APPENDIX

A. Learnable Parameters

Learnable parameters are adjusted during memory operations through a reinforcement-based continual learning process. The reinforcement comes from data access patterns, search success/failure, and query patterns. There are five sets of learnable parameters:

1) **NoK Spatial Graph**: This is an $n \times n$ matrix denoting the spatial connectivity of all n neurons in the NoK. Changing the values of this matrix results in graph structure alteration leading to a change in NoK navigation and search behaviour. Although this is theoretically represented as a matrix, this can be stored as an adjacency list or in a sparse format to save space.
2) **NoK Temporal Graph**: This is another $n \times n$ matrix capturing the temporal linkage between neurons in the graph. Changes to this matrix indicate an alternation of temporal knowledge stored in the memory. Although this is theoretically represented as a matrix, this can also be stored as an adjacency list or in a sparse format to save space.
3) **Memory Strength**: This is a n_d dimensional vector where n_d is the number of data neurons in the NoK. The i^{th} element in this vector denotes the memory strength of the specific data neuron. A higher value indicates more allocated storage space and stronger perceived importance of the data.
4) **GIST Composition**: This is a n_g dimensional vector of pairs where n_g is the total number of GIST neurons in the NoK. Each element consists of (GIST-Val, GIST-Confidence). GIST-Val is the high-level information of the surrounding data and GIST-Confidence is the projected confidence of this information.
5) **CoT Model**: This is a dictionary, $D = \{A_1 : \{A_1^1 : w_1^1, A_1^2 : w_1^2, ...\}, A_2 : \{A_2^1 : w_2^1, A_2^2 : w_2^2, ...\}, ..., A_c : \{A_c^1 : w_c^1, A_c^2 : w_c^2, A_c^3 : w_c^3, ...\}\}$. Here, $A_1, A_2, ...$ are observed access chains, each associated with another dictionary containing all the prospective next accesses accompanied by weights representing the likelihood of such access. For example, if the neurons in the access chain A_1 have been retrieved then the next access should be the neuron A_1^1 with a likelihood of w_c^1.

B. Hyperparameters

We propose several hyperparameters to adjust and tune the behaviour of the proposed memory system in terms of (1) how the learnable parameters are adjusted and (2) how each memory operation is performed. For this specific implement we only use two levels of cues (L1, L2), so $n = 2$. The full list of these proposed memory hyperparameters is as follows:

1) **Memory strength modulation factor** (δ_1): Used to control the data neuron strength increment step-size. This hyperparameter is used during store (Algo. 1) and retrieve (Algo. 6) operations.
2) **Memory decay rate** (δ_2): Controls the rate at which data neurons loose memory strength. This hyperparameter is used during the retention operation (Algo. 10).
3) **Maximum Memory Strength** (δ_3): The maximum data neuron strength allowed. This hyperparameter is used during the store (Algo. 1) and retrieve (Algo. 6) operations.
4) **Association strengthening step-size** (η_1): Step-size for increasing spatio-temporal edge strengths. This hyperparameter is used during the store (Algo. 1) and retrieve (Algo. 6) operations.
5) **Association weight decay rate** (η_2): Controls the rate of decay of spatio-temporal edge strengths. This hyperparameter is used during the retention operation (Algo. 10).
6) **Association pull-up hastiness** (η_3): Controls the momentum with which neuron connectivities are re-adjusted based on data access and search patterns. This hyperparameter is used during the store (Algo. 1) and retrieve (Algo. 6) operations.
7) **Cue neuron matching metric for Store**(Λ_1): This is a list of floating-point values in the range [0,100]. Each element corresponds to a particular Cue level and denotes the minimum threshold beyond which a feature match during store operation is considered to be a success. $\Lambda_1 = \{\lambda_{11}, \lambda_{12}, ...\lambda_{1l}\}$, for a system with l different cue levels. This hyperparameter is used during the store (Algo. 1) operation.
8) **Cue neuron matching metric for Load**(Λ_2): This is a list of floating-point values in the range [0,100]. Each element corresponds to a particular Cue level and denotes the minimum threshold beyond which a feature match during Retrieve/Retrieve_Delta operations is considered to be a success. $\Lambda_2 = \{\lambda_{21}, \lambda_{22}, ...\lambda_{2l}\}$, for a system with l different cue levels. This hyperparameter is used during the Retrieve (Algo. 6) the Retrieve_delta (Algo. 13) operations.
9) **Degree of allowed impreciseness**(φ): Minimum value of data neuron strength. Setting $\varphi = 0$ will enable data neuron deletion. This hyperparameter is used during the retention operation (Algo. 10).
10) **Initial association weight** (ε_1): This is the initial value assigned to a newly formed edge. This hyperparameter is used during the store (Algo. 1) and retrieve (Algo. 6) operations.
11) **Minimum association weight** (ε_2): This is the minimum allowed edge weight. Setting $\varepsilon_2 = 0$, enables edge deletion. This hyperparameter is used during the retention operation (Algo. 10).
12) **Store Effort Limit** (π_1): Limits the amount of effort/energy utilized during the search phase of a store operation (Algo. 1). -1 indicates an infinite limit.
13) **Retrieve Effort Limit** (π_2): Limits the amount of effort/energy utilized during the search phase of a load operation (Algo. 6). -1 indicates an infinite limit. This hyperparameter is used during the Retrieve (Algo. 6) the Retrieve_delta (Algo. 13) operations.
14) **Locality Crossover** (ω): A boolean flag that enables rigid locality formation. This hyperparameter is used during the store (Algo. 1), the retrieve (Algo. 6), and the retrieve_delta (Algo. 13) operations.
15) **Frequency of retention procedure** (ξ): Controls the frequency at which the retention operation is invoked by the system.

16) **GIST Confidence Threshold** (γ): The GIST confidence beyond which it is considered correct by the proposed memory system. This hyperparameter is used during the store (Algo. 1), the retrieve (Algo. 6), and the retrieve_delta (Algo. 13) operations.
17) **Temporal Fusion** (χ): This is a boolean flag that enables the NoK search process to utilize both the spatial edges and the temporal edges (if $\chi = 1$). If $\chi = 0$, then only the spatial connections are used for the search process. This hyperparameter is used during the store (Algo. 1) operation.
18) **Spread Limit** (θ_1): Controls the extent of GIST information spread during the GIST extraction (Algo. 12).
19) **Spread Decay** (θ_2): Controls the intensity of GIST information spread during the GIST extraction (Algo. 12).
20) **Compression techniques**: The data compression technique(s) utilized to adjust data neuron size based on their perceived importance. This hyperparameter is used during the retention operation (Algo. 10). We use JPEG compression [55], [56], [57] for all our experiments.
21) **Chain of Thought Length** (μ_1): Indicates the number of past accesses that are considered while making a decision (using the memory chain model) about the potential next access candidate. This hyperparameter is used during the retrieve (Algo. 6) operation.
22) **Chain of Thought Model Confidence** (μ_2): Indicates the minimum observed frequency of an access chain (as recorded in the memory chain model) before it is utilized for making a prediction. This hyperparameter is used during the retrieve (Algo. 6) operation.
23) **Chain of Thought Model Size** (μ_3): The maximum number of memory access sequences that are stored as a part of the memory chain model. This hyperparameter is used during the retrieve (Algo. 6) operation.
24) **Chain of Thought Blind Fetch** (μ_4): A Boolean flag indicating whether to blindly fetch (without cue checking) the predicted neuron while utilizing the memory chain model. This hyperparameter is used during the retrieve (Algo. 6) operation.
25) **NoK Update Flag** (μ_5): A Boolean flag indicating whether to update the NoK or not based on feedback. This hyperparameter is used during the retrieve delta (Algo. 13) operation.

C. Operation Algorithms

1) Store: Storing data is one of the most important operations in any memory framework. For our proposed framework, incoming data must be organized inside the neural memory network (NoK) based on spatial-temporal relevance with respect to nearby data units. From a high level, based on the incoming data features and cues: (1) the entry point into the NoK is selected, (2) based on a traversal algorithm, the location where the incoming data should ideally reside is decided, and (3) then the NoK is modified to reflect the new insertion and other efficiency related adjustments (see fig. 7).

Algo. 1, Algo. 2, Algo. 3, Algo. 4, Algo. 5, and Algo. 10 together provide is a more detailed description of the store operation in the proposed memory. In Algo. 1, the inputs are: (1) MEM, the proposed memory instance, (2) D, the new data to be inserted, (3) C, a set of cues associated with D, and (4) HP, the hyperparameters of the proposed memory instance. First, the system checks if enough space exists to insert D and in an event of space shortage, the retention operation is invoked to make extra space (line 2-3). The first data unit insertion follows a specific pattern as shown in Fig. 11 (line 4-5). Next the entry point (Loc_C_n), a level n cue neuron is selected to serve as the starting point of the NoK traversal process. In line 7, the SEARCH (Algo. 2) algorithm is invoked to find the most ideal position (given all constraints) for inserting D. Based on the outcome of the SEARCH sub-operation, the new data insertion and NoK modification take place (line 9).

Inside the SEARCH sub-operation (Algo. 2) lines 2-7, the search effort limit and the cue matching threshold are selected based on the type of the operation (store/retrieve). This is followed by the initialization of a few variables and a limited breadth-first search traversal (lines 13-35) of the NoK starting from ep, the entry point. This traversal may terminate early based on the predefined hyperparameters (lines 14-15). The check in line 18 limits the search within a specific locality with locality boundaries determined based on the presence of level-n cue neurons. For each level-1 cue neuron encountered (check at line 20), the associated GIST neuron is checked to determine whether to perform a feature comparison or not (line 21). Only if GIST_CH (Algo. 3) returns true (T), the comparison between the provided search cue (C_1) and the cue neuron under investigation (Neu) is performed (lines 23). If a sufficient feature match is observed during the previously mentioned comparison, then the access path (AP) in the NoK to Neu is returned along with a flag 1 to indicate a success. The best candidate neuron is tacked in lines 27-29 and the access path to it (and a flag value of 0) is returned at the line 36 if a good match was not found during this search. Lines 30-35 enqueues adjacent neurons of visited neurons (standard steps in any breadth-first search).

Inside the GIST_CH sub-operation (Algo. 3), first the GIST neuron (GN) associated with the cue neuron (Neu) is identified (line 2). A True verdict is automatically returned if: (1) GN is empty, (2) the search GIST value ($GIST$) is invalid (-1), or (3) the confidence value associated with GN ($Conf$) is less than the hyperparameter γ. Otherwise, the result is determined based on the comparison between $GIST$ and $GN \rightarrow val$ (value of GIST neuron GN).

Inside the LEARN_STORE (Algo. 4) sub-operation the NoK is modified based on the previous search outcomes and there are four different scenarios that can arise. If $Flag\_C_n$ and $Flag\_C_1$ are both 1, then it implies that good matches were found for the level-n cue and also for the level-1 cue (lines 2-7). In that scenario, the memory strength of the data neuron (DN) associated with the level-1 cue neuron at the end of the access path ($AP[-1]$) is strengthened (line 3). The accessibility of the cue neuron $AP[-1]$ is increased (line 4) using Algo. 5. The level-1 cue neuron inserted ($MEM \rightarrow last\_ins$) and $AP[-1]$ are temporally connected (line 5). If $HP \rightarrow \chi$ is 1, then a spatial connection is also created between $MEM \rightarrow last\_ins$ and $AP[-1]$ (lines 6 and 7). If $Flag\_C_n = 0$ and $Flag\_C_1 = 1$, then it implies that a

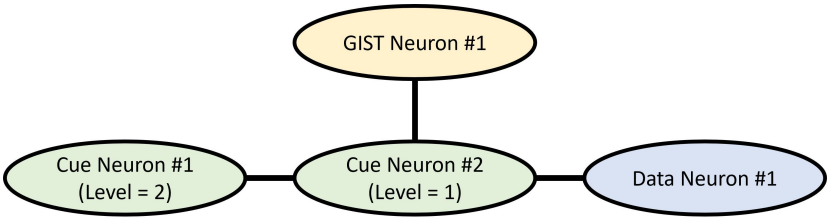

Fig. 11: Initialization scheme for the proposed memory framework.

Algorithm 1 STORE

1: **procedure** STORE(MEM, D, C, HP)
2: **while** $size(D) > remaining\_space(MEM)$ **do**
3: $RETENTION(MEM, HP)$
4: **if** $neuron\_Count(MEM) = 0$ **then**
5: $Initial\_Insertion(MEM, D, C, HP)$
6: **else**
7: $[Loc\_C_n, Flag\_C_n] = find\_Entry\_Point(MEM, C, HP)$
8: $[AP, Flag\_C_1] = SEARCH(MEM, C, HP, Loc\_C_n, -1, \text{“}S\text{”})$
9: $LEARN\_STORE(MEM, C, HP, D, Flag\_C_n, AP, Flag\_C_1)$

Algorithm 2 SEARCH

1: **procedure** SEARCH($MEM, C, HP, ep, GIST, opFlag$)
2: **if** $opFlag =$ “S” **then**
3: $limit = HP \rightarrow \pi_1$
4: $matchThresh = HP \rightarrow \lambda_{11}$
5: **else if** $opFlag =$ “L” **then**
6: $limit = HP \rightarrow \pi_2$
7: $matchThresh = HP \rightarrow \lambda_{21}$
8: $neuronQueue = [(ep, -1)]$
9: $bestCandidate\_Sim = -1$
10: $bestCandidate = \phi$
11: $visited = []$
12: $traversalMotion = []$
13: **while** $len(neuronQueue)! = 0$ **do**
14: **if** $limit! = -1$ & $len(visited) >= limit$ **then**
15: **break**
16: $(Neu, parentNeuron) = neuronQueue.dequeue()$
17: **if** $Neu \notin visited$ **then**
18: **if** $HP \rightarrow \omega = 0 \mid isCue(Neu) = F \mid level(Neu)! = n$ **then**
19: $traversalMotion.append((parentNeuron, Neu))$
20: **if** $isCue(Neu) = T$ & $level(Neu) = 1$ **then**
21: **if** $GIST\_CH(Neu, GIST, MEM, HP) = T$ **then**
22: $C_1 = C \rightarrow level\_1\_cue$
23: $sim = compute\_Similarity(Neu, C_1)$
24: **if** $sim > matchThresh$ **then**
25: $AP = compute\_AP(traversalMotion, Neu)$
26: **return** $[AP, 1]$
27: **else if** $sim > bestCandidate\_Sim$ **then**
28: $bestCandidate\_Sim = sim$
29: $bestCandidate = Neu$
30: $visited.append(Neu)$
31: $n = sort\_descending(neighbours(Neu))$
32: $index = 0$
33: **while** $index < len(n)$ **do**
34: $neuronQueue.enqueue(n[index], Neu)$
35: $index++$
36: $AP = compute\_AP(traversalMotion, bestCandidate)$
37: **return** $[AP, 0]$

Algorithm 3 GIST_CH

1: **procedure** GIST_CH($Neu, GIST, MEM, HP$)
2: $GN = find\_associated\_GN(MEM, Neu)$
3: **if** $GN! = \phi \& GIST! = -1 \& GN \rightarrow Conf > HP \rightarrow \gamma$ **then**
4: **if** $GIST = GN \rightarrow val$ **then**
5: **return** $True$
6: **else**
7: **return** $False$
8: **else**
9: **return** $True$

good match was found for the level-1 cue neuron but not for the level-n cue neuron. In this case, a new level-n cue neuron ($newCN$) is instantiated and is associated with $AP[-1]$ (lines 9 and 10). After that, similar steps follow as explained earlier (lines 11-15). If $Flag\_C_n = 1$ and $Flag\_C_1 = 0$, then it implies that a good match was found for the level-n cue neuron but not for the level-1 cue neuron. In this case, a new level-1 cue neuron($newCN$), a new data neuron ($newDN$), and a new GIST neuron ($newGN$) are created (lines 17-19). This is followed by new associations as shown in lines 20-25. The accessibility of the cue neuron $AP[-1]$ is increased (line 26) using Algo. 5. If $Flag\_C_n = 0$ and $Flag\_C_1 = 0$, then it implies that good matches were not found for the level-n cue and also for the level-1 cue. Hence in this case, a new level-1 cue neuron($newCN\_l_1$), a new level-n cue neuron($newCN\_l_n$), a new data neuron ($newDN$), and a new GIST neuron ($newGN$) are created (lines 28-31). This is followed by new associations as shown in lines 32-38. The accessibility of the cue neuron $AP[-1]$ is increased (line 39) using Algo. 5.

Inside the INC_ACCESSIBILITY (Algo. 5) sub-operation, the accessibility of the cue neuron $AP[-1]$ is increased by creating a shortcut edge (lines 2-4) or via edge weight strengthening (lines 5 and 6). The description of the RETENTION operation is provided in sec. D.

2) Retrieve: Retrieving a stored data based on a query feature (cue) is a standard requirement for many applications. For the proposed memory, the retrieve operation (see Fig. 7) is performed given a set of search cues (C), and an optional search GIST (G). The basic approach is to first find the entry point into the NoK based on C (or the chain of thoughts model) and then search the NoK for the target data using C and G. Based on the search outcome, the NoK is modified and any found data is returned.

Algo. 6, Algo. 2, Algo. 3, Algo. 7, Algo. 5, Algo. 8, and Algo. 9 together provide a more detailed description of the retrieve operation in the proposed memory. In Algo. 6, the inputs are: (1) MEM, the proposed memory instance, (2) C, a set of search cues, (3) HP, the hyperparameters of the proposed memory instance, and (4) $GIST$ a search GIST value. If no neuron exists in MEM, then an empty set is returned. If the chain of thoughts(CoT) model size, μ_3, is greater than 0, we obtain the optimal search point prediction (P_CN) using Algo. 8. If P_CN is valid and the blind fetch flag is set to 0 (μ_4), then we carry out a search processing in the NoK starting from P_CN (using Algo. 2), followed by NoK readjustment (Algo. 7) and CoT Model update (Algo. 9), in lines 7-11. If P_CN is valid but the blind fetch flag is set to 1 (μ_4), then the data associated with (P_CN) is retrieved without check followed by CoT Model update using Algo. 9 (lines 12-15). If P_CN is invalid, then the entry point into the NoK is selected based on C, a search is carried out from

Algorithm 4 LEARN_STORE

1: **procedure** LEARN_STORE($MEM, C, HP, D, Flag\_C_n, AP, Flag\_C_1$)
2: **if** $Flag\_C_n = 1$ & $Flag\_C_1 = 1$ **then**
3: $strengthen\_Associated\_DN(AP[-1], HP)$
4: $INC\_ACCESSIBILITY(MEM, HP, AP)$
5: $associate\_temp(MEM, MEM \rightarrow last\_ins, AP[-1], HP)$
6: **if** $HP \rightarrow \chi = 1$ **then**
7: $associate(MEM, MEM \rightarrow last\_ins, AP[-1], HP)$
8: **else if** $Flag\_C_n = 0$ & $Flag\_C_1 = 1$ **then**
9: $newCN = make\_new\_CN(MEM, C \rightarrow level\_n\_cue, HP)$
10: $associate(MEM, newCN, AP[-1], HP)$
11: $strengthen\_Associated\_DN(AP[-1], HP)$
12: $INC\_ACCESSIBILITY(MEM, HP, AP)$
13: $associate\_temp(MEM, MEM \rightarrow last\_ins, AP[-1], HP)$
14: **if** $HP \rightarrow \chi = 1$ **then**
15: $associate(MEM, MEM \rightarrow last\_ins, AP[-1], HP)$
16: **else if** $Flag\_C_n = 1$ & $Flag\_C_1 = 0$ **then**
17: $newCN = make\_new\_CN(MEM, C \rightarrow level\_1\_cue, HP)$
18: $newDN = make\_new\_DN(MEM, D, HP)$
19: $newGN = make\_new\_GN(MEM)$
20: $associate(MEM, newCN, newDN, HP)$
21: $associate(MEM, newCN, AP[-1], HP)$
22: $associate(MEM, newGN, newCN, HP)$
23: $associate\_temp(MEM, MEM \rightarrow last\_ins, newCN, HP)$
24: **if** $HP \rightarrow \chi = 1$ **then**
25: $associate(MEM, MEM \rightarrow last\_ins, newCN, HP)$
26: $INC\_ACCESSIBILITY(MEM, HP, AP)$
27: **else if** $Flag\_C_n = 0$ & $Flag\_C_1 = 0$ **then**
28: $newCN\_l_1 = make\_new\_CN(MEM, C \rightarrow level\_1\_cue, HP)$
29: $newCN\_l_n = make\_new\_CN(MEM, C \rightarrow level\_n\_cue, HP)$
30: $newDN = make\_new\_DN(MEM, D, HP)$
31: $newGN = make\_new\_GN(MEM)$
32: $associate(MEM, newCN\_ln, newCN\_l1, HP)$
33: $associate(MEM, newCN\_l1, newDN, HP)$
34: $associate(MEM, newCN\_ln, AP[-1], HP)$
35: $associate(MEM, newGN, newCN\_l1, HP)$
36: $associate\_temp(MEM, MEM \rightarrow last\_ins, newCN\_l1, HP)$
37: **if** $HP \rightarrow \chi = 1$ **then**
38: $associate(MEM, MEM \rightarrow last\_ins, newCN\_l1, HP)$
39: $INC\_ACCESSIBILITY(MEM, HP, AP)$

Algorithm 5 INC_ACCESSIBILITY

1: **procedure** INC_ACCESSIBILITY(MEM, HP, AP)
2: **if** $len(AP) > 2$ **then**
3: $pull\_str = min(len(AP) - 2, max(1, HP \rightarrow \eta_3))$
4: $associate(MEM, AP[-(2 + pull\_str)], AP[-1], HP)$
5: **else if** $len(AP) = 2$ **then**
6: $strengthen\_association(MEM, AP[0], AP[1], HP)$

Algorithm 6 RETRIEVE

1: **procedure** RETRIEVE($MEM, C, HP, GIST$)
2: $D = NULL$
3: **if** $neuron\_Count(MEM) = 0$ **then**
4: **return** D
5: **else if** $HP \rightarrow \mu_3 > 0$ **then**
6: $P\_CN = PREDICT(MEM, HP)$
7: **if** $P\_CN != NULL \& HP \rightarrow \mu_4 == 0$ **then**
8: $[AP, Flag\_C_1] = SEARCH(MEM, C, HP, P\_CN, GIST, \text{“}L\text{”})$
9: $D = LEARN\_RETRIEVE(MEM, C, HP, AP, Flag\_C_1)$
10: $UPDATE\_COT(AP[-1], C \rightarrow l\_1\_cue, MEM, HP)$
11: **return** D
12: **else if** $P\_CN != NULL \& HP \rightarrow \mu_4 == 1$ **then**
13: $D = fetch\_Associated\_DN(P\_CN)$
14: $UPDATE\_COT(P\_CN, C \rightarrow l\_1\_cue, MEM, HP)$
15: **return** D
16: **else if** $P\_CN == NULL$ **then**
17: $[Loc\_C_n, Flag\_C_n] = find\_Entry\_Point(MEM, C, HP)$
18: $[AP, Flag\_C_1] = SEARCH(MEM, C, HP, Loc\_C_n, GIST, \text{“}L\text{”})$
19: $D = LEARN\_RETRIEVE(MEM, C, HP, AP, Flag\_C_1)$
20: $UPDATE\_COT(AP[-1], C \rightarrow l\_1\_cue, MEM, HP)$
21: **return** D
22: **else**
23: $[Loc\_C_n, Flag\_C_n] = find\_Entry\_Point(MEM, C, HP)$
24: $[AP, Flag\_C_1] = SEARCH(MEM, C, HP, Loc\_C_n, GIST, \text{“}L\text{”})$
25: $D = LEARN\_RETRIEVE(MEM, C, HP, AP, Flag\_C_1)$
26: $UPDATE\_COT(AP[-1], C \rightarrow l\_1\_cue, MEM, HP)$
27: **return** D

Algorithm 7 LEARN_RETRIEVE

1: **procedure** LEARN_RETRIEVE($MEM, C, HP, AP, Flag\_C_1$)
2: **if** $Flag\_C_1 = 1$ **then**
3: $DN = associated\_DN(AP[-1], HP)$
4: $strengthen\_Associated\_DN(AP[-1], HP)$
5: $INC\_ACCESSIBILITY(MEM, HP, AP)$
6: **return** DN
7: **else if** $Flag\_C_1 = 0$ **then**
8: **return** $NULL$

Algorithm 8 PREDICT

1: **procedure** PREDICT(MEM, HP)
2: $Relevant\_History = MEM \rightarrow Access\_History[-HP \rightarrow \mu_1]$
3: $Sub\_Model = MEM \rightarrow COT\_Model[Relevant\_History]$
4: **if** $Sub\_Model == NULL$ **then**
5: **return** $NULL$
6: $targetNode = -1$
7: $maxWeight = -1$
8: **for each** $(Access, Weight) \in Sub\_Model$ **do**
9: **if** $Weight > maxWeight$ **then**
10: $maxWeight = Weight$
11: $targetNode = Access$
12: **if** $maxWeight > HP \rightarrow \mu_2$ **then**
13: **return** $targetNode$
14: **else**
15: **return** $NULL$

that entry point (Algo. 2), the NoK is readjusted based on the outcome (Algo. 7), and the COT model is also updated using Algo. 9 (lines 16-21). And this is the same if μ_3, is 0 (lines 22-27).

A detailed description of Algo. 2, Algo. 3, and Algo. 5 can be found in sec. C1. Algo. 8 and Algo. 9 are described in Section C3. Inside the LEARN_RETRIEVE sub-operation (Algo. 7), the NoK is modified to reflect a successful retrieve effort (when $Flag\_C_1 = 1$). In that case, the memory strength of the data neuron (DN) associated with the level-1 cue neuron $AP[-1]$ is strengthened (line 3 and line 4). The accessibility of the cue neuron $AP[-1]$ is increased (line 5) using Algo. 5 and the target data neuron is returned (line 6).

3) Chain of Thought: Inside the PREDICT sub-operation (Algo. 8), the relevant memory access history is first extracted (line 2). The Sub_Model corresponding to that access history is fetched in line 3. If Sub_Model is not NULL, then the target neuron having the highest weight is selected (lines 8-11). If this weight passes a confidence threshold check, then the corresponding target neuron is returned as the predicted neuron (lines 12 and 13).

Inside the UPDATE_COT sub-operation (Algo. 9), we first determine if the previously fetched data is a good match for what was requested. Based on this the memory access history is updated (lines 2-6). Next, based on the access history (line 8), we store the recent access ($Fetch\_CN$) inside the $Model$ (CoT Model) (lines 7-16).

D. Retention

The retention operation is used to introduce the effect of ageing into the proposed memory framework (see Algo. 10 and Fig. 7). In each invocation, the recently unused associations are weakened (lines 4-8) and memory strengths are also allowed to decay for recently unused data neurons (lines 9-13). This operation acts as a check against all the association and memory strength reinforcement in Algo. 4 and Algo. 7 with high-level description provided in Fig. 7.

Algorithm 9 UPDATE_COT_MODEL

```
1: procedure UPDATE_COT(Fetch_CN, Target_CN, MEM, HP)
2:     Match_Indicator = match(Fetch_CN, Target_CN, HP)
3:     if Match_Indicator == 1 then
4:         MEM → Access_History.append(Fetch_CN)
5:     else
6:         MEM → Access_History.append(NULL)
7:     if length(MEM → Access_History) > HP → μ1 then
8:         Relevant_History = MEM → Access_History[−HP → μ1]
9:         Model = MEM → COT_Model
10:        if Relevant_History ∈ Model then
11:            if Fetch_CN ∈ Model[Relevant_History] then
12:                Model[Relevant_History][Fetch_CN] + +
13:        else if HP → μ3 > 0 then
14:            if size(Model) ≥ HP → μ3 then
15:                make_Space(Model)
16:            Model[Relevant_History][Fetch_CN] = 1
```

Algorithm 10 RETENTION

```
1: procedure RETENTION(MEM, HP)
2:     A = MEM → Associations
3:     D = MEM → Data_Neurons
4:     for each a ∈ A do
5:         if recently_unused(a) then
6:             Weaken(a, HP)
7:             if strength(a) <= 0 then
8:                 delete(a)
9:     for each d ∈ D do
10:        if recently_unused(d) then
11:            d_str_new = reduce_mem_str(d, HP)
12:            if d_str_new <= 0 then
13:                delete(d)
14:            else
15:                Compress_Mem(d, d_str_new, HP)
```

Algorithm 11 GIST_EXT_SELECTION

```
1: procedure GIST_EXT_SELECTION(MEM)
2:     D = MEM → Data_Neurons
3:     lowestConfidence = 101
4:     targetData = NULL
5:     for each d ∈ D do
6:         conf = d → GIST_Neuron → confidence
7:         if conf = 0 then
8:             return d
9:         else if conf < lowestConfidence then
10:            lowestConfidenc = conf
11:            targetData = d
12:    return targetData
```

Algorithm 12 GIST_EXT_PROPAGATION

```
1: procedure GIST_EXT_PROPAGATION(MEM, HP, DN)
2:     SL = HP → θ1
3:     SD = HP → θ2
4:     GIST_Neu = DN → GIST_Neuron
5:     conf = (GIST_Neu → conf)/SD
6:     gist = GIST_Neu → content
7:     neuronQueue = [GIST_Neu]
8:     numProp = 0
9:     visited = []
10:    while len(neuronQueue)! = 0 do
11:        if numProp >= SL then
12:            break
13:        Neu = neuronQueue.dequeue()
14:        if Neu ∉ visited then
15:            if isGIST(Neu) = T & Neu! = GIST_Neu then
16:                KB = Neu → KB
17:                if gist ∈ KB & KB[gist] < 100 then
18:                    KB[gist] = min(99, conf + KB[gist])
19:                else if gist ∉ KB then
20:                    KB[gist] = conf
21:                setKnowledgeBase(Neu, KB)
22:                numProp = numProp + 1
23:                conf = conf/SD
24:            visited.append(Neu)
25:            n = sort_descending(neighbours(Neu))
26:            index = 0
27:            while index < len(n) do
28:                neuronQueue.enqueue(n[index])
29:                index + +
```

1) GIST Extraction: The spatial organization of data in the proposed memory memory, provides us with a unique advantage when it comes to extracting additional insight (GIST) regarding the stored data. Extracting the GIST of a centrally located neuron in a graph region can allow us to estimate the GIST of neighbouring neurons. This can lead to energy and data transmission saving in different application settings (results in sec. IV). The GIST extraction process is carried out using Algo. 11 and Algo. 12 with high-level description provided in Fig. 7.

Algo. 11 is designed to select the ideal data neuron for which the GIST should be extracted to maximize the overall GIST accuracy of all the data neurons. The data neuron with the lowest GIST confidence is tracked and returned at the end of the search (lines 5-12). If any data neuron's GIST has confidence 0, then it is immediately selected (lines 7 and 8).

Algo. 12 is designed to efficiently propagate GIST information among neighbouring GIST neurons. The input are: (1) MEM, the proposed memory instance, (2) HP, the hyperparameters of the proposed memory instance, and (3) DN, the data neuron from which the GIST is required to be propagated. Few hyperparameters are extracted and variables are initialized in lines 2-9. In lines 10-29, a limited breadth-first search is carried out to propagate the $gist$. The number of propagation is limited by SL (see lines 11 and 12). For any GIST neuron encountered which is not $GIST\_Neu$ (check in line 15), a propagation step is performed. During the propagation, first the knowledge base (KB) is extracted (line 16). KB for each GIST neuron holds the cumulative score corresponding to every GIST value encountered. KB is updated (lines 17-22) and the $conf$ variable is decayed (line 23).

2) Delta Search: In presence of temporal connectivity, it is also possible to fetch multiple data neurons with a specific temporal relation/distance. This operation is implemented using Algo. 13, Algo. 14, Algo. 3, and Algo. 15 (with high level description in Fig. 7).

For Algo. 13 the inputs are: (1) MEM, the proposed memory instance, (2) C, a set of search cues, (3) HP, the hyperparameters of the proposed memory instance, (4) $GIST$ a search GIST value, (5) TL is the temporal search element limit, and (6) M indicates how far or how close temporally the data units should be. An empty data set is returned if MEM is empty (lines 3 and 4). Otherwise, the entry point into the NoK is decided (line 6) and the search for the relevant data is carried out (line 7).

Inside Algo. 14, first we initialization few variables (lines 2-6). Next, we perform a limited breadth-first search to find the target data units (lines 7-28). The search is limited by $limit$ and TL (lines 8-11). Line 14 restricts locality crossover (by limit search beyond level-n cue neurons). If the GIST check is valid and the level-1 cue neuron matches the search cue, then the target neuron is added to the $retList$ provided it is coherent (checked with Algo. 15) with the rest of the elements in the list (lines 16-21).

Fig. 12: Depiction of sample delta search queries.

Algorithm 13 RETRIEVE_DELTA

```
1: procedure RETRIEVE_DELTA(MEM, C, HP, GIST, TL, M)
2:     D = NULL
3:     if neuron_Count(MEM) = 0 then
4:         return D
5:     else
6:         [Loc_Cn, Flag_Cn] = find_Entry_Point(MEM, C, HP)
7:         D = S_DELTA(MEM, C, HP, Loc_Cn, GIST, "L", TL, M)
8:         if HP → μ5 == 1 then
9:             Update_NoK(MEM, Loc_Cn, Flag_Cn, D)
10:        return D
```

Inside Algo. 15, if $M < 0$, it implies that the temporal distance must to at most $\mid M-1 \mid$ (lines 2-9). if $M \geq 0$, it implies that the temporal distance must be at least $\mid M \mid$ (lines 10-16).

3) Activity Insights: Temporal Loops & Sequences: Temporal associations in the NoK can also be inferred as events and activities. For example, a temporal connection between a deer image and a wolf image may indicate that the wolf is following the deer. To detect such activities, we propose the activity detection operation (Fig. 7). This algorithm first identifies background/repeating frames by selecting neurons ($Anchor$) in the NoK with $indegree > SinkTH$. Next, unique paths and loops are identified from these selected neurons ($Anchor$) to detect events and activities.

Algo. 16 describes this operation in more detail. For Algo. 16, the inputs are: (1) M, the proposed memory instance, (2) $sinkTH$, is the sink determination threshold, and (3) HD is the highlight distance or the maximum size of activities to detect. We first extract the reverse graph (line 2). Next for each neuron with an in-degree more than $sinkTH$, we store all the outgoing edges in to_List. Each depth-first search from a given to_List entry that returns back to the same to_List entry) is an event. We limit this search by HD which leads to the discovery of events of length HD max.

Event sequences can also be extracted by analyzing these temporal loops. Assume that event-1 has neurons $E_1 = \{x_1^1, x_1^2, ..., x_1^a\}$ and event-2 has neurons $E_2\{x_2^1, x_2^2, ..., x_2^b\}$. Also let us assume that $\forall x_1^i \in E_1$, the GIST of x_1^i contains a feature F_1 but not F_2. Also let us assume that $\forall x_2^j \in E_2$, the GIST of x_2^j contains a feature F_2 but not F_1. Then E_1 and E_2

Algorithm 14 S_DELTA

procedure S_DELTA($MEM, C, HP, ep, GIST, opFlag, TL, M$)
  $limit = HP \rightarrow \pi_2$
  $matchThresh = HP \rightarrow \lambda_{21}$
  $neuronQueue = [(ep, -1)]$
  $visited = []$
  $retList = []$
  while $len(neuronQueue)! = 0$ **do**
    if $limit! = -1$ & $len(visited) >= limit$ **then**
      break
    if $TL! = -1$ & $TL <= len(retList)$ **then**
      break
    $(Neu, parentNeuron) = neuronQueue.dequeue()$
    if $Neu \notin visited$ **then**
      if $HP \rightarrow \omega = 0 \mid isCue(Neu) = F \mid level(Neu)! = n$ **then**
        if $isCue(Neu) = T$ & $level(Neu) = 1$ **then**
          if $GIST\_CH(Neu, GIST, MEM, HP) = T$ **then**
            $C1 = C \rightarrow level\_1\_cue$
            $sim = compute\_Similarity(Neu, C1)$
            if $sim > matchThresh$ **then**
              if $COHERENT(retList, Neu, M)$ **then**
                $retList.append(Neu)$
      $visited.append(Neu)$
      $n = sort\_descending(neighbours(Neu))$
      $index = 0$
      while $index < len(n)$ **do**
        $neuronQueue.enqueue(n[index], Neu)$
        $index + +$
  return $retList$

Algorithm 15 COHERENT

procedure COHERENT($retList, Neu, M$)
  if $M < 0$ **then**
    $coherent = True$
    $M = -M$
    for each $Neu2 \in retList$ **do**
      if $shortestTemporalPathLength(Neu, Neu2) > M$ **then**
        $coherent = False$
        return $coherent$
    return $coherent$
  else if $M \geq 0$ **then**
    $coherent = True$
    for each $Neu2 \in retList$ **do**
      if $shortestTemporalPathLength(Neu, Neu2) < M$ **then**
        $coherent = False$
        return $coherent$
    return $coherent$

Algorithm 16 ACTIVITY_INSIGHT

procedure ACTIVITY_INSIGHT($M, sinkTH, HD$)
  $RG = generate\_reverse\_graph(M \rightarrow Temporal\_Graph)$
  $Events = dict()$
  for each $Neu \in RGsuchthatindegree(Neu) > sinkTH$ **do**
    $to\_List = adjacent(Neu)$
    for each $event\_Init \in to\_List$ **do**
      $DFS\_List = limitedDFS(event\_Init, RG, Neu, HD)$
      $Events[Neu \rightarrow index] = [DFS\_List]$
  return $Events$

can together form a sequence where two events comprising of F_1 and F_2 separately occur one after the other. For example, if E_1 is related to a deer sighting and E_2 is related to a bear sighting, then E_1 and E_2 together form an event sequence where a deer is followed by a bear sighting. Such sequences can have significant meaning to certain applications such as wildlife surveillance and can provide valuable insight into the observed data.

E. Additional Results

In continuation of our discussion in Sec. IV-D, we visually showcase some of the delta search queries in Fig. 12. The top two sub-figures depict queries with 2 retrieval image limit and the bottom two sub-figures depict queries with 3 retrieval image limit. The queries with clause such as “Atmost 10 temporal distance" retrieves images from the same sighting. The queries with clause such as “At least 40 temporal distance" retrieves images from the different sightings.