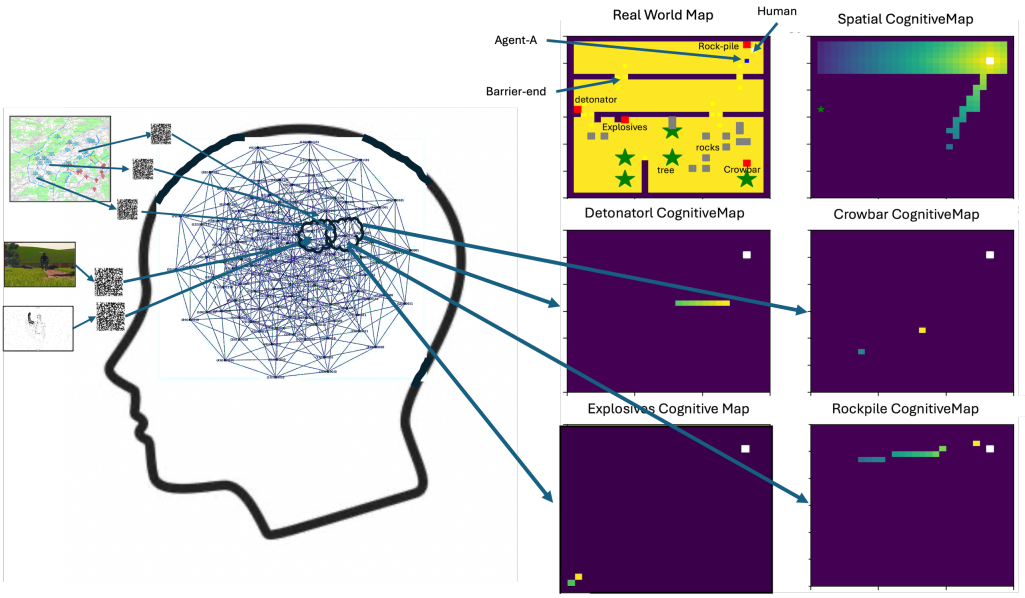

Figure 6. Vectors which are represented in the figure as 'semantic QR codes' are stored in the agents self-organising hypercube cognitive memory. The spatial cognitive map shows the field of view of the agent from its current location. Also shown are cognitive maps for four different objects derived from the agent memory. The colour coding is a measure of Hamming similarity and represents the distance from the agents current location to any location at which the object was seen or is dark if the location has not been visited.

Once the location of the detonator is established the same logical flow steps are repeated for the remaining objects and when all object locations have been determined. Finally, Agent A sets its goal object to be the pile of rocks and navigates to this location using the same planning strategy.

3.1.5 Learning Using Local Knowledge Exchange

In some situations the observed gesture or object vector is unknown. In these situations Agent A can perform an action that requests additional information from some other agent (e.g., Agent B). In this case Agent A constructs a request vector that comprises the observed gesture or object vector with a simple semantic request vector (e.g., 'what is this'). This vector is broadcast and any agents on the network that hears the request can potentially respond. Any agent such as Agent B that recognises this as a request for information compares the request vector with its exemplar object and gesture vectors to determine if there is a match. If a match exists then the corresponding exemplar vector is re-broadcast and the requesting agent can add this vector to its memory. Since the response is a broadcast message any agent that receives the request can update its own memory of the exemplar vector. It is therefore possible to distribute learning from multiple agents by simply sharing lightweight vector descriptors.

In the demonstration the capability to exchange local knowledge among VSA-based agents at the edge was demonstrated for the case where a gesture vector, previously unseen by Agent A, was known by Agent B to have the same meaning as the action 'Follow'. In this case Agent A was able to identify that there were two possible exemplar vectors for the same workflow action sequence. The process required the broadcast of a 2kbit message by Agent A and a 2kbit response message from Agent B.

4. VECTOR REPRESENTATIONS

In the case of the human-robot teaming demonstration, vectors are required to represent a number of different objects, workflows, angles, positions and cognitive maps. The vectors are stored in no particular order since they are self referencing and hence self-organising as described earlier.

The following vectors are generated and stored:

4.1 Exemplar Object and Gesture Vectors

Each object in the environment is described using three component vectors as discussed earlier in 2.3.2:

1. A semantic description vector (Z_Objects_desc) which describes what the object represents using a semantic representation that is derived from a common vector embedding model (e.g., BERT, Llama) and mapped into the binary hypervector representation used in the demonstration.
2. An FCR exemplar vector (Z_Objects_cam) which is a vector representing the expected output from the FCR camera when the object is in the FOV of the FCR.
3. A workflow vector (Z_Objects_wf) is a vector used to represent the actions Agent A can perform if it comes within a specified closeness range of the object. In the demonstration for most objects this was a null-vector but this would typically include a sequence of actions (e.g., 'unlock', 'open').

To construct an object-vector these three vectors are each bound to a corresponding role vector (Z_Role_description, Z_Role_camera, Z_Role_workflow). Each of these role vectors are random hypervectors and so each bundled vector is guaranteed to be quasi-orthogonal to the other vectors(see section III of Reference[12]). These are then bundled to create the required Object vector i.e.,

$$\begin{aligned} Z\_Object\_Vector = Z\_Role\_description * Z\_Objects\_desc \\ + Z\_Role\_camera * Z\_Objects\_cam \\ + Z\_Role\_workflow * Z\_Objects\_wf \end{aligned} \tag{1}$$

Where the '*' operator indicates an XOR binding operation and the '+' operator indicates a bundling operation.

4.2 Workflow Description Vectors

Workflow description vectors are compound vectors that define the sequence of actions that are required for Agent A to perform a specific task. Since Agent A only senses the camera vectors the workflow vectors are comprised of a sequence of camera vectors which the agent uses to determine which is the next goal to which it has to navigate. The workflow vector can therefore be a bundled vector using role-filler vector pairs where the role is a vector indicating the order of the workflow or it can be a bundled vector of cyclically shifted camera vectors which can be recursively unbound to reveal the next goal. The sequence is terminated with a stop vector to indicate the end of the workflow. In the demonstration the cyclic shift method was chosen and so any workflow vector has the following representation:

$$Z\_workflow = [Z\_Objects\_cam[a]] + [Z\_Objects\_cam[b]]^1 + \ldots [Z\_Objects\_cam[n]]^m + [Z\_Stop]^{m+1} \tag{2}$$

Where a vector raised to the power m represents a right cyclic shift by m positions

As each element of the workflow is completed the workflow vector is left cyclically shifted by one position until the Z_Stop vector is encountered and the workflow is complete.

4.3 Angle Vectors

To represent angles, it is necessary to have vectors that can be compared such that the Hamming similarity between any two vectors is a measure of the minimum angular distance between the two vectors (e.g., the angle between 10-degrees and 350-degrees is 20-degrees and not 340 degrees). This is achieved by constructing an array of 'circular' hypervectors which have this property.

In the demonstration Agent A moves over a simulated grid of size 30 × 30 and is assumed to be capable of orienting its camera in steps of 10-degrees relative to the Y-direction which is referenced as 0-degrees. This therefore requires Agent A to store a minimum of 36 circular hypervectors for each 10-degree step form 0-degrees to 350-degrees. These vectors are designated Zc[0] ,Zc[10] . . . , Zc[350].

4.4 Position Vectors

To represent the location of an object or that of Agent A it is also necessary to have vectors such that the Hamming similarity between vectors representing any two locations is a measure of the distance between the locations. This is essentially a positional cognitive map. To achieve this, we use the concept of 'linear', also known as 'level' hypervectors. This is an array of vectors where the Hamming similarity is a measure of the linear distance. To construct a position vector we create two linear hypervector arrays, Z_X and Z_Y that represent the X and Y axis of the position grid and then the required position vector for any grid location(i,j) is computed by binding the corresponding X and Y hypervectors i.e.,

$$Zp[i,j] = Z\_X[i] * Z\_Y[j] \tag{3}$$

Since Agent A can generate a position vector whenever it needs to record its own location, or the location of another object, the agent only needs to store a total of 60 linear hypervectors.

4.5 Cognitive Map Vectors

As explained in Section 3.1.4 Agent A constructs cognitive maps that are used to record objects that are observed from each location that Agent A visits as it moves around the simulated environment. These vectors are constructed using combinations of the observed FCR camera vectors and then look angle of Agent A when the objects are in the FOV of the FCR. As an example of the structure of these vectors we use the FCR FOV images illustrated in Figure 5 when the agent is at location(1,1). In this case the vectors generated are as follows:

$$\begin{aligned} Zo = Z\_Objects\_cam[fence] + Z\_Objects\_cam[barrier\_end] + Z\_Objects\_cam[rock] \\ + Z\_Objects\_cam[wooden\_crate] + Z\_Objects\_cam[explosives\_crate] + Z\_Objects\_cam[tree] \end{aligned} \tag{4}$$

$$\begin{aligned} Zoa\_shift = [Z\_Objects\_cam[fence] * Zc[340]] + \\ [[Z\_Objects\_cam[fence] + Z\_Objects\_cam[barrier\_end]] * Zc[0]]^1 \\ + [[Z\_Objects\_cam[rock] + Z\_Objects\_cam[wooden\_crate] + Z\_Objects\_cam[explosives\_crate]] * Zc[20]]^2 \\ + [Z\_Objects\_cam[tree] * Zc[40]]^3 \end{aligned} \tag{5}$$

The Z_o vectors from each frame are bundled to produce a new vector ZO which represents of all the objects observed from a specific location. The vector ZO is bound to the current agent position Z_p. This makes it possible to easily find from which locations any object has been seen and is used in the planning phase of the OODA loop to locate the nearest location of an object as described in the next section.

Zoa_shift is more specific, as it also encodes at which angle each object was seen. This vector is used to determine the angle at which the agent must travel as it moves towards a goal object.

5. VECTOR 'NEAR-MEMORY' AND 'IN-MEMORY' PROCESSING

The concept underpinning 'Near-Memory' and 'In-Memory' processing is that most operations are performed directly on the data memory, rather than having to be transferred to central processor unit (CPU) registers first. VSA processing has previously been demonstrated to be compatible with potential next generation 'Near-Memory' and 'In-Memory' neuromorphic processing technology including phase change memory (PCM) processors[2] and spiking neural network processors.[13] These types of processing device have the capability to perform massively parallel vector comparison operations at extremely low energy and therefore offer great future potential for VSA processing.

The basic concept is illustrated in Figure 7. Existing vectors in the memory can be triggered to become active in any processing cycle and/or new vectors can be simultaneously presented as additional input. The triggered vectors and any new vector are automatically bundled to generate an output vector. The output vector can be further processed (e.g., bound to another vector in a 'Near-Memory' processing operation). The resulting vector is then compared to all the memory vectors and the best matching vector(s) generate triggers that are recursively

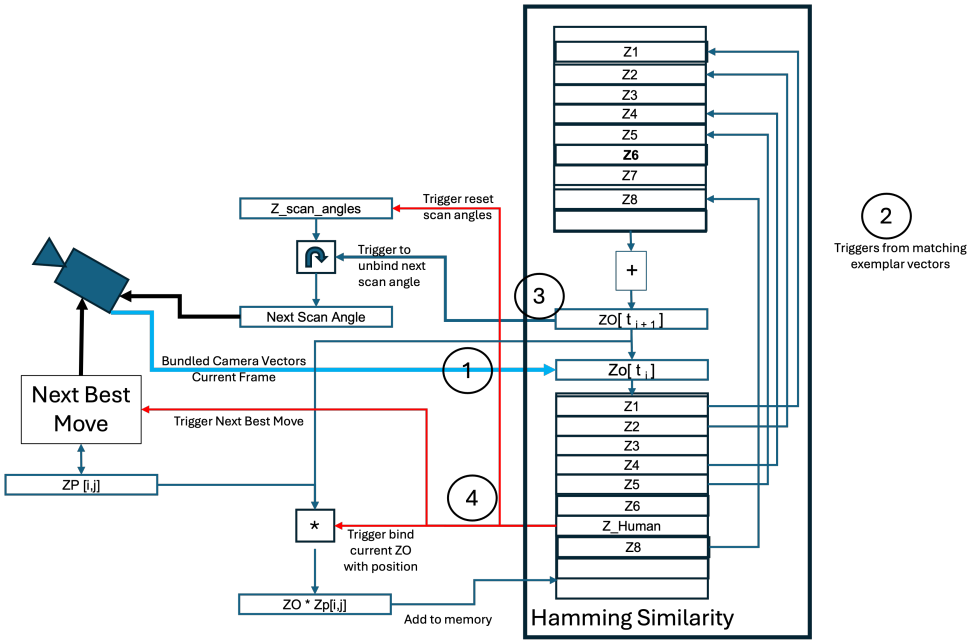

Figure 7. Example 'Near-Memory' and 'In-Memory' Processing to control the agent camera scan the next best move and to create the ZO cognitive memory vector for the current agent position.

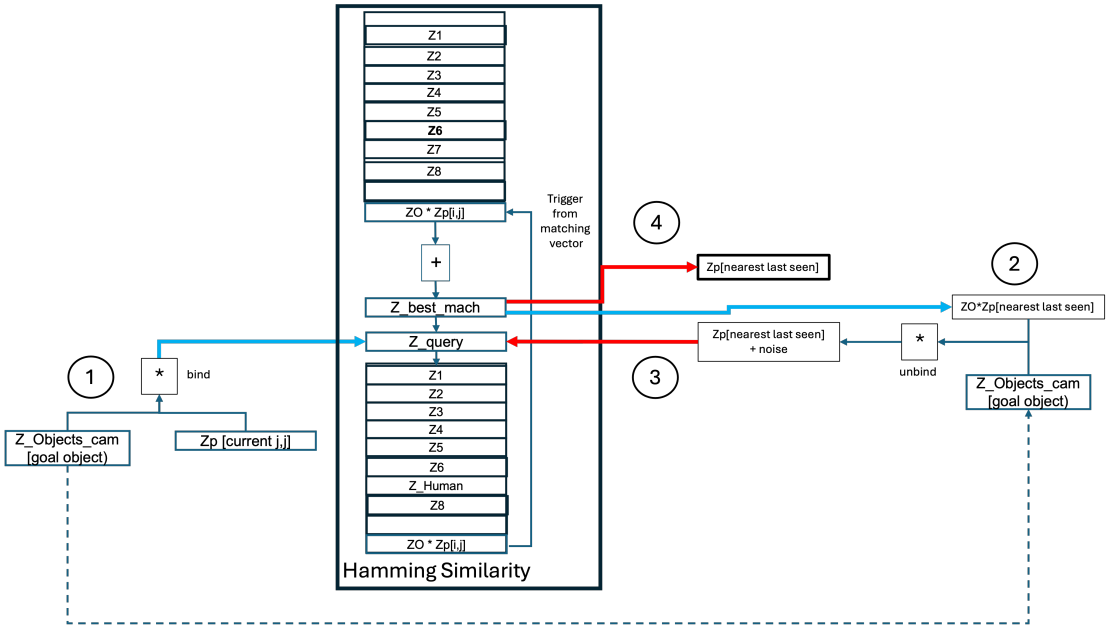

Figure 8. Querying memory for nearest last seen position of a goal object from the current agent position, using Best_Match_Vector; the numbering reflects the sequence of the vector processing.

fed back to generate new vectors and can also be used to trigger other external memories or as triggers for eternal actions (e.g., instructions for Agent A to orientate to a new direction or to move to a new location).

To assist in understanding the process we describe the creation of the cognitive map vector ZO (i.e., bundled vector of all objects seen by Agent A at a given location) as a typical example:

The FCR camera processes a single frame at the agent's current look angle, and for all objects that are in the FOV the camera object vectors are bundled into a single vector (as example of a 'Near-Memory' process).

This bundled vector is fed into the 'In Memory' process. The memory does not contain all possible camera vectors but only the exemplar vectors for each object type as explained in Section 2.4. Hence these vectors will not be an exact match, but by using an appropriate threshold the Hamming similarity will be sufficient to activate the corresponding exemplar vectors. The resulting output from the 'In Memory' process will therefore be a bundled vector of the corresponding exemplar vectors (i.e., Zo for that look angle). At this point, unless a human has been detected (see below), the agent is instructed to orientate to the next look angle and generate the next bundled frame vector. The activated exemplar vectors each produce a trigger which, following a processing delay corresponding to the time to generate the next frame vector, reactivates these vectors together with any exemplar vectors for new objects observed in the next frame. This results in a bundled vector of all exemplar object camera vectors observed at the current location during the current scan. This process continues until the maximum scan angle is reached or a Human is detected. If a human is in the FOV of any frame, then the exemplar vector for human will be activated and the resulting trigger can be used to terminate the cycle. This results in the cognitive map vector ZO being bound to the current position vector and added to the agents memory (i.e., ZO*Zp[i,j]).

In addition to the vector bundling example given above, the memory can also be queried for vectors that best match a particular query. One example that illustrates the power of the VSA approach is the vector solution to finding the nearest location to the agent's current location where a particular goal object is seen as part of the planning activity. This can be simply achieved by constructing a query vector:

$$Z\_query = Z\_Objects\_cam[goal\_object] * Zp[Current\_agent\_position] \tag{6}$$

This query vector is issued to the memory and whilst a number of the ZO_CM vectors will match the one with the highest Hamming similarity will be the vector that contains the goal object that is closest in distance to the agents current location. This vector can in turn be unbound using the Z_Objects_cam[goal_object] vector to obtain the position vector of this location. Whilst this vector is the actual required vector plus noise a clean version can be obtained by passing this vector back through the memory. The process is illustrated in Figure 8.

All of the other logical operations described in Section 3.1 are performed using similar 'Near-Memory' and 'In-Memory' operations. In the demonstration all of the processing is performed using 2kbit hypervectors and the total memory requirements for all of the vectors used was $\sim$ 500k bits ($\sim$ 62.5kB).

6. CONCLUSION & FUTURE WORK

In this paper we have demonstrated that the VSA paradigm offers robust solutions for generation-after-next AI deployment at the network edge. Specifically, we have shown how VSA effectively models and integrates the cognitive processes required to perform intelligence, surveillance, and reconnaissance (ISR). The demonstration described integrates functions across the observe, orient, decide and act (OODA) loop and we have shown the following:

1. Sensed observations from both event-based and frame-based cameras can be processed into VSA vector spaces allowing recognition and determination of appropriate courses of action. (Section 2)

2. Cognitive maps provide an effective mechanism to integrate VSA-based cognitive processing functions into a uniform set of processes able to implement the OODA loop. (Section 3.1.4)

3. The VSA-based approach is compatible with current state-of-the-art LLM approaches, allowing interoperability between these forms of AI. (Section 3.1.3)

4. Multiple agents at the network edge can exchange information using very low bandwidth and learn from each other about previously unseen objects and appropriate actions to take. (Section 3.1.5)

5. The VSA processing can be performed just using vector operations demonstrating 'Near-Memory' and 'In-Memory' processing can be performed across the full OODA loop. In the demonstration this was achieved using 2kbit hypervectors requiring just 500k bits of cognitive memory. (Section 5)

Whilst the demonstration was focused on an human-agent teaming scenario this was just an illustration of the potential capabilities of VSA processing as an integration technology for a wide range of AI technologies including symbolic and sub-symbolic approaches. We propose to further investigate these capabilities in other edge applications with a specific focus on emerging neuromorphic processing capabilities[2,13] that offer the potential for ultra-low power cognitive processing.

In summary: VSA has been shown to posses a unique set of attributes making it ideal to provide a key part of the Defence approach to exploiting distributed data and algorithms to provide actionable insight in a decentralised manner at the edge, with contested communications and data, and limited compute and power.

6.1 Acknowledgments & Copyright

This work was funded by Dstl under project number U129, contract number DSTL0000022661.

REFERENCES

[1] Bent, G., Davies, C., Vilamala, M. R., Li, Y., Preece, A., Sola, A. V., Caterina, G. D., Kirkland, P., Pearson, G., and Tutcher, B., "The transformative potential of vector symbolic architecture for cognitive processing at the network edge," in [*Artificial Intelligence for Security and Defence Applications II*], SPIE (2024).

[2] 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, SPIE (2021).

[3] Davies, C., Meek, S., Hawkins, P., Tutcher, B., Preece, A., and Bent, G., "Vector symbolic architecture for semantic data discovery in coalitions," in [*Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications VI*], **13051**, SPIE (2024).

[4] Devlin, J., Chang, M., Lee, K., and Toutanova, K., "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR* **abs/1810.04805** (2018).

[5] Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., et al., "Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection," *arXiv preprint* **arXiv:2303.05499** (2023).

[6] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint* **arXiv:2010.11929** (2020).

[7] Gansner, E. R. and North, S. C., "An open graph visualization system and its applications to software engineering," *Software: Practice and Experience* **30**(11), 1203–1233 (2000).

[8] Meta, "Introducing Meta Llama 3: The most capable openly available LLM to date." https://ai.meta.com/blog/meta-llama-3/ (2024).

[9] Meta, "Llama 2: open source, free for research and commercial use." https://llama.meta.com/llama2/ (2024).

[10] Bent, G., Davies, C., Vilamala, M. R., Li, Y., Preece, A., Sola, A. V., Caterina, G. D., and Kirkland, P., "Opportunity white paper (Serapis Lot 6 U129): The transformative potential of vector symbolic architecture for cognitive processing at the network edge," tech. rep., Dstl (2024).

[11] Stöckl, C., Yang, Y., and Maass, W., "Local prediction-learning in high-dimensional spaces enables neural networks to plan," *Nature Communications* **15**(2344) (2024).

[12] Kleyko, D., Davies, M., Frady, E., and Kanerva, P., "Vector symbolic architectures as a computing framework for nanoscale hardware. arxiv preprint arxiv...(2021)," *URL https://arxiv. org/abs/2106.05268* (2021).

[13] Bent, G., Simpkin, C., Li, Y., and Preece, A., "Energy efficient spiking neural network neuromorphic processing to enable decentralised service workflow composition in support of multi-domain operations," in [*Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications IV*], **12113**, 479–499, SPIE (2022).