Towards Richer Realizations of Holographic CBR

Renganathan Subramanian[1], Devi Ganesan[1], Deepak P[1,2], and Sutanu Chakraborti[1]

[1] Indian Institute of Technology, Madras, Chennai 600036, TN, India
ch16b058@smail.iitm.ac.in, gdevi@cse.iitm.ac.in, sutanuc@cse.iitm.ac.in
[2] Queen's University Belfast, UK deepaksp@acm.org

Abstract. Holographic Case-Based Reasoning is a framework developed to build cognitively appealing case-based reasoners with proactive and interconnected cases. Improved realizations of the Holographic CBR framework are developed using the principles of dynamic memory proposed by Roger Schank and tested on their cognitive appeal, efficiency, and solution quality compared to other relevant systems.

Keywords: Case-based Reasoning · Cognitive CBR · Holographic Systems · Dynamic Memory

1 Introduction

Case-Based Reasoning (CBR) has its inspiration in Roger Schank's seminal work on Dynamic Memory [11] that aspired to model learning in computers based on how learning happens in humans. Schank also proposed the concepts of scripts, plans, and goals as possible knowledge structures used by humans while understanding a piece of text. Kolodner actively worked on Schank's work on dynamic memory and built a computer program called CYRUS [7] which stands for Computerized Yale Retrieval and Updating System. In particular, CYRUS was an attempt to model the reconstructive nature of human memory. CYRUS can be uniquely contrasted against the current day CBR systems in terms of its rich case representation. CBR systems like CREEK [1] , CELIA [13] , CHEF [6] were also built with richly inter-connected case structures. However, in the conventional CBR theory, a case is usually represented as a simple problem and solution pair with no provision to accommodate the interconnections/dependencies between cases. In other words, the conventional CBR theory does not have provisions to neatly accommodate the richly inter-connected case representations found in the complex CBR systems of the past. Holographic CBR [5] is an attempt to provide a single conceptual framework that can cover a spectrum of CBR systems with case representations ranging from simple problem-solution pairs to complex inter-connected cases. This is achieved by modifying the case representation to include a solo and a holo component. While the solo component stands for the conventional problem-solution pair, the holo component is responsible for acquiring/storing/updating all the interconnections between cases.

The idea of holographic CBR was inspired by the holographic nature of the human brain. The term 'holographic' refers to the concept of every part bearing information about the whole. For example, when a holographic image of an apple is broken down into small pieces, every piece would still be able to reconstruct the entire apple image under appropriate conditions. Similarly, the holonomic brain theory suggests that every part of the brain contains information about the whole. Motivated by this observation, holographic CBR was proposed with a holo component to capture the connection/relation of a single case to the entire case base. However, the realizations proposed in [5] are simplistic in that they restrict themselves to the mode of knowledge acquisition during the case acquisition/case addition process. In knowledge-rich domains, a holographic reasoner learns the relation of a case to the whole from the domain expert, whereas, in a knowledge-light setting, it attempts to infer the same from the cases already present in the case base using bottom-up learning methods. It is interesting to note that the paradigm of holographic CBR opens an avenue for integrating both top-down and bottom-up approaches in the building of a cognitively appealing case base. While there is significant scope for exploration in this aspect, in this paper, we focus primarily on forming generalized cases in a holographic reasoner during the case acquisition process itself. This involves invoking a failure-driven reminding process combined with bottom-up learning of the connections between cases. We have, however, restricted our work to regression and classification tasks. In the past, there have been works on generalizations and abstraction in CBR [2,9,15]. However, we are interested in developing a robust and cognitively appealing bottom-up approach to the same.

We discuss the Holographic CBR framework in Section 2. Section 3 introduces the key ideas realized and the realizations built. We present our results in Section 4 and summarize our findings, and discuss the future scope in Section 5.

2 Holographic Case-Based Reasoning

Traditional CBR systems treat cases as isolated entities. Any changes made to one case do not affect the rest of the case base. This is unlike human memory, where a new experience affects related memories, and information is not localized. This idea stems from the experiments of Lashley on mice [8] and observations of Pribram on accident victims [10]. Even when parts of the brain were removed, an organism could still form a hazy recollection of past experiences instead of completely losing them. This shows the 'holographic' nature of human memory, where every part of the system contains information about the entire system. Inspired by this, Holographic CBR treats cases as proactive interconnected entities which actively affect and are affected by any changes to the rest of the case base. Holographic cases develop their own *local* knowledge containers, which helps them understand their problem-solving competence in relation to the rest of the case base. They also proactively interact with and modify the case base. More importantly, this interaction is not necessarily engi-

neered in the reasoner but is learned as the reasoner solves new problems. This Holographic CBR framework has certain key properties.

The Holographic Case: Cases in a holographic CBR system are made up of two parts. The *solo component* stores the problem-solution as in traditional CBR systems and represents the individual experience that the case stands for. On the other hand, the *holo component* defines the case's role with respect to the case base and captures diverse forms of relationship of a case with other cases in the case base.

Holographic Case Addition: New cases are added to the case base only when the system is unable to solve the new case using the existing cases. Thus, the case base grows only when it identifies a knowledge gap. Instead of merely adding the new case to the case base, the system informs the existing cases of the new case's presence. It highlights why the existing case base could not solve the problem and the new case's value-addition and is later used to decide when/how to use the newly added case to solve future problems.

Holographic Problem Solving: The system has a coarse knowledge of the competence of the different cases inside it, but problem-solving happens in a decentralized manner. Cases are expert problem solvers in their neighborhoods. The system uses its global similarity knowledge to retrieve a case to solve a new problem. The retrieved case, in turn, uses its holo component to identify if any other cases can solve the problem better and, if found, transfers control to such a better case.

This treatment of CBR has several advantages. The ability of cases to interact with other cases allows us to design helpful ways to use, modify, and reorganize the case base. The presence of explanations for adding a case not only ensures that only useful cases get added but also highlights the added case's novelty. Ganesan et al. built holographic CBR realizations, which demonstrated some of these ideas. However, the realizations restrict themselves to the mode of knowledge acquisition during the case acquisition/case addition process and explored only limited cognitive ideas. We utilize this framework to infuse several dynamic memory ideas, absent in Ganesan et al.'s initial realizations like forming generalizations based on multiple cases, updating links between cases based on usage, etc., into our CBR realizations to make it more cognitively appealing. These ideas draw inspiration from Schank's works on human understanding and the properties of a dynamic memory system.

3 Methodology

3.1 Key Ideas

In this section, we provide the intuition and justification for the key cognitive ideas implemented. These are then implemented in holographic CBR systems in section 3.2. We use an example of an animal classification task from the UCI Zoo Dataset to motivate these ideas. In this task, each animal (case) has certain

binary biological features like - *lays eggs, produces milk, is a predator, airborne*, etc., and belongs to a class which is one of mammal, bird, fish, amphibian, reptile, flight, and non-flight invertebrate. The CBR system is presented with the cases one at a time from which it learns and then classifies new cases.

Forming Generalizations Schank observed that past experiences stored in memory might contain detailed information not relevant to solving a given task. When two experiences are similar, and when their differences do not contribute additional value to problem-solving, it is useful to form generalizations by combining such similar experiences. These generalizations should only retain information which 1) help them solve the task and 2) differentiate them from other non-similar experiences. This makes the system more efficient by focusing only on important information and ignoring unnecessary details. Moreover, it helps in identifying novel information present in new experiences by comparing them with existing generalizations.

For example, whether an animal is a predator or not does not help in the classification. When the system sees several animals with the same class but different predator values, it should identify this unnecessary feature. Similarly, it should be able to find features that have typical values for a certain class. For example, the class *mammals* has *lays eggs* as predominantly false. With this information, the system should form generalizations about mammals that ignore the useless feature and highlights the typical feature. This generalization can immediately capture interesting information in new cases. For example, when faced with a platypus case (which lays eggs but is still a mammal), the system can identify its novelty by comparing it to the mammal generalization.

Failure-Driven Reminding When a CBR system uses a similar past case to solve a new problem, it expects that the solutions to the past and the new problem are similar. When such *expectations* do not match the ground truth (*expectation-failure*), there is a scope for the system to learn. Schank hypothesized that a dynamic memory system should explain such expectation failures and use them to extract valuable information from the new experience and retain it in memory. Thus, we want a system that remembers its past mistakes and their reasons, which it uses to avoid making similar mistakes again. For example, a tuatara (reptile) and a newt (amphibian) share all features except *aquatic* but belong to different classes. So, a CBR system might incorrectly use its memory of tuatara to classify a newt. Once the mistake is identified, the system should realize that the aquatic feature explains the failure. This intuition forms the basis for our *failure-driven links* present in a case's *holo component*. These links are created when a case makes a mistake in solving a new case. They hold explanations identified by the system for failures and connect the two cases. They later help the system avoid mistakes by reminding it of its past mistakes. In our example, if the system retrieves tuatara again to solve a new problem, it checks if the new case is also aquatic. If so, the system remembers its past mistakes and instead transfer problem-solving control to the connected case newt.

Outcome-driven Solo and Holo Component Weights Cases in holographic CBR have two components - a *solo* component and a *holo* component. However, not all information present in these components would be equally important in solving the CBR task. More importantly, which information is important might differ from one case to another. For example, a seal (mammal) and a frog (amphibian) differ in four features - *hair, legs, milk, and eggs*. But the differing feature *legs* is less important as there are other 4-legged mammals. However, between a honeybee (flight-invertebrate) and a carp (fish), the feature *legs* is important as all flight-invertebrates have legs while no fish do. We have introduced the concept of outcome-driven weights to handle this by which the feature *legs* gets a higher weight in certain cases and a lower weight in certain other cases. These weights indicate the case's confidence in the correctness and utility of different parts of information stored in it. The system progressively learns these weights as it solves new problems.

3.2 Holographic CBR Realization Framework

In this section, we describe a framework of Holographic CBR, which implements the concepts described earlier. We use this framework to build and test two systems for a classification task and a regression task.

Components of the Framework We propose two levels of memory units - cases and generalized cases. Generalized cases are made of multiple cases as discussed in Section 3.1. These units have a solo component that stores their standalone expertise and a holo component connecting them with other units. Both these components have outcome-driven weights (solo-weights and holo-weights) as discussed in Section 3.1 which denote their importance in the unit. When faced with a new problem, these units are retrieved by the system and are used to solve the new problem.

Cases: Cases are the storehouses of knowledge from individual training data points. Each case represents one data point from which the system has learned and serves as a primary knowledge source to solve new problems. Each case is stored within a generalized case. A case as shown in Figure 1 contains the problem definition, solution, and local knowledge in its solo component. It has information about its relative competence with respect to its generalized case in its holo component that is updated as the system learns.

Generalized Cases (GCs): Schank introduced Memory Organization Packets which are organizers of individual experiences centered around common contexts or similar themes. Storing experiences within these MOPs would highlight the interesting aspects of the experiences, and if such interesting aspects are absent or irrelevant, the MOPs aid in removing the unnecessary experience. MOPs are also connected to other MOPs based on important differences.

Similar to this, Generalized Cases are combinations of cases with similar problem representations and similar solutions. Every time a GC is retrieved to

solve a new case during training and is able to do so, the knowledge in the new case updates the GC, and the new case is stored within the GC. The GC can replace the individual cases if the individual case offers no additional value. Thus, a GC represents a region in the problem space that has similar solutions. A GC, as shown in Figure 1 contains a generalized problem description which is a combination of the problem descriptions of the individual cases. The GC's relative competence with respect to other GCs is stored in its holo component as failure-driven links. Multiple cases are retained within a GC.

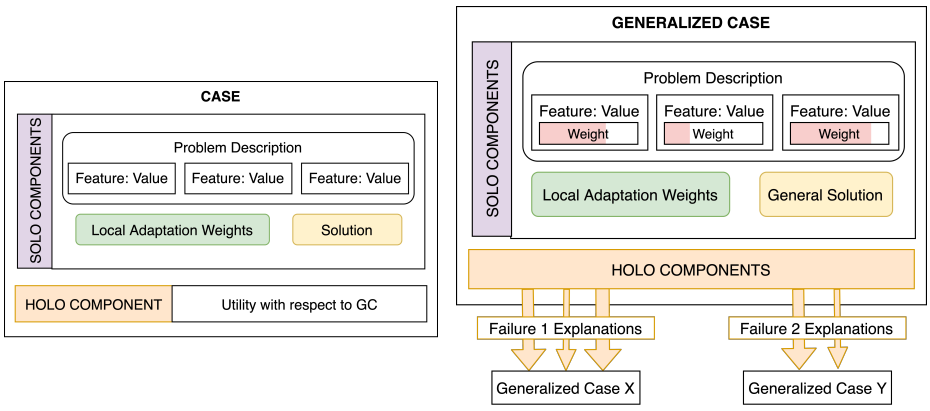


Fig. 1: Memory Units Visualized

Solo Component: The solo component of a memory unit contains:

- *Problem Description:* These are features and their corresponding values. In addition, GCs have outcome-driven weights discussed in Section 3.1 that indicate the importance of a feature-value combination. For GCs, the value for a real feature is the mean value from the cases stored within it. In contrast, categorical features have multiple values for each feature (the feature's value in each case inside it) with different importance weights for each value.
- *Solution:* The solution to the problem the memory unit represents. For regression tasks, a GC's solution is the mean solution of the cases within it.
- *Local Adaptation Knowledge:* This is present only in regression tasks and modifies a memory unit's solution to account for differences between its problem description and that of a new case. We have used the difference between the values of features in the retrieved and to-be-solved case to perform adaptation. Let the unit retrieved be represented as a feature vector $\mathbf{x} = [x_1, \ldots x_n]$ and the new problem by $\tilde{\mathbf{x}} = [\tilde{x}_1, \ldots \tilde{x}_n]$. We know $f(\mathbf{x})$ and want to predict $f(\tilde{\mathbf{x}})$. We define adaptation weights $w_1^a, \ldots w_n^a$ for each feature which are initialized as 0 and progressively learned. We perform adaptation using:

$$f(\tilde{\boldsymbol{x}}) = f(\boldsymbol{x}) + w_1^a(\tilde{x}_1 - x_1) + w_2^a(\tilde{x}_2 - x_2) + \ldots + w_n^a(\tilde{x}_n - x_n) \quad (1)$$

Holo Component: This stores the relative competence of a memory unit with respect to the rest of the case base. In a regression setup, GCs initially store the individual cases within them and compare their adaptation success to that of the GC for future problems. The results of this comparison (how better the case is at solving new problems when compared to its GC, measured in terms of closeness of adapted solution to the ground truth) are stored in each case's holo component and are used to remove cases that do not add additional value over the GC. In classification setups, where the GC and its cases have the same solution and adaptation is not required, such competence is meaningless. Hence, cases update the GCs but are not retained in the case base.

For a GC, the holo component stores the failure-driven links explained in Section 3.1 and connects GCs with one another. When a GC is retrieved but cannot solve a new problem, the explanations for the failure identified are stored in these links, and the new problem is added to the case base and connected via this link. These links are later used to transfer problem-solving control from the retrieved GC to another GC linked to it. In addition, these links also have weights denoting the GC's confidence in their explanation.

Learning Processes in the Framework This section describes how the memory units are created/updated during the learning phase. The system is presented with training data points one at a time, and it iteratively *learns by solving.*

Class 1: Holographic CBR System

```
1  Class HOLOGRAPHIC_SYSTEM:
2  |GC_base //List of GCs in the system
3  |Function ADD_CASE(newCase):
4  | |if GC_base is not emtpty then
5  | | |retGC = Retrieve GC in GC_base with closest solo-weighted distance to newCase
   | | |  (Line 15)
6  | | |retCG.ADD_CASE(newCase)
7  | |else
8  | | |Create newGC by copying newCase, initialize empty holo component and add to
   | | |  GC_base (Line 15)
9  | |end
10 |end
11 |Function SOLVE(newCase):
12 | |retGC = Retrieve GC in GC_base with closest solo-weighted distance to newCase
13 | |retGC.SOLVE(newCase)
14 |end
15 end
```

Initial Solo-based Retrieval: When a new case (QUERY) is encountered, the system compares this problem description with the solo problem description of each of its GCs (GC_i) to compute a distance between the two ($RET_d(GC_i, QUERY)$). It weighs each feature f by the corresponding outcome-driven solo weight ($w_f^{(S,i)}$).

For real features, we use a standardized-Euclidean distance:

$$\mathrm{RET}_d(\mathrm{GC}_i, \mathrm{QUERY}) = \sqrt{\sum_{f \in \text{features}} w_f^{(S,i)} \cdot \frac{(x_{f,\mathrm{GC}_i} - x_{f,\mathrm{QUERY}})^2}{\mathrm{Var}(f)}} \quad (2)$$

And for categorical features we use a modified weighted-Hamming distance:

$$\mathrm{RET}_d(\mathrm{GC}_i, \mathrm{QUERY}) = \sum_{f \in \text{features}} \sum_{x_{f,k} \in \mathrm{GC}_i(f)} w_{x_{f,k}}^{(S,i)} \cdot I(x_{f,k} \neq x_{f,\mathrm{QUERY}}) \quad (3)$$

where x_f is the value taken by the feature f (Categorical features in a GC can have multiple values each of which is represented as $x_{f,k}$), $Var(f)$ is the variance of feature f, and $I(\text{condition}) = 1$ if the condition is true and 0 otherwise. The GC with the lowest distance to the new case is retrieved (Class 1, Step 5) and is used to predict the solution to the new problem (Class 2, Step 6), and this solution is validated with the ground truth (expectation validation).

Formation of Generalized Cases: The first case encountered by the system is stored within a GC with features, values, and solution equal to this case (Class 1, Step 8). It has equal outcome-driven weights for all features and no holo components. When a GC is later retrieved, and there is an expectation success, the problem descriptions are modified to the means of the problem descriptions of the existing cases within the GC, and the new case (for real features) or new feature-value pairs are added to the description (for categorical features) as shown in Class 3, Step 7. If none of the existing GCs can solve a new case, a new GC is again created by copying the new case in Class 3, Step 19.

Update of Outcome-driven Solo Weights: The outcome-driven solo weight of a feature in a GC is increased if the values for the feature in the GC and the case are close during expectation success (Class 3, Step 8) and are far apart during expectation failures(Class 3, Step 16). For example, if a GC containing frog and newt (amphibians) is retrieved to classify a toad (also an amphibian and hence an expectation success), the weight for the feature *backbone*, that has a matching value of 1 in both the GC and the new case, increases. Similarly, if the same GC is retrieved to classify a flea (flight-invertebrate and hence an expectation failure), the weight for the feature *backbone*, which has mismatching values in the GC and the unsolvable case, increases again. Thus, mismatching during failures and matching during successes increases the feature-value importance and vice-versa. For categorical features, we use the ratio of times the feature had matching values in successes and mismatching values in failures to the number of times the GC was retrieved as the weight for a feature. For real features, we compute the difference in values ($\mathrm{DIFF}_f(\mathrm{GC}_i, \mathrm{QUERY})$) for feature f in the GC (GC_i) and a new case (QUERY) as:

$$\mathrm{DIFF}_f(\mathrm{GC}_i, \mathrm{QUERY}) = \frac{(x_{\mathrm{GC}_i,f} - x_{\mathrm{QUERY},f})^2}{\mathrm{Var}(f)} \quad (4)$$

Class 2: Holographic Case

```
Class CASE:
    solo_component //Problem definition, solution and adaptation weights
    holo_component //Relative competence with GC
    pointer_to_GC //Parent GC within which the CASE is stored
    Function PREDICT(newCase):
        Use adaptation weights to predict solution for newCase
    end
    Function UPDATE_ADAPT(newCase):
        Use difference between CASE.predict(newCase) and newCase's solution to update
          adaptation weights of CASE
    end
    Function UPDATE_HOLO(newCase):
        Check whether CASE or CASE.pointer_to_GC is better at predicting newCase and store
          result in CASE.holo (Section 3.2)
        If CASE is consistently worse, DELETE CASE
    end
end
```

Class 3: Holographic Generalized Case

```
Class GC(CASE):
    solo_component //Problem definition, solution and adaptation weights
    holo_component //Failure driven links connecting to other GCs
    cases //Cases stored within GC pointer_to_system
    Function ADD_CASE(newCase):
        if GC.PREDICT(newCase) close to solution of newCase then
            Update problem description and solution of GC
            Increase(decrease) solo weights of features with close(far) values in newCase and GC
            Decrease(increase) holo weights of links with close(far) values in newCase and GC
            for case in GC.cases do
                case.ADAPT_WEIGHT(newCase)
            end
            GC.ADAPT_WEIGHT(newCase)
            Add newCase to GC.cases
        else
            Decrease(increase) solo weights of features with close(far) values in newCase and GC
            transfers = Use holo links to find linked GCs with correct linkGC.predict(newCase)
            if transfers is empty then
                Create newGC by copying newCase, initialize empty holo component, add to GC_base
                Create holo link from GC to newGC for every feature
                Initialize holo weights based on difference in feature values between newGC and GC
            else
                for linkGC in transfers do
                    Increase(decrease) holo weight of links between linkGC and GC where the link value
                      and the corresponding feature value in newCase are close(far)
                end
            end
        end
    end
    Function SOLVE(newCase) (Line 40):
        if Solo-weighted distance between GC and newCase high then
            transferGC = Use holo links to find linked GC with minimum holo link distance to
              newCase
            if Holo distance between newGC and transferGC small then
                transferGC.SOLVE(newCase)
            end
        else
            retCase = Case in GC.cases closest to newCase
            RETURN retCase.PREDICT(newCase)
        end
    end
end
```

Here $x_{\mathrm{GC}_i,f}$ and $x_{\mathrm{QUERY},f}$ are the values of the feature in the i^{th} GC and a new case respectively. Since small differences should increase feature weights during expectation success and decrease weights during expectation failure, we update the solo weights($w^{(S,i)}$) for feature f in the i^{th} GC for expectation success as follows. η is a learning rate parameter between 0 and 1 to avoid over-fitting.

$$w_f^{(S,i)} \leftarrow w_f^{(S,i)} + \eta^{(S,i)} \frac{\max(d_f) + \min(d_f) - d_f}{\sum_f (\max(d_f) + \min(d_f) - d_f)} \tag{5}$$

$$w_f^{(S,i)} = \frac{w_f^{(S,i))}}{\sum_f w_f^{(S,i)}} \tag{6}$$

For expectation failure we use:

$$w_f^{(S,i)} \leftarrow w_f^{(S,i)} + \eta^{(S)} \frac{d_f}{\sum_f d_f} \qquad w_f^{(S,i)} = \frac{w_f^{(S,i)}}{\sum_f w_f^{(S,i)}} \tag{7}$$

Formation of Failure-driven Links and Holo Update: When an expectation fails, the system must explain the failure and create failure-driven holo links (if they do not exist). All feature-values of the unsolved new case are possible explanations for the failure and become links between the initially retrieved GC and a new GC which only has the new case (Class 3, Step 20). However, not all feature-value pairs are equally valid explanations. Features whose values differ significantly between the GC and the case are more likely to be the correct explanations and are weighed more (Class 3, Step 24). However, during an expectation success, the links are not needed, and existing links should not match with the new case. Thus weights of links that match with the new case are reduced, and weights of links that do not match are increased (Class 3, Step 9). Hence, every time a GC is retrieved to solve a new problem, the holo weights of existing failure-driven links are updated/created depending on whether they are useful or not.

For example, a GC made of antelope and buffalo (mammals) might have links to another GC made of crab and lobster (non-flight invertebrates) with features *aquatic:1*, *eggs:1*, and *backbone:0*. When the first GC is retrieved to classify a dolphin (mammal but has *aquatic:1*), there is an expectation success, and the failure-driven links should not be used. Thus, the weight of the link *aquatic:1*, which spuriously matched, goes down while the confidence in *eggs:1* and *backbone:1* as valid failure-driven links goes up.

For categorical features, we assign the holo weight for a failure-driven link as the ratio of the number of times the link matched when needed (expectation failure) or did not match when not needed (expectation success) to the total number of times the GC was retrieved. For real features, we use the difference in feature values between the link and new case defined as $\mathrm{LINK\_DIFF}_f(\mathrm{link}_{ij,f}, \mathrm{QUERY})$ where $\mathrm{link}_{ij,f}$ is the failure-driven link between GCs i and$j$ with feature f. This is calculated as:

$$\mathrm{LINK\_DIFF}_f(\mathrm{link}_{ij,f}, \mathrm{QUERY}) = \frac{(\mathrm{link}_{ij,f} - x_{\mathrm{QUERY},f})^2}{\mathrm{Var}(f)} \tag{8}$$

If the holo weights are $w_f^{(H,ij)}$, during expectation failure, we update it using:

$$w_f^{(H,ij)} \leftarrow w_f^{(H,ij)} + \eta^H \frac{\max(d_f) + \min(d_f) - d_f}{\sum_f (\max(d_f) + \min(d_f) - d_f)} \tag{9}$$

$$w_f^{(H,ij)} = \frac{w_f^{(H,ij)}}{\sum_f w_f^{(H,ij)}} \tag{10}$$

as features with small differences in values are more likely to be correct links.

Whereas, during expectation success, we want to decrease weights of features which have small difference in values and update them as:

$$w_f^{(H,ij)} \leftarrow w_f^{(H,ij)} + \eta^H \frac{d_f}{\sum_f d_f} \qquad w_f^{(H,ij)} = \frac{w_f^{(H,ij)}}{\sum_f w_f^{(H,ij)}} \tag{11}$$

Learning Adaptations: For regression tasks, the adaptation weights in the solo component need to be learnt. During expectation success, when the difference between problem descriptions of the new case and GC is small, the adaptation weights of the GC and the cases stored within it are updated using the Newton's method for optimization (Class 3, Step 11). If $\hat{f}(\boldsymbol{x})$ and $f(\boldsymbol{x})$ are the adapted and true solutions of the new case respectively, we update the adaptation weight vector($\boldsymbol{w^a}$) to minimize the squared difference between these two using:

$$\boldsymbol{w^a} \leftarrow \boldsymbol{w^a} - \left[(\tilde{\boldsymbol{x}} - \boldsymbol{x})((\tilde{\boldsymbol{x}} - \boldsymbol{x}))^T\right]^{-1} \left[\left(\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x})\right)(\tilde{\boldsymbol{x}} - \boldsymbol{x})\right] \tag{12}$$

Solving a New Problem We discuss how the system solves a new problem during the prediction phase.

- *GC Retrieval:* The reasoner uses solo-weighted distance to find the closes GC to solve the new problem (Class 1, Step 12). If the distance between the retrieved GC and the new case is greater than a threshold, the system must decide whether to use this GC or transfer control to another linked GC.
- *Failure-driven Reminding:* The reasoner matches failure-driven links to identify potential GCs to transfer problem-solving control (Class 3, Step 31). The sum of matching weights that lead to any GC denotes the usefulness of a transfer. Control is transferred to the linked GC with the maximum confidence if the total weight leading to such a GC is greater than a threshold.
- *Local-Adaptation:* However, if the confidence is low, control is retained with the GC. If the GC has cases stored inside it, the case closest to the new problem is retrieved (Class 1, Step 36), and its solution is adapted using the case's local adaptation. If no cases are stored within the GC, the GC's adaptation knowledge is used to predict the solution.

Thus, the reasoner can use the interconnected and proactive case base of the holographic CBR framework to implement the dynamic memory ideas of outcome-driven weighted retrievals, forming and validating expectations, creating generalizations, and failure-driven reminding to learn and solve problems.

4 Results and Interpretations

This section tests our realizations on their efficiency and solution quality. All results are averages over 50 runs where the train-test split and the order of case addition are randomized. We build two systems that solve different tasks using data from the UCI Machine Learning Repository [4]:

- *Zoo Case Base:* This is a classification task that has been discussed before in Section 3.1. It has animals from 7 classes represented by 16 categorical biological features.
- *Energy Efficiency Case Base [14]:* This is a regression task to predict the heating load and cooling load requirements of buildings (that is, energy efficiency) using eight real-valued building parameters.

4.1 Comparison with Baseline

In this section, we compare our holographic approach with other systems which perform the same tasks. We use these systems as a baseline to illustrate the improvements obtained by the holographic framework. For the classification task, we compare with two machine learning models and two holographic systems which lack some cognitive aspects developed in this work:

- *Naive Bayes:* This is a parametric ML model that does not retain experiences but builds a model using the entire training data at once.
- *K-Nearest Neighbors:* This is a non-parametric model which retains the entire training data but uses multiple (k) data points taken together to solve a problem. This is a simplified version of our holographic CBR without weights, failure-driven links, or generalizations.
- *Ganesan et al.'s System:* This previous holographic system does not have generalizations or outcome-driven weights but has expert-given failure-driven links to connect cases.
- *ML Switching Model:* This is a modification made to our realization where the failure-driven links transfer control to an ML model (Naive Bayes) instead of other GCs during expectation failures.

For the regression task, we compare the system with:

- *Ridge Polynomial Regression:* This is a parametric method that, unlike our approach, does not retain experiences but instead builds a model based on all the training data points taken at once.
- *K-Nearest Neighbors:* This non-parametric model uses the average solution of the k-nearest neighbors to predict the new solution.
- *Ganesan et al.'s System:* This previous system has local adaptation with a weighted linear regression model in each case. It does not have control transfers or generalizations and retains all training data points as cases.

Table 1: Comparison with other systems (averages over 50 runs)

Classification Models	**Test Accuracy**	**Regression Models**	**Test RMSE**
Current System	93.771	Current System	1.4606
Previous System	91.523	Previous System	1.8272
KNN	93.226	KNN	2.1749
Naive Bayes	91.649	Polynomial Ridge Regression	3.1693
ML Switching Model	92.718		

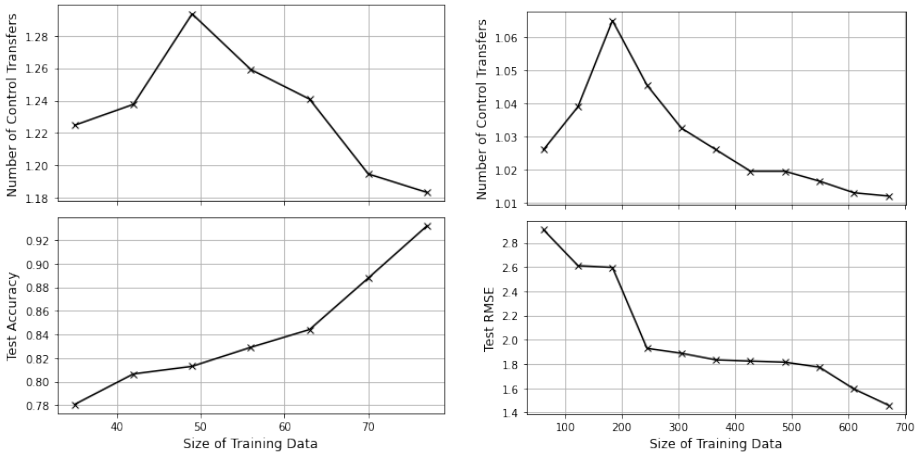
The results of the comparison are in Table 1. The improved holographic systems outperform the ML models. The holographic system does not lose out on the solution quality despite its cognitive appeal. KNN, which retains all the data points, can still not outperform our approach, which only retains a fraction of the training data. In our approach, cases are aware of each other's competence and can coordinate better to solve the problem. On the other hand, in (parametric) ML models (Naive Bayes and Ridge Regression), the training data points interact to create the model but lose their individual competencies when a model which might not reflect the ground truth is enforced. Regression builds a model by treating the entire problem space as one while the holographic system has local pockets of knowledge in the local knowledge containers.

Our approach thus finds a middle ground by retaining cases but also allowing them to interact in a holographic fashion. It is able to identify structures in the problem in a bottom-up fashion and exploit it to achieve better performances. Our system also outperforms the previous holographic realizations. The previous realizations treat all components equally important and miss out on the merits of forming generalizations. This highlights the importance of the generalization mechanism, which identifies regions of similarity in the problem space and combines the knowledge present in multiple similar cases.

4.2 Tests for Efficiency

CBR systems suffer from the utility problem [13] where, as the case base grows, the knowledge of the system and its solution quality improve, but the system's efficiency drops. It has been observed that better indexing and case base maintenance can handle this trade-off [3]. This section tests how our system handles this trade-off by monitoring its efficiency as the number of training cases increases. We track the number of problem-solving control transfers as a proxy for efficiency. The holographic reasoner has additional computation costs over a traditional CBR system due to control transfer using failure-driven links. The number of such transfers is an indication of this additional cost. We also track the test data accuracy (or RMSE for regression tasks). The system's test accuracy should ideally increase with more training data without reducing efficiency.

In both the tasks, as shown in Figure 2a and Figure 2b, we observe that as the training data increases, the test performance increases without significantly



(a) Zoo Case Base (b) Energy Efficiency Case Base

Fig. 2: Tracking efficiency and accuracy with increase in training data

increasing the control transfers. Contrarily, the number of transfers decreases after a point. As the amount of training data crosses a limit, the outcome-driven weights are well-tuned, and as a consequence, the initial retrievals are more accurate. Even when control transfers are used, they arrive at the correct solution faster. This indicates the system's ability to improve both its efficiency and solution quality with increasing training data.

5 Conclusions and Future Scope

We have expanded the holographic CBR framework and developed improved realizations that draw insights from popular models of dynamic memory and are cognitively appealing. We have demonstrated holographic CBR's broad scope, which offers an interconnected and proactive case base to build practical systems that can outperform traditional methods in selected tasks both in terms of efficiency and solution quality. With this, we aim to establish holographic CBR as a general-purpose CBR framework using which we can build a myriad of systems with different applications, memory models, amount of domain knowledge, and end goals. In this way, we establish holographic CBR not as a problem-solving tool but rather as a paradigm to design such tools.

We aim to view CBR the way it was envisioned during its initial phases and look past the haze created by practical constraints. By framing CBR as a Memory-based Reasoning Framework and improving its cognitive appeal using insights from models of human understanding, we aim to demonstrate the richness of the CBR framework and its relevance in building better Artificial Intelligence systems [12]. This research work is a step in that direction.

The ideas developed here - forming generalizations, expectation-failure-driven

reminding, local knowledge containers, and outcome-driven weights are just a few of the numerous cognitive mechanisms that can be realized. More importantly, the way in which these have been realized in this work is not the only way to do so. Nevertheless, we hope that the results from this work and the ideas presented pave the way for further integrating cognitive memory-based reasoning components with holographic CBR.

References

1. Aamodt, A.: Knowledge-intensive case-based reasoning in creek. Lecture Notes in Computer Science Advances in Case-Based Reasoning p. 1–15 (2004). https://doi.org/10.1007/978-3-540-28631-8_1
2. Bergmann, R., Vollrath, I.: Generalized cases: Representation and steps towards efficient similarity assessment. In: Burgard, W., Cremers, A.B., Cristaller, T. (eds.) KI-99: Advances in Artificial Intelligence. pp. 195–206. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
3. De Mantaras, R.L., Mcsherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Keane, M., Aamodt, A., Watsoni, I.: Retrieval, reuse, revision and retention in case-based reasoning. Knowledge Engineering Review **20**(3), 215–240 (2005), www.scopus.com
4. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
5. Ganesan, D., Chakraborti, S.: Holographic case-based reasoning. In: Watson, I., Weber, R. (eds.) Case-Based Reasoning Research and Development. pp. 144–159. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58342-2_10
6. Hammond, K.: Chef: A model of case-based planning. In: AAAI (1986)
7. Kolodner, J.L.: Reconstructive memory: a computer model*. Cognitive Science **7**(4), 281–328 (1983). https://doi.org/10.1207/s15516709cog0704_2
8. Lashley, K.S.: In search of the engram., pp. 454–482. Physiological mechanisms in animal behavior. (Society's Symposium IV.)., Academic Press, Oxford, England (1950)
9. Maximini, K., Maximini, R., Bergmann, R.: An investigation of generalized cases. In: Ashley, K.D., Bridge, D.G. (eds.) Case-Based Reasoning Research and Development. pp. 261–275. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
10. Pribram, K.H.: Brain and perception: holonomy and structure in figural processing. Lawrence Erlbaum Associates, Hillsdale, N.J (1991)
11. Schank, R.: Dynamic memory - a theory of reminding and learning in computers and people (1983)
12. Slade, S.: Case-based reasoning: a research paradigm. AI Magazine **12**(1), 42–42 (Mar 1991). https://doi.org/10.1609/aimag.v12i1.883
13. Smyth, B., Cunningham, P.: The utility problem analysed. In: Smith, I., Faltings, B. (eds.) Advances in Case-Based Reasoning. pp. 392–399. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
14. Tsanas, A., Xifara, A.: Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy and Buildings **49**, 560–567 (2012). https://doi.org/https://doi.org/10.1016/j.enbuild.2012.03.003
15. Zito-Wolf, R., Alterman, R.: Multicases: a case-based representation for procedural knowledge. In: Proceedings of the 14th Annual Conference of the Cognitive Science Society. pp. 331–336. Lawrence Erlbaum (1992)