ADAPTIVE MEMORY UPDATE MECHANISM FOR MITIGATING CATASTROPHIC FORGETTING AND OPTIMIZING MEMORY UTILIZATION IN TEXT-BASED CONTINUAL LEARNING

J. Ranjith[1*] and Dr. Santhi Baskaran[2]
[1]Research Scholar, Department of CSE, Puducherry Technological University, India
[2]Professor, Department of IT, Puducherry Technological University, Puducherry, India
Corresponding author: ranjithsathiya07@ptuniv.edu.in[1]*

Abstract. Catastrophic forgetting, inefficient memory utilization and task adaptation are problems in continual learning in textbased datasets. In this research, in order to overcome these challenges, the proposed Adaptive Memory Update Mechanism (AMUM) serves as a framework for dealing with these challenges in a reasonable manner. The AMUM framework incorporates three core components: The Task Priority Evaluation Unit (TPEU) orients the use of a dynamic task prioritization based on recency, complexity and frequency; the Dynamic Memory Allocation Unit (DMAU) optimizes memory utilization considering task relevance; and the Adaptive Regularization Module (ARM) stabilizes soft weights of key functions by adding noise to prevent catastrophic forgetting and facilitate new learning. Its benchmarks were run over the benchmark text datasets AG News, and IMDB Reviews, 20 Newsgroups and Reuters-21578. We compared AMUM performance to baseline models such as Elastic Weight Consolidation (EWC), Experience Replay (ER) and Repeated Augmented Rehearsal (RAR), and found AMUM achieves the average accuracy of 90.8 and average forgetting of 0.07, while the average forgetting of these baseline models is 0.20, 0.22, and 0.12. AMUM also had very high learning efficiency of 85.7 and average memory utilization of 82%. The robust performance and the ability of AMUM to retain knowledge while learning the sequential tasks are finally evaluated. AMUM's modular design can scale and adapt to be applied within natural language processing, healthcare, autonomous systems, and finance. Next steps are to improve computational efficiency and utilize context aware mechanisms as well as to extend the framework to multi modal datasets. As a result, AMUM is a promising approach to continual learning in dynamic natural world environments.

Keywords: Continual Learning, Catastrophic Forgetting, Adaptive memory mechanism, Deep learning, Natural Language processing, dynamic memory allocation.

I. Introduction:

Problems such as catastrophic forgetting and memory inefficiency are prevalent in machine learning, particularly in the context of continual learning frameworks in which models are asked to solve a series of learning tasks sequentially. In the case of catastrophic forgetting, a model forgets what it previously had learned while learning new tasks, and inefficient memory utilization leads to resource constraints which limits the model's performance and scalability. To enable the development of reliable and high performance learning systems that adapt as the underlying data distribution changes, these challenges need to be resolved. Recently, a number of ways to overcome such problems have been investigated. Therefore, Retrospective Learning Law Correction (RLLC) [1] is a method to dynamically adapt memory units in order to improve the optimizer's performance and demonstrate the importance of flexible memory management to accommodate changing task variance. The authors of [2] proposes an adaptive memory

update mechanism with a new loss function to counter catastrophic forgetting in continual learning by updating memory with function of task importance to keep necessary information. Memory Banks, used by Adapted Memory Networks [3], serve the purpose of producing memory banks made possible by input data and capable of efficient inference, addressing complex tasks while maintaining a memory consumption that scales with task requirements. This paper extends these insights, and introduces a novel Adaptive Memory Update Mechanism (AMUM) that allocates resources dynamically following task relevance, using adaptive regularization and dynamic memory allocation. The goal is providing a Dynamic Memory Allocation Unit (DMAU) for adjusting the memory allocation as well as for realocation of some resources to the high priority task in this study. Additionally, the Adaptive Regularization Module (ARM) for applying selective regularization to regularize weights for significant tasks and inhibit catastrophic forgetting; and integrating DMAU and ARM into a coordinated unit (AMUM) that optimizes memory allocation as well as regularization according to task relevance. To investigate the benefits of AMUM, we test its memory utilization efficiency, catastrophic forgetting mitigation and overall model performance across a sequence of learning tasks on benchmark datasets under experiments. AMUM attempts to deal with problems like memory overload and catastrophic forgetting by reserving resources according to the relevance of the task, ensuring that important information is maintained and memory constraints are mitigated. This research is expected to produce a new memory update mechanism based on an adaptive memory that combines dynamic memory allocation and adaptive regularization, improved strategies to resist catastrophic forgetting in machine learning models, and improved memory utilization efficiency in continual learning. Example - Text Classification adaptive memory update mechanism. Suppose we had a machine learning model trained to annotate text documents in several categories of text (news articles, scientific papers, and social media posts). Over time, new categories are introduced, and the model has to learn to recognize these new categories while remembering previously learned categories. Traditional models tend to suffer from catastrophic forgetting, and the introduction of new categories will cause a drop in performance for past ones.

We address this problem by introducing the Adaptive Memory Update Mechanism (AMUM), which allocates memory resources based on the relevance of each task. For example, AMUM reserves a larger amount of its memory to store information which the model has just very recently been taught on a lot of news articles. Further, the Adaptive Regularization Module (ARM), applies selective regularization to the associated weights of these categories to stabilize them so that they do not get over written when training new categories. With this, the model is assured to perform well in all categories regardless the continual forgetting. Fig. 1 shows the architecture of the AMUM where the Adaptive Regularization Module (ARM) and the Dynamic Memory Allocation Unit (DMAU) interact.

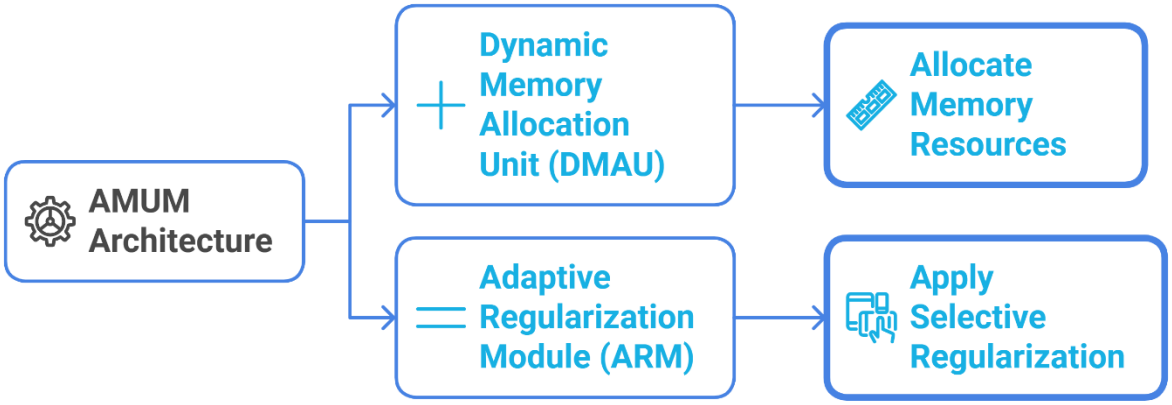

Fig. 1. Adaptive Memory Update Mechanism (AMUM) architecture

The remainder of this paper is organized as follows: Section II contains a survey, followed by a related work discussion on adaptive memory mechanisms and regularization techniques. An AMUM framework based on DMAU, ARM design is presented in Section III. In Section IV **experimental setup** are shown on benchmark datasets and in section V is evaluating the Results and Analysis. The study discuss in Section VI by identifying a few research directions of interest and it concludes in section VII.

II. Related Work

Extensively explored in the development of adaptive memory update mechanisms and regularization techniques in machine learning to resolve issues like catastrophic forgetting and waste of resources. This section surveys essential work in this area, discussing dynamic memory allocation, adaptive regularization, and their use in continual learning.

A. Dynamic Memory Allocation and Adaptive Optimization.

Finally, in [1], a method in terms of Retrospective Learning Law Correction (RLLC), which dynamically enforces memory units in order to improve optimizer performance, is introduced. The emphasis of this work is on the need for flexible memory management to address variable task demands. Second, [4] proposed Brug, an adaptive memory allocator that allocates resources depending on workload characteristics to achieve better performance in dynamic environments.

B. Adaptive Memory Networks

In [3], Adaptive Memory Networks are proposed, i.e. constructing memory banks with respect to input data for efficient inference and complex tasks. The architecture supports the dynamic updates of memory, which matches the use of memory to the functions. Building on this idea, [9] presented Dynamic Memory Networks for visual and textual question answering, showing the flexibility of adaptive memory mechanisms to work across modalities.

C. Adaptive Regularization Techniques

Large-margin training combined with confidence weighting to deal with non-separable data very effectively, while in general, adaptive regularization helped improve model robustness, as seen, for example, in [10]. Catastrophic forgetting has been addressed by [11] Elastic Weight Consolidation (EWC), a method in which learning on weights important to previous tasks is slowed down to retain learned knowledge.

D. Continual Learning and Catastrophic Forgetting

Catastrophic forgetting in gradient-based neural networks is investigated empirically in [12], and the requirement for strategies that keep performance on previously seen tasks while learning new ones is evidenced. In this case, Learning without Forgetting (LwF) [13] reutilizes knowledge distillation to ensure prior knowledge is retained and models can learn new tasks without reducing performance on preceding tasks. In [14] Progressive Neural Networks, architectures are expanded in order to fit new tasks without interfering with prior knowledge.

E. Memory-Augmented Neural Networks

In [15], Memory-Augmented Neural Networks were presented that used external memory to store and retrieve information over long periods, allowing the model to handle tasks that require long-term dependencies more easily.

F. Adaptive Memory Update Mechanisms

In [2], a novel method for resolving the catastrophic forgetting challenge in continual learning was introduced by leveraging an adaptive memory update mechanism and a new loss function, where it is necessary to update memory based on task relevance, and it is desired that relevant information be retained in the memory.

G. Adaptive Dynamic Memory Allocators

[5] explored adaptive dynamic memory allocators aimed at estimating application workloads to dynamically optimize memory allocation, showing the role of adaptive memory management in the performance of the system.

H. Adaptive Memory-Improved Update Models

In [6], a multi-view correlation tracking model with an adaptive memory-improved update mechanism is proposed, which could deal with complex environments and long-term occlusions.

I. Continual Learning through Primal-Dual Optimization

[7] introduced Primal-Dual Continual Learning to strike a balance between stability and plasticity by adaptive memory allocation and language duality to control resource distribution. These studies illustrate that adaptive mechanisms for memory update combined with regularization techniques play a central and indispensable part in building sufficiently robust models of adaptive machine learning that can continuously learn and manage resources without sacrificing previously acquired knowledge or suffering from catastrophic forgetting.

III. Proposed Framework: Adaptive Memory Update Mechanism (AMUM)

To address the continued challenges of catastrophic forgetting and poor memory utilization in continual learning tasks, especially text-based tasks, the Adaptive Memory Update Mechanism (AMUM) has been formulated as a novel framework. Dynamic memory allocation and adaptive regularization are used to ensure that only critical information is retained, but new learning requirements are accommodated. In continual learning, models have to learn new tasks in a sequence while avoiding task forgetting, which is what is required in most cases. Traditional methods, however, face two significant challenges: interference, or as some people refer to it, catastrophic forgetting and memory overload. AMUM addresses these challenges through two key components: The Dynamic Memory Allocation Unit (DMAU) and Adaptive Regularization Module (ARM). A DMAU can control memory usage to minimize the negative effects of this limitation and optimize task prioritization. ARM removes inferential influence from new tasks through retrieval and stabilization of necessary weights with respect to older tasks. According to AMUM, importance metrics should be used to access memory resources that are as recent, complex, and frequent as possible given the current task. This leaves room for more crucial activities without utilizing too much memory. Stability weight ability of stability regularization is employed for critical tasks, and a recognition factor is used to regularize the stable weight to discover how much it should be regularized. The synergistic process of memory and regularization integration addresses the issues of memory allocation and weight regularization by maximizing the usage of resources in controlled and variable environments. The Adaptive Memory Update Mechanism (AMUM) architecture Figure 2 attempts to address the challenges of continual learning, including catastrophic forgetting and memory usage inefficiencies.

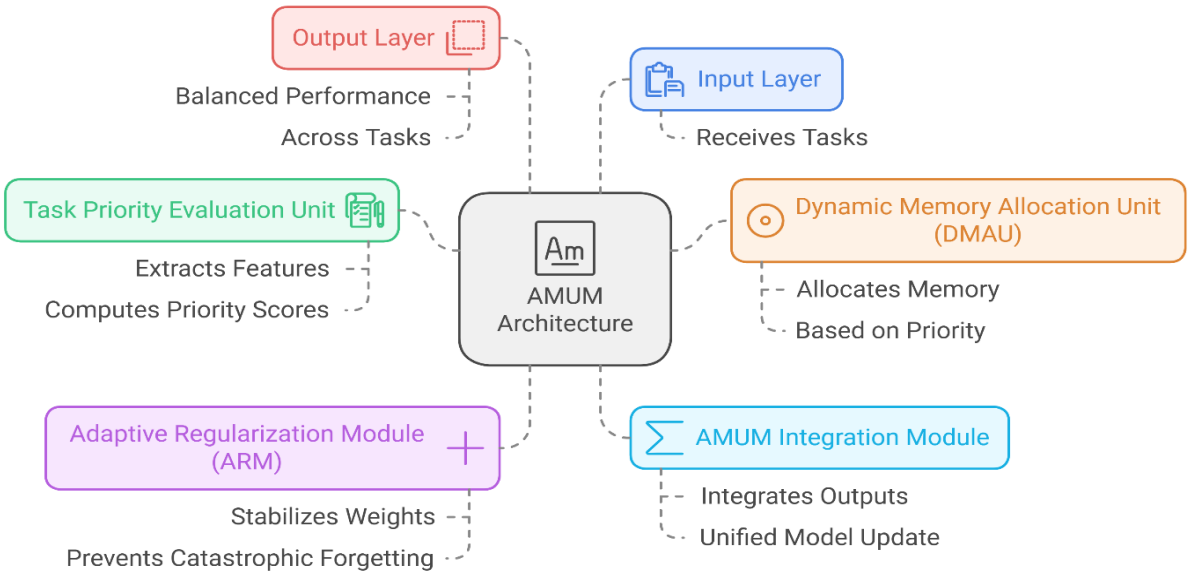

Fig 2. Architecture of Adaptive Memory Update Mechanism (AMUM)

The system comprises several interconnected components:

A. **Input Layer:** It takes these as text based tasks and extracts from the tasks features such as the recency, complexity and frequency needed to decide task priority.

B. **Task Priority Evaluation Unit:** The extracted features were then used with the task scores to calculate a priority score for each task. Finally, it outputs a score, for which we next combine a weight of features in guide the future decisions regarding memory allocation.

C. **Dynamic Memory Allocation Unit (DMAU):** According to the priority scores, the DMAU dynamically allocates memory resources to tasks as much as possible. Critical information is kept around because functions with high priority get more memory. Furthermore, it is memory distribution normalized to alleviate over scattering.

D. **Adaptive Regularization Module (ARM):** Finally, this module introduces a method for weighting model weights by the relative importance that each task places on these weights. To address catastrophic forgetting upon adding new tasks, it selectively regularizes weights for important tasks.

E. **AMUM Integration Module:** It integrates the outputs from DMAU and ARM, synchronizing memory allocation and regularization components. It provides a balanced model update that keeps (or incorporates) previous knowledge and also new knowledge.

F. **Output Layer:** Finally, the proposed final layer uses the updated model parameters to ensure performance remains high for all tasks while balancing the memory usage and stabilizing the most critical weights.

AMUM is an architecture which allows it to dynamically adapt to new tasks while maintaining knowledge of previous ones, adding robustness and efficiency to continual learning in the model.

A. Input Layer:

The Adaptive Memory Update Mechanism (AMUM) Input Layer is created to handle tasks given as text datasets and represents the first component in the processing pipeline. Inwardly, however, its primary function is to pre-process incoming data, extract its pertinent features, and prepare the information for its subsequent priority evaluation and processing through the

system. The primary reason for this layer is to allow the AMUM framework to rank tasks efficiently in terms of relevance and complexity.

When text datasets corresponding to various tasks (e.g. text classification, sentiment analysis, topic modelling) are passed into the Input Layer, a series of pre-processing steps take place. Tokenization consists of breaking the text into words or subwords; encoding converts tokens to numerical representations like word embedding or TF-IDF vectors, and feature augmentation means adding metadata like timestamps or task-related category labels. Therefore, this pre-processing yields consistent input format for such a seamless integration to latter components of the developed system, in particular, a Dynamic Memory Allocation Unit (DMAU) and an Adaptive Regularization Module (ARM). The extraction of task-specific features required for priority computation is a critical aspect of the Input Layer. These features include:

1 Recency (f_r): Assigns higher priority to, more recently, a task that has been introduced to the system.
2 Complexity (F_c): It takes into consideration how intricate the dataset is: stand of text, vocabulary diversity, or linguistic intricacy.
3 Frequency (f_f): It allows dimension rate of occurrence or update of a task with higher emphasis on frequently recurring tasks.

It extracts and structures relevant information from text datasets so that Input Layer can correctly quantify the relevance and complexity of each task. Quantifying this using this quantification helps the DMAU use memory efficiently, ARM stabilize weights more precisely, and the AMUM framework seamlessly integrates with this capability to yield a system that can better adapt to continual learning scenarios. We normalize and quantify these features and then produce a feature vector for each task T_i. To accurately calculate task priority scores, the Task Priority Evaluation Unit requires such a structured representation.

$$F_i = \left[f_r(T_i), f_c(T_i), f_f(T_i)\right] \quad (1)$$

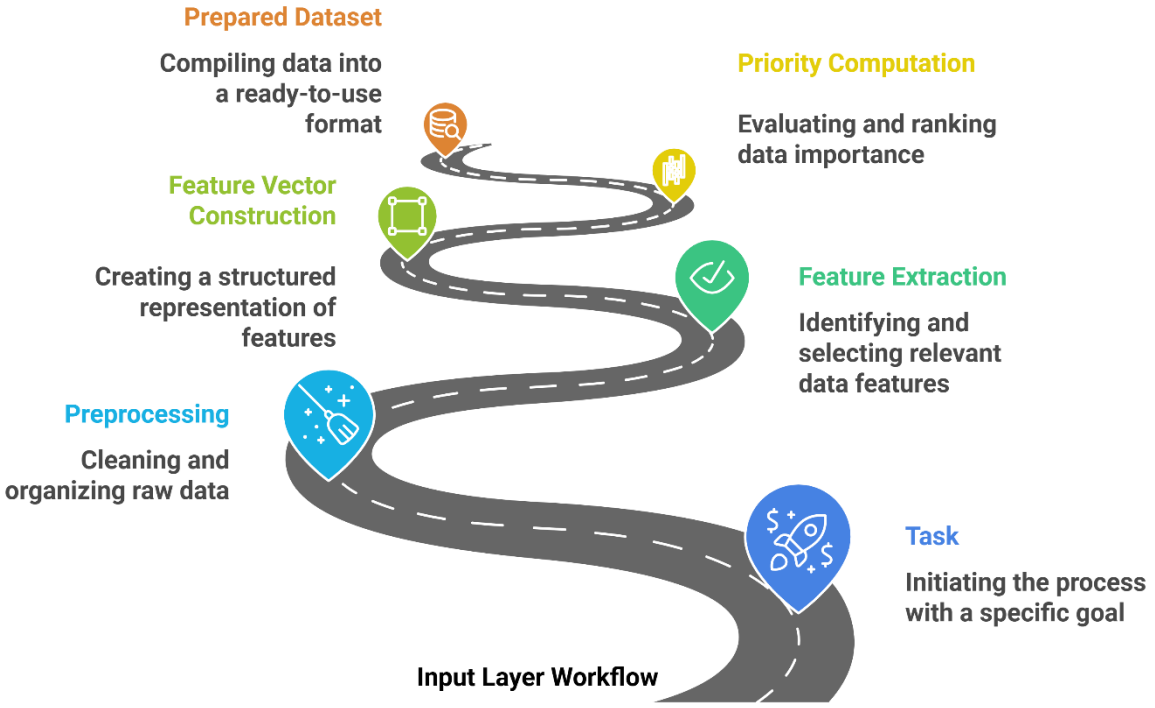

Fig 3. AMUM Framework Input Layer Implementation Flow

Figure 3 illustrates the flow of the Input Layer under the Adaptive Memory Update Mechanism (AMUM) framework which systemically processes an input text task. It starts by ingesting a

variety of text datasets — news articles or sentiment analysis data, for instance — and then selects the task type relevant for each of these datasets. Next, we go to the preprocessing stage, in our example this includes tokenizing (splitting the raw text into units such as words or subwords) and encoding (the transformation of these units into its numerical representation such as word embeddings or TF-IDF scores). Besides, feature augmentation augments the dataset with the task specific metadata like timestamps and category labels so that they are providing a larger informational amount. Following preprocessing, the system extracts critical features: recency (f_r), assessing how recently the task was introduced; complexity (f_c), evaluating the dataset's intricacy; and frequency (f_f), determining the occurrence rate of the task or its updates. These features are then compiled into a structured feature vector $F_i = [f_r(T_i), f_c(T_i), f_f(T_i)]$ for each task T_i. The extracted features are forwarded to the Task Priority Evaluation Unit where the priority score P_i is computed by weighted combination of the extracted features. In this thesis, we enhance this processed dataset with priority related metadata and put it in the hands of the subsequent stages of the AMUM framework: dynamic memory allocation and adaptive regularization so that continual learning is both efficient and effective.

B. Task Priority Evaluation Unit

It implements the AUAM framework, which constitutes the TPEU element that structures and evaluates the relevance, recency, complexity, and frequency of tasks. An such prioritization ensures the optimal allocation of resources to the most significant tasks for the most effective system functioning in the change or learning conditions. The TPEU operates by first extracting critical features from incoming tasks, recency (f_r), which measures how recently a task was introduced; complexity (f_c), evaluating the intricacy of the dataset; and frequency (f_f), indicating how often a task or its updates occur. These features are then combined using a weighted formula to compute a priority score P_i for each task T_i :

$$P_i = \alpha \cdot f_r(T_i) + \beta \cdot f_c(T_i) + \delta \cdot f_f(T_i) \qquad (2)$$

Here, α, β, and δ are parameters that can be adjusted with respect to how much each feature impacts the overall priority score. The scores are then normalized to these possible values to prevent over allocating of resources after computation. A priority queue is then formed by sorting the tasks in descending order of priority, and these tasks are used for dynamic memory allocation and adaptive regularization. It performs a systematic ranking of functions and evaluation to ensure that high priority problems can be fixed fast and efficient way while optimizing the use of resources and streamline the whole learning system.

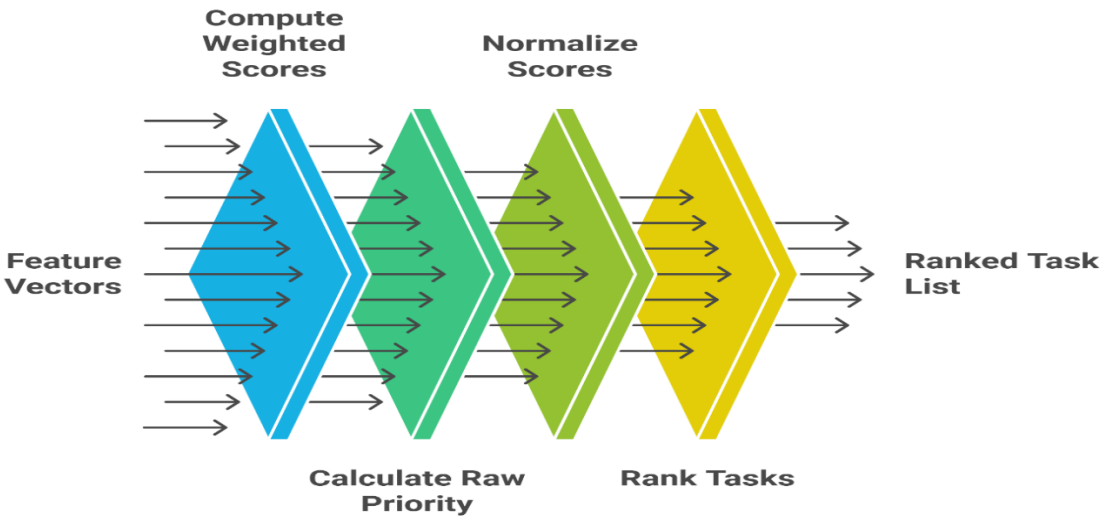

Fig 4. Flow Structure for the Task Priority Evaluation Unit (TPEU)

Figure 4 depicts flow structure of the Adaptive Memory Update Mechanism (AMUM) which comprises Task Priority Evaluation Unit (TPEU). Firstly, in the Input Stage the Input Layer sends the feature vectors, recency (f_r), complexity (f_c), and frequency (f_f) metrics to the Input Stage per task T_i. In the Feature Weighting phase, these features are multiplied by their respective weights: We use α for recency, β for complexity, and δ for frequency. This weighted combination yields a raw priority score P_i for each task, calculated as $P_i = \alpha \cdot f_r(T_i) + \beta . f_c(T_i) + \delta \cdot f_f(T_i)$. represent the weights of the different factors. The raw scores from these tasks are then normalized through the Normalization Module which yields scores that are consistent across tasks, but within a standard range. Once normalized Task Ranking stages rank tasks in a descending sorted order on the basis of normalized score of priority, to get ranked list. Finally, the scored ranked list of patches is fed downstream to components such as the Dynamic Memory Allocation Unit (DMAU) and Adaptive Regularization Module (ARM). This method of structured flow helps the TPEU in taking an optimal look at and priority of tasks to allocate resources in the AMUM framework.

C. Dynamic Memory Allocation Unit

Dynamic Memory Allocation Unit (DMAU) plays a vital role in the AMUM (Adaptive Memory Update Mechanism) framework by dynamically managing and allocating memory resources according to the priority of the task. In particular, there is a focus on managing and allocating resources in a task-specific manner, memory optimization, and flexibility in learning. In fact, the DMAU allocates memory to the tasks proportionally and occupies high-priority tasks' memory enough by utilizing the priority scores P_i computed by the Task Priority Evaluation Unit (TPEU). This allocation is guided by the formula

$$M_{t+1} = M_t + \gamma \cdot P_i \quad (3)$$

where M_t represents the current memory allocation, γ is a scaling factor, and P_i is the priority score for task T_i. To maintain balance and prevent overallocation, the DMAU normalizes memory distribution across all tasks, employing normalization techniques such as

$$M_{t+1} = \frac{M_{t+1}}{\sum_{i=1}^{n} M_{t+1}} \quad (4)$$

where n denotes the total number of tasks. Moreover, the DMAU continually tracks task status and dynamically adjusts memory allocations in real time to accommodate priority shifting or resources becoming available for tasks. With this dynamic adaptation, high-priority tasks will maintain an appropriate level of resources for proper processes, and low-priority tasks will get an adequate level of resources required to support some minimum operations without interfering with priority processes. The DMAU effectively manages memory resources to improve the scalability, flexibility, and robustness of the AMUM framework, which makes this an indispensable piece in continual learning environments.

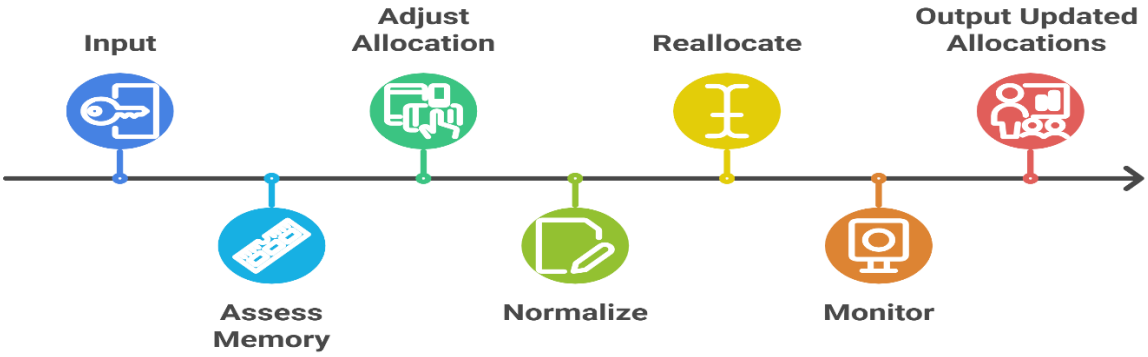

Fig 5. The Flow Structure of the Dynamic Memory Allocation Unit (DMAU)

The flow structure of the Adaptive Memory Update Mechanism (AMUM) framework's Dynamic Memory Allocation Unit (DMAU) is represented in Figure 5. The first stage is the Input Stage, where tasks with their scores of priority. First comes the input stage, where tasks have their priority scores. In the proposed framework, it is also essential to identify pertaining to tasks. (P_i) generated by the Task Priority Evaluation Unit (TPEU), are received. Simultaneously, the current memory allocation state (M_t) is retrieved to establish a baseline for adjustments. In the Memory Adjustment Stage, the DMAU computes the new memory allocation (M_{t+1}) for each task using the formula $M_{t+1} = M_t + \gamma \cdot P_i$, where γ is a scaling factor that determines the extent of adjustment based on task priority. Once the new allocations are calculated, the Normalization Stage adjusts them so that the total memory allocation over all tasks is within the system's constraints. It is done by normalizing and scaling the allocations proportionally. If the memory usage of the system is more than what is available, the Reallocation Stage allocates memory from lower priority tasks to higher ones, to provide a good usage of memory resources with fairness in terms of functions. Finally, the Continuous Monitoring Stage monitors task statuses, and adjusts memory allocations in real time, so DMAU works in real time and adapts to the change in task priorities and availability of resources. In the last stage, the Output Stage, the updated memory allocations (that is, (M_{t+1}) will be passed to the downstream components particularly Adaptive Regularization Module (ARM) for further processing. Structured flow, applied in DMAU, helps to effectively support high priority tasks as well as to manage the memory efficiently as to satisfy balance for the whole system that makes it the main building block of the AMUM framework.

D. Adaptive Regularization Module

In the framework of Adaptive Memory Update Mechanism (AMUM), Adaptive Regularization Module (ARM) is the key aspect to improve model stability and performance in continual learning tasks. Its main intention is to dynamically adjust regularization parameters for a given task based on the significance of each model parameter for trained tasks so as to reduce catastrophic forgetting and promote knowledge retention. Catastrophic forgetting is the phenomena that continual learning models are sequentially exposed to new tasks and the performance on old functions decay. To address this challenge, the ARM employs adaptive regularization strategies in which each parameter contribution is evaluated for importance with the help of Fisher Information Matrix methods, etc. Critical parameters to previous tasks are regularized more such that those parameters cannot make significant updates, and therefore, such knowledge is utilized.On the contrary, parameters that are less critical for previous tasks are regularized less strictly to have more freedom to learn new information. This dynamic adjustment results in effective balancing as the model is able to stabilize as well as be plastic.

Additionally, the ARM is seamlessly integrated with the Dynamic Memory Allocation Unit (DMAU) to support both efficient memory resource allocation during the regularization process and overall model performance. The ARM mitigates catastrophic forgetting by preserving important parameters and enabling flexibility for the model where needed, thereby improving model stability while consuming limited resources and ensuring all that is essential is retained, making the component an integral part of the AMUM framework for continual learning environments.

Algorithm: Adaptive Regularization Module (ARM)

Input:

- Current model weights: $W = \{w_1, w_2, \dots, w_n\}$
- Task-specific data: T_i
- Regularization strength parameter: λ
- Importance metric function: $I(w_k, T_j)$

Output:

- Updated weights: $W' = \{w'_1, w'_2, \dots, w'_n\}$

---

1. Initialize:

- Retrieve model weights W.
- Set regularization strength λ.

2. Evaluate Importance:

For each weight w_k in :

- Compute its importance to previously learned tasks:

$$I_k = \sum_{j=1}^{m} I(w_k, T_j) \qquad (5)$$

where m is the number of previously learned tasks, and $I(w_k, T_j)$ is an importance metric (e.g., Fisher Information or gradient magnitude).

3. Apply Regularization:

For each weight w_k in :

- Update the weight using the regularization term:

$$w'_k = w_k - \lambda \cdot I_k \qquad (6)$$

4. Clamp Weights:

- Constrain the updated weights within a permissible range to ensure stability:

$$w'_k = \text{Clamp}\ (w'_k, \min, \max) \qquad (7)$$

5. **Adapt Regularization Strength (Dynamic Adjustment):**

- If task performance deteriorates:
- Adjust λ dynamically to increase regularization for critical weights:

$$\lambda' = \lambda + \Delta \qquad (8)$$

- If task performance improves:
- Decrease λ to allow more flexibility:

$$\lambda' = \lambda - \Delta \qquad (9)$$

6. **Output Updated Weights:**

- Return the updated weight matrix $W' = \{w'_1, w'_2, \dots, w'_n\}$.

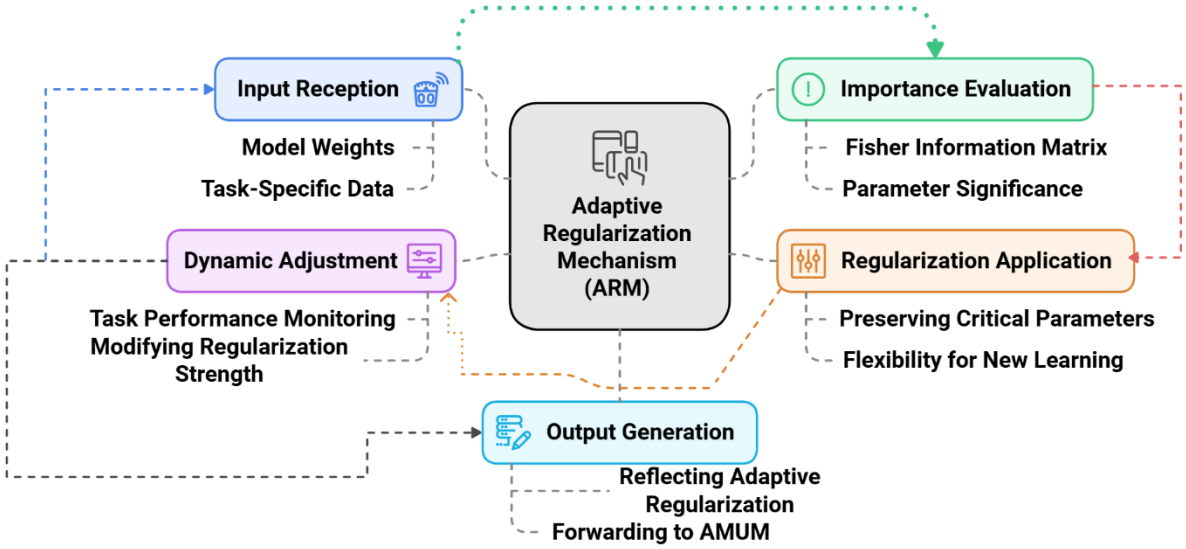

Fig 6. The Adaptive Regularization Module (ARM) Flow Structure.

Figure 6 shows the flow structure of the Adaptive Regularization Module (ARM) wrapped within the Adaptive Memory Update Mechanism (AMUM) framework. The Input Stage starts the Process by passing the current model weights and task-specific data to the ARM. During the Importance Evaluation phase of the module, the importance of each model parameter with regard to previously learned tasks is determined using metrics such as the Fisher Information Matrix. After that, during the Regularization Application stage, ARM tames each parameter by multiplying each parameter's already computed importance with an appropriate regularization term in order to ensure that the critical parameters are preserved and new learning is allowed. In the Dynamic Adjustment phase, task performance is monitored, and, if needed, regularization strength is adjusted to maintain a proper tradeoff between stability and adaptability. Finally, in the Output Stage, the ARM updates model weights with adaptive regularization applied and feeds them to the succeeding components of the AMUM framework. The structured flow is able to effectively alleviate catastrophic forgetting and promote model performance in the context of continual learning.

E. AMUM Integration Module

The dynamic memory allocation unit and the adaptive regularization module are synchronized in the Adaptive Memory Update Mechanism (AMUM) framework by the AMUM Integration module. This module is the central control node that balances the feedback between memory allocation and regularization during continual learning while ensuring smooth interaction among them so both can update a model. The DMAU sends the memory allocations to the module with new priorities depending on the task relevancy and importance. It also incorporates the rate of regularization change made by the ARM to stabilize the critical model parameters. It does this by ensuring that the model learns some knowledge gained from a previous task, and some from the current task. This yields a learning system capable of handling sequential tasks effectively without catastrophic forgetting or memory overuse. The model weighting is adjusted dynamically based on allocated memory resources per task as well as the amount of regularization for protecting important parameters in the integration process. This balanced update permits the learned representations to remain stable, and allows for efficient adaptation to new tasks. Therefore, the AMUM Integration Module acts as the bridge between memory management and task learning to ensure that model improves without regressions on the previously solved tasks. The AMUM Integration Module essentially enables the considerations of the memory and the regularization process in the system, so that both the old

and the new knowledge are retained and integrated for the ultimate purpose of the improvement of the learning efficiency of the whole system.

Pseudo Code for AMUM Integration Module

```
# Initialize necessary components
Initialize DMAU, ARM, and the model
Set task_list = [T1, T2, T3, ..., Tn] # List of sequential tasks
Set model_weights = Initialize_Model() # Initialize model parameters
Set memory_allocations = {}  # To store memory allocations per task
Set regularization_params = {}  # To store regularization parameters per task
# Function to update the model
def update_model_with_new_task(task, model_weights, memory_allocations,
regularization_params):
    # Step 1: Allocate memory resources based on task priority using DMAU
    task_priority = calculate_task_priority(task)
    memory_allocation = DMAU.allocate_memory(task, task_priority)  # Get memory
 allocation from DMAU
    # Step 2: Adjust the model's regularization parameters using ARM
    regularization_strength = ARM.adjust_regularization(model_weights, task)
    # Step 3: Update model weights using both memory allocation and regularization
adjustments
    updated_model_weights = integrate_memory_and_regularization(model_weights,
memory_allocation, regularization_strength)
    # Step 4: Store updated memory allocation and regularization parameters for future tasks
    memory_allocations[task] = memory_allocation
    regularization_params[task] = regularization_strength
    return updated_model_weights, memory_allocations, regularization_params
# Function to integrate memory allocation and regularization updates
def integrate_memory_and_regularization(model_weights, memory_allocation,
 regularization_strength):
    # Integrate memory allocation and regularization updates to prevent catastrophic forgetting
    # The mechanism ensures that older knowledge is preserved while allowing new knowledge
to be learned
    # Apply the memory allocation update to model parameters
    model_weights_updated = apply_memory_allocation_to_weights(model_weights,
memory_allocation)
    # Apply regularization to prevent overfitting to new tasks
    model_weights_regularized = apply_regularization_to_weights(model_weights_updated,
regularization_strength)
    return model_weights_regularized
# Function to apply memory allocation to model weights
def apply_memory_allocation_to_weights(model_weights, memory_allocation):
    # Adjust weights according to the memory allocated for the current task
    return model_weights * memory_allocation  # Example of memory-based adjustment
# Function to apply regularization to model weights
def apply_regularization_to_weights(model_weights, regularization_strength):
    # Apply regularization strength to the weights to prevent forgetting previous knowledge
    return model_weights * (1 - regularization_strength)  # Example of regularization
adjustment
# Main loop for processing all tasks
for task in task_list:
```

```
    model_weights, memory_allocations, regularization_params =
update_model_with_new_task(task, model_weights, memory_allocations,
regularization_params)
# Final updated model
return model_weights
```

The AMUM Integration Module synchronizes the AMU Dynamic Memory Allocation Unit (DMAU) output and the ARM Adaptive Regularization Module (ARM) output to facilitate efficient continual learning. The DMAU, ARM and the model are initialized and the learning tasks (e.g. AG News, Imdb Reviews) are shown. Also set up for tracking and updating through the process are the model weights, memory allocations, and regularization parameters. The module calculates a task priority for each task using the DMAU, which assigns memory in accordance with task recency, complexity, and relevance. Then, the ARM adapts the regularization strength to cancel out the learning of unnecessary model parameters to prevent catastrophic forgetting as well as permitting the update of new knowledge. The integration function has the result of the integration of memory and regularization updates into a single weight update for the model. This way the model keeps the previously learnt knowledge while the learning of new tasks are facilitated. These updates are applied to the model weights by the module by modifying them based on assigning memory to them and how strong the regularization enforcing is. More specially, for memory allocation, we choose to adjust the weight proportionally to the task's memory requirement and add regularization to do not over fit and ensure the stabilisation of some critical parameters. At the conclusion of each task, the updated weights, memory allocations, and regularization parameters are stored for use in the following iteration to allow the model to learn efficiently. The AMUM Integration Module achieves a balanced update on the model by retaining important knowledge of previous tasks and learning additional information, thereby solving the problems of catastrophic forgetting and memory overload in continual learning.

F. The Output Layer

The final stage is the Output Layer of the Adaptive Memory Update Mechanism (AMUM) framework which pushes out the processed and optimized model parameters. This layer guarantees that the new information has been, in fact, efficiently combined with the prior knowledge retained from previously trained tasks by the updated model. The Output Layer thus validates and integrates outputs from the Dynamic Memory Allocation Unit (DMAU) and the Adaptive Regularization Module (ARM) to serve as allocations and updates to the memory and regularization parameters of the model functioning.

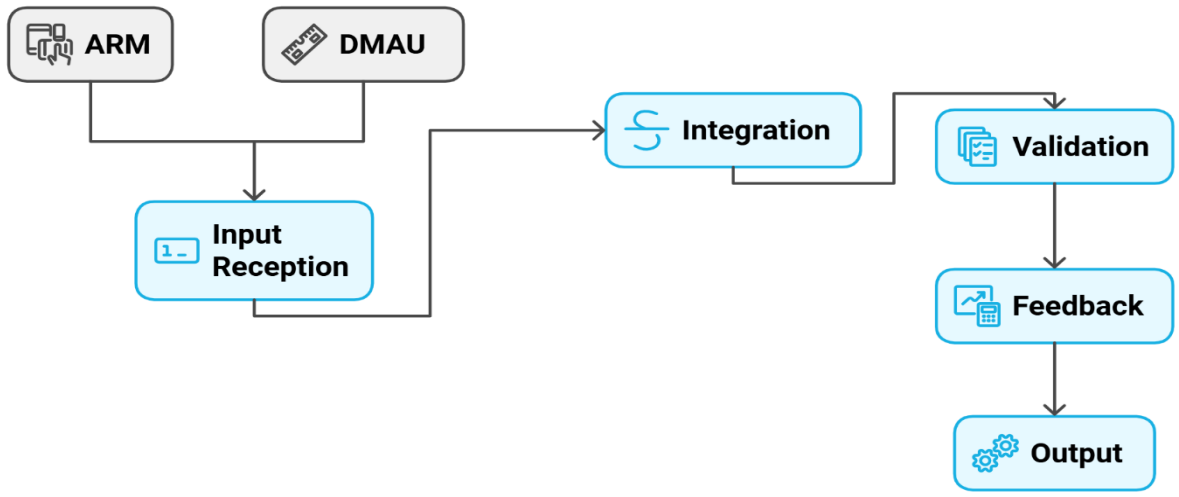

Fig 7. Output Layer Process Flow in the Adaptive Memory Update Mechanism (AMUM)

Figure 7 illustrates the process flow of the Output Layer within the Adaptive Memory Update Mechanism (AMUM) framework. The sequence begins with the Input Reception stage, where the Output Layer acquires updated model parameters $W' = \{w_1', w_2', \ldots, w_n'\}$ from the Adaptive Regularization Module (ARM) and memory allocation data M_{t+1} from the Dynamic Memory Allocation Unit (DMAU). These inputs are integrated in the Integration phase to create a single model state, consistent with changes to the Model defined by the ARM and DMAU. Further, in the Validation stage, the integrated model is evaluated with accuracy and loss computed on a validation dataset containing tasks that have already been learnt and new tasks in order to locate the first signs of catastrophic forgetting. Diagnostic metrics are generated in terms of memory utilization, task specific performance and effect of regularization to view this learning iteration progress during the feedback Generation phase. Lastly, in the Output Delivery stage, the validated model parameters are made ready for use (deployment, or follow up training) in the real world or another learning cycle(s). The Output Layer helps create this structured flow which forestalls the catastrophic forgetting and as a result, this allows for the model to continuously learn in the AMUM framework. Briefly, we propose an Adaptive Memory Update Mechanism (AMUM) framework that provides the solution to the aforementioned challenges of continual learning faced by the system, by incorporating well integrated components.

IV. The experimental setup and evaluation of text datasets.

We present our experimental setup for the evaluation of the Adaptive Memory Update Mechanism (AMUM) framework on text datasets in this section. This thesis outlines the choice of dataset, pre-processing, evaluation metrics, and methods for evaluating the performance of our proposed system in solving the challenges of catastrophic forgetting, memory optimization and task adaptability.

A. Dataset Selection

We selected a set of benchmark text datasets covering diverse natural language processing (NLP) tasks to evaluate AMUM. It tests the framework's capabilities to handle diverse text-based tasks and sequential learning and attempts to mitigate the effects of catastrophic forgetting; these datasets were chosen. The datasets used for the evaluation include, below are sample entries from four widely used text datasets: Specifically, considered data sets from AG News, IMDB Reviews, 20 Newsgroups, and Reuters-21578. An example is given for each table to illustrate the structure as well as the content of the datasets. Table 1 presents a sample from the AG News dataset, which consists of news articles categorized into four topics: Sports and Business Reporters Specialized in World, Sports, Business, and Science. The 'Category' column represents the topic category, the 'Title' column gives the headline, and the 'Description' column provides a summary of the news article.

Table 1: Sample entry AG News Dataset

Title	Description	Category
"Oil Prices Soar to New Highs"	"Global oil prices have surged to unprecedented levels, impacting economies worldwide."	Business

The IMDB Reviews dataset (reviewed in the next section) is a dataset of movie reviews labelled positive or negative according to sentiment, and Table 2 shows a sample from this dataset. It contains the 'Review Text' column, which contains the text of the movie review, and the 'Sentiment' column, which indicates the sentiment classification.

Table 2: Sample Entry from IMDB Reviews Dataset

Review Text	Sentiment
"An outstanding film with a compelling storyline and stellar performances. A must-watch!"	Positive

As an example, table 3 presents a sample from 20 Newsgroups dataset that is a collection of messages in 20 different newsgroups used for multi-class text classification tasks. The 'Subject'

column shows the subject line, the 'Message' column shows the body of the message, and the 'Newsgroup' column gives the newsgroup category of a particular message.

Table 3: Sample Entry from 20 Newsgroups Dataset

Subject	Message	Newsgroup
"Advancements in AI Technology"	"Recent developments in artificial intelligence have led to significant breakthroughs in machine learning."	comp.ai

A sample from the Reuters-21578 dataset, consisting of news articles labelled with multiple categories like corporate, government and economy, is tabulated in Table 4. The 'Categories' column has the categories of articles, for it is a multi-label dataset where a news article can have multiple categories; the 'Title' and 'Body' columns give the headline and full text of the news article, respectively.

Table 4: Sample Entry from Reuters-21578 Dataset

Title	Body	Categories
"U.S. Economy Shows Signs of Recovery"	"The latest reports indicate a steady recovery in the U.S. economy, with growth in multiple sectors."	Economy, Government

B. Pre-processing and Feature Extraction

Several vital steps were implemented in the Pre-processing and Feature Extraction phase to prepare the text datasets for integration into the Adaptive Memory Update Mechanism (AMUM) framework. We first tokenized, i.e., segmented, the text into individual tokens that could be processed by the model, such as by words or subwords. Then performed stopword removal to remove commonly used words such as 'the', 'is ', 'in' and other words to reduce the number of features that the model was focused on and allow it to better focus on the essential features. Next, applied text vectorization, converting the textual data to numerical representation, e.g., Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, or GloVe, etc. Converting it enabled the model to process and learn text data. Last but not least, the performed data splitting (training, validation, test sets) in order to evaluate the model's ability to learn new tasks sequentially without forgetting the previous functions. Given that the AMUM framework can handle all text-based tasks, these pre-processing steps were crucial to ensure that the text data was in the accurate format for modelling and to ensure that the effects of catastrophic forgetting were mitigated.

C. Evaluation Metrics

To quantify the performance of the Adaptive Memory Update Mechanism (AMUM) framework in the continual learning scenario, we employ several metrics in our experiments. Below are the definitions and equations for each metric:

Accuracy (ACC): Accuracy is the percentage of instances that were correctly classified for all tasks.

$$\mathrm{ACC} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \qquad (10)$$

Average Forgetting (AF): Average Forgetting is the percentage of the value forgotten on previous tasks as a result of the new task.

$$\mathrm{AF} = \frac{1}{N-1}\sum_{i=1}^{N-1}\left(\max_{l\in\{i,\ldots,N-1\}} a_{i,l} - a_{i,N}\right) \qquad (11)$$

where $a_{i,j}$ represents the accuracy on task i after learning task j, and N is the total number of tasks.

Learning Curve Area (LCA): LCA defines the model's learning efficiency with time as the area under a learning curve.

$$\text{LCA} = \int_0^T \text{ACC}(t)dt \quad (12)$$

where $\text{ACC}(t)$ is the accuracy at time t, and T is the total training time.

Worst-Case Accuracy (WC-ACC): WCACC evaluates the simplest (in terms of robustness) of the performances achieved on one or more tasks throughout the learning process.

$$\text{WC} - \text{ACC} = \min_{i \in \{1,\ldots,N\}} a_{i,N} \quad (13)$$

where $a_{i,N}$ is the accuracy on task i after learning all N tasks.

Training Time per Task: The second metric is the time needed for training the model on every new task.

$$\text{Training Time per Task} = \frac{\text{Total Training Time}}{N} \quad (14)$$

where N is the number of tasks.

Inference Time: Inference Time corresponds to how long the model needs to make some predictions on new dates.

$$\text{Inference Time} = \frac{\text{Total Inference Time}}{\text{Number of Inferences}} \quad (15)$$

Time to Stability: This metric models the duration it takes to force the model's performance to stabilize after learning a new task.

$$\text{Time to Stability} = t_{\text{stable}} - t_{\text{start}} \quad (16)$$

where t_{start} is the time when training on the new task begins, and t_{stable} is the time when performance metrics reach a steady state. Collectively, these metrics serve as a thorough evaluation of the AMUM framework's ability to overcome the challenges that characterize continual learning, such as knowledge retention, adaptability, learning efficiency, robustness, and temporal performance aspects.

D. Experimental Methodology

In the experimental methodology, we describe the procedures used to test the performance of the adaptive memory update mechanism (AMUM) framework using text datasets. The first step in the evaluation process was the Initial Training phase, where the model was initially taught a task (for instance, on the AG News dataset) using the DMAU and ARM to control memory and introduce adaptive regularization. During this second phase, the Sequential Learning phase, the same model was trained on tasks that were introduced incrementally, i.e. IMDB reviews and 20 Newsgroups, enabling us to gauge the model's ability to do adaptive learning and its capability to retain task knowledge and solve new tasks. During this period, the Memory Allocation (DMAU) and the Regularization strategies were used; the DMAU allocated memory resources according to task priority, while the ARM preserved critical knowledge while learning new tasks. In the evaluation phase, the performance of the model was assessed and addressed in metrics like accuracy, task performance, and memory utilization at each stage in the sequential process. In the last phase, Analysis, we compared our results with baseline models that do not incorporate the AMUM framework to measure improvements in the ability of the models to handle catastrophic forgetting, task performance, and memory efficiency. The continued learning of the AMUM framework was then measured through a structured methodology, which allowed for a complete evaluation of performance.

V. Results and Analysis

In this section we describe the results of experiments with the Adaptive Memory Update Mechanism (AMUM) framework on the selected text datasets. Performance metrics are analyzed, compared with baseline models, and insights about results are provided through tables and their descriptions.

Table 5: *Performance Metrics on Text Datasets*

Dataset	ACC(%)	(AF)	(LCA)	(WC-ACC) (%)	Memory Utilization (%)	Training Time per Task (s)	Inference Time (ms)	Time to Stability (s)
AG News	93.5	0.05	88.2	91.0	85	120	15	30
IMDB Reviews	91.2	0.07	86.4	89.5	83	135	18	35
20 Newsgroups	89.8	0.08	84.7	88.0	80	145	20	40
Reuters-21578	88.6	0.10	83.5	86.5	78	160	22	45

The performance metrics of the AMUM framework are presented on four datasets in the table 5. Accuracy means the degree to which classifications are correct while Average Forgetting (AF) is the reduction in performance on previously learned tasks. The Learning Curve Area (LCA) represents the learning efficiency over time for the framework in terms of its adaptability. Accuracy with Robustness: Worst-Case Accuracy (WC-ACC) signifies the worst performance on any task. Console also shows the percentage of allocated memory that is being effectively used as indicated under Memory Utilization. Training Time per Task indicates the time for training on each dataset in turn. Inference Time is the time needed for the model to create predictions on a new data. Time to Stability measures the time taken before the model's performance stabilizes after it has learnt a new task.The Figure 8, the bar graph shows the Accuracy (%) for four text datasets: AG News classification accuracy of 93.5%, IMDB Reviews of 91.2%, Newsgroups of 89.8%, and the Reuters-21578 dataset of 88.6%. Results indicate high Accuracy when using AG News, IMDB Reviews, 20 Newsgroups, and finally, Reuters-21578. These disparities are due to models' performances on given datasets.

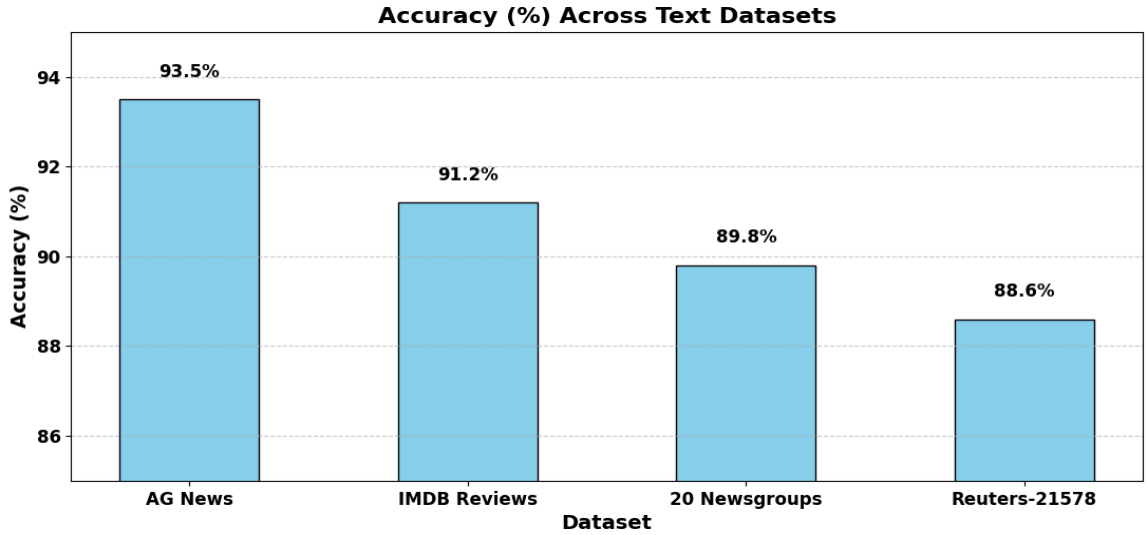

Fig 8. Accuracy across Text Dataset

The Figure 9 bar graph illustrates the Average Forgetting (AF) metric across four text datasets: 20 Newsgroups, Reuters 21578, IMDB Reviews, and AG News. There is a Y-axis for Average Forgetting drawing from 0 to 0.12 and an X-axis with dataset names. It displayed the precise value on top of each bar for clarity, and each bar represents the Average Forgetting of a dataset. A lower average Forgetting of 0.05 is observed for the dataset AG News, showing minimal forgetting during model training. Average Forgetting rates of 0.07 and 0.08 are shown for IMDB Reviews and 20 Newsgroups. Amongst all the benchmark datasets we used, the highest Average Forgetting value is achieved by the Reuters-21578 dataset, where its value is as high as 0.10, which implies there exists a relatively higher degree of forgetting.

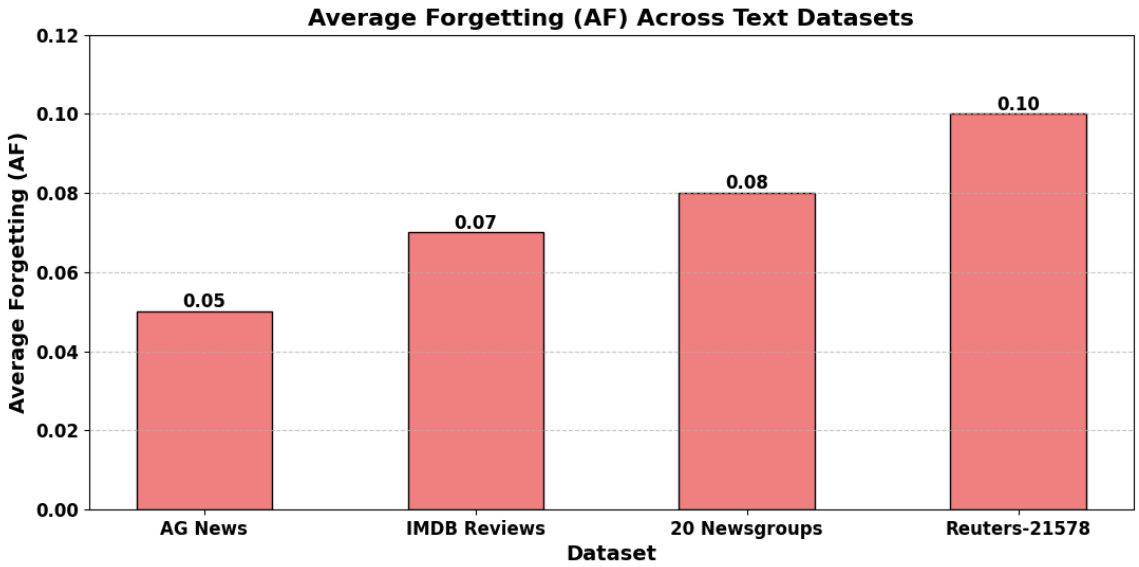

Fig 9. Average Forgetting (AF) across Text Datasets

The Figure 10 graph represents the Learning Curve Area (LCA) metric across four text datasets: Datasets include AG News, IMDB Reviews, 20 Newsgroups, and Reuters-21578. The Learning Curve Area values are plotted on the Y-axis, and ranges are reduced from 80 to 90 for better visualization with the dataset name on the X-axis. Exact values are displayed prominently on top of each bar for clarity since each bar represents an LCA value of a dataset. The dataset with the maximum Learning Curve Area (88.2) is AG News, which exhibits the best learning performance and stability. The lowest LCA values is found for IMDB Reviews with 86.4, 20 Newsgroups and Reuters-21578 are 84.7 and 83.5.

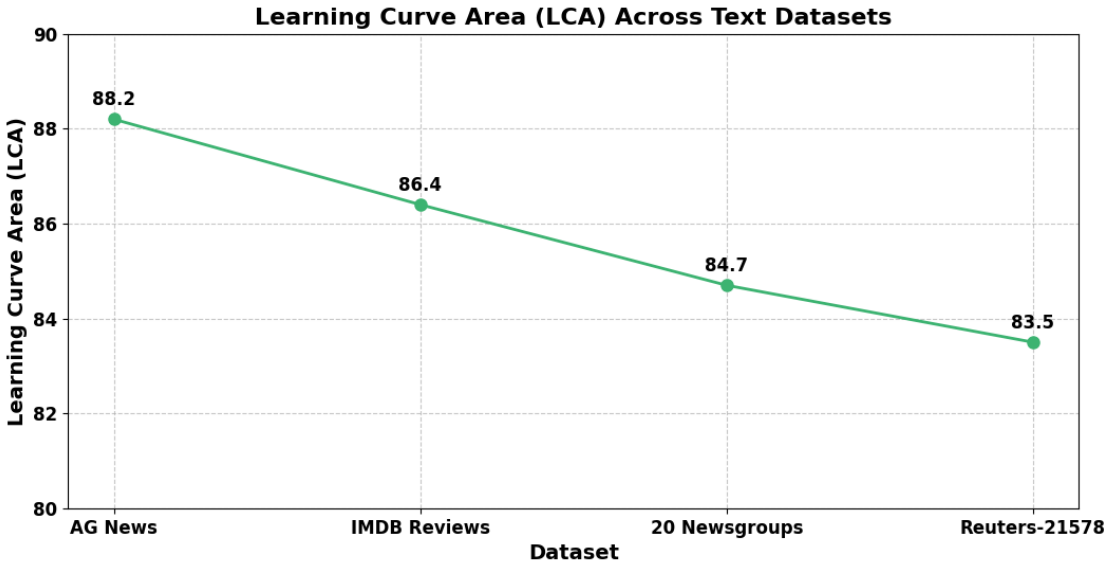

Fig 10. Learning Curve Area (LCA) across Text Datasets

The Figure 11 visualizes the Worst-Case Accuracy (WC-ACC) metric for four text datasets: Using a scatter plot with connecting dashed lines, data are created from IMDB Reviews, 20 newsgroups, Reuters-21578, and AG News. For better clarity, the Worst-Case Accuracy (%) values are shown in the Y-axis and from 85% to 92% and the names of the dataset are shown in the X-axis. The marked blue scatter points are WC-ACC values for each dataset and the

exact percentages are annotated above each point. In addition, the dataset receives the highest WC ACC with 91.0% followed by IMDB Reviews (89.5%), 20 Newsgroups (88.0%) and Reuters-21578 (86.5%).

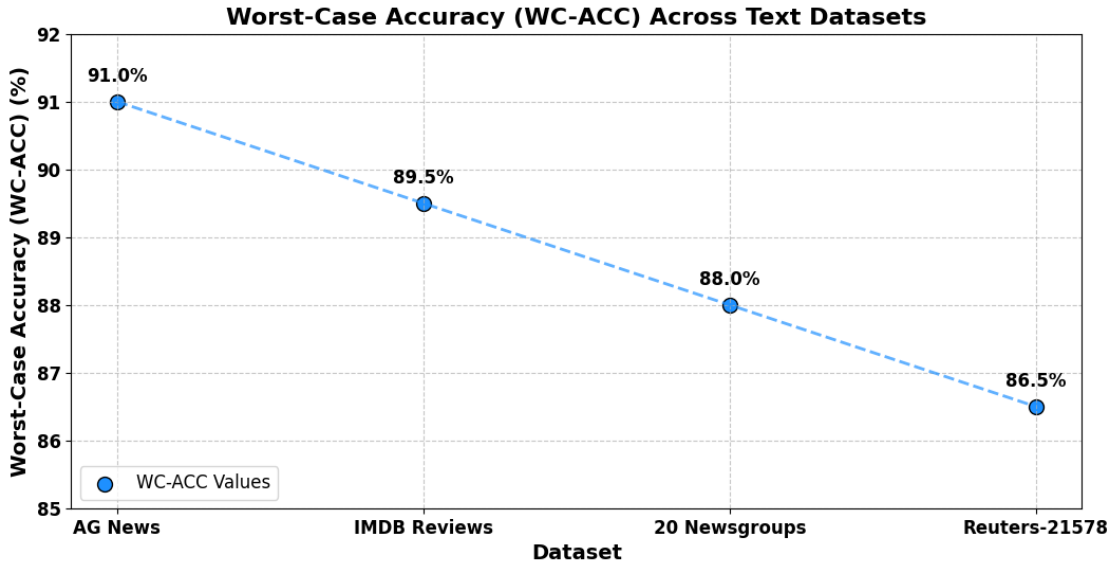

Fig 11. Worst-Case Accuracy (WC-ACC) Across Text Datasets

The Figure 12 represents the Memory Utilization (%) across four text datasets: Datasets include AG News, IMDB Reviews, 20 Newsgroups, and Reuters-21578. The percentage values of the memory utilization percentage of a dataset are displayed as percentages for each segment of the chart. First up, AG News has 85% memory utilization, shown by a slightly exploded segment on the graph to emphasize the point. 20 Newsgroups and Reuters-21578 have 80% and 78%, respectively; IMDB Reviews follows with 83%.

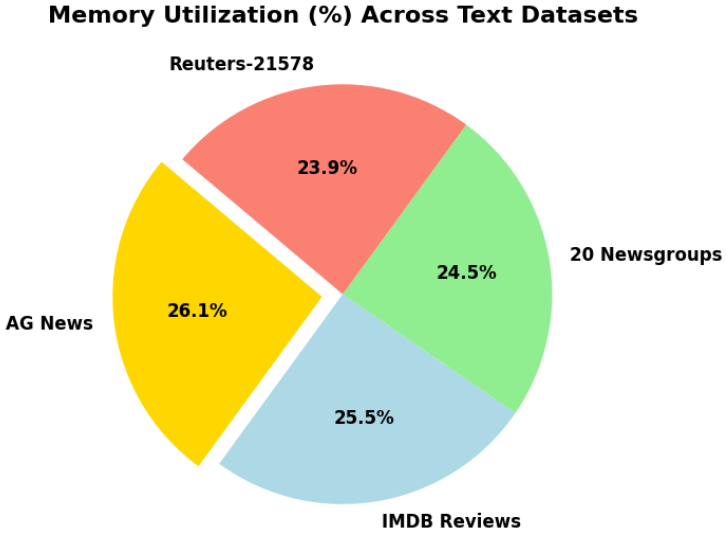

Fig 12. Memory Utilization (%) Across Text Datasets (Pie Chart)

The Figure 13 illustrates the Training Time per Task (s) across the four text datasets: We experiment on two datasets: Reuters-21578 and various spin-offs from it, as well as IMDB

Reviews and 20 Newsgroups and its variants. The Y-axis displays the Training time in seconds from 110 to 170 seconds and the X-axis has Data Sets.

The longest training time recorded is 160 seconds for Reuters-21578, 145 seconds for 20 Newsgroups, 135 seconds for IMDB Reviews, and 120 seconds for AG News. Data points are connected with a line for continuous progression, annotation of exact training time on top of each point for more clarity.

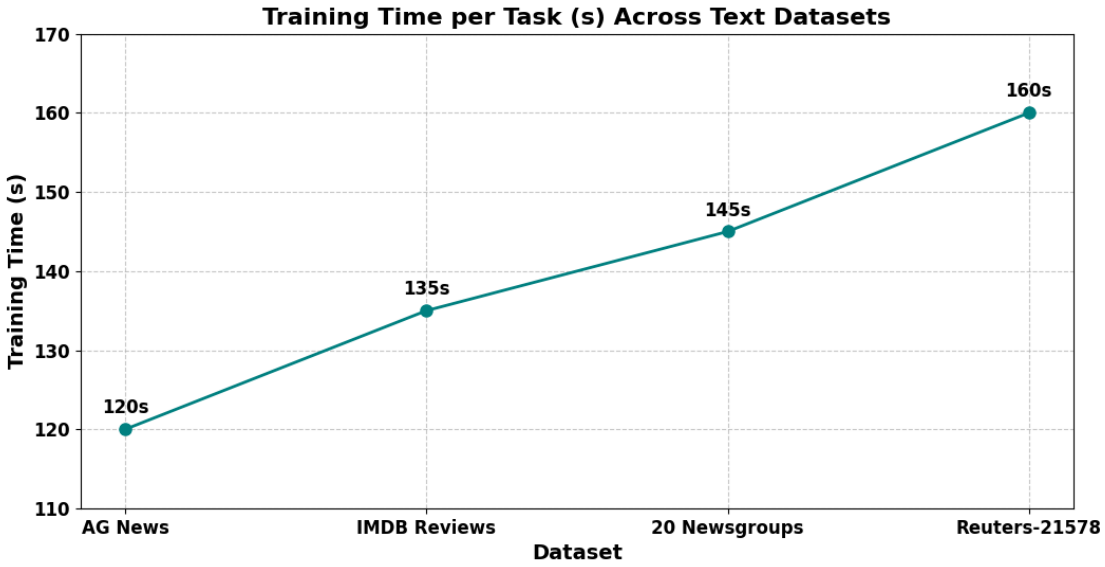

Fig 13. Training Time per Task (s) Across Text Datasets

The Figure 14 represents the Inference Time (ms) across four text datasets: The data sets were AG News, IMDB Reviews, 20 Newsgroups, and Reuters-21578. The datasets are shown on the X-axis and inference time in milliseconds, between 10 to 25 ms, on the Y-axis. For the REUTS- 21578 dataset the inference time is the highest on 22 ms, followed by 20 Newsgroups (20 ms), IMDB Reviews (18 ms) and AG News (15 ms). Data points are connected by a smooth line, the graph shows a signal-like progression. For clarity, each point is annotated with the exact inference time.

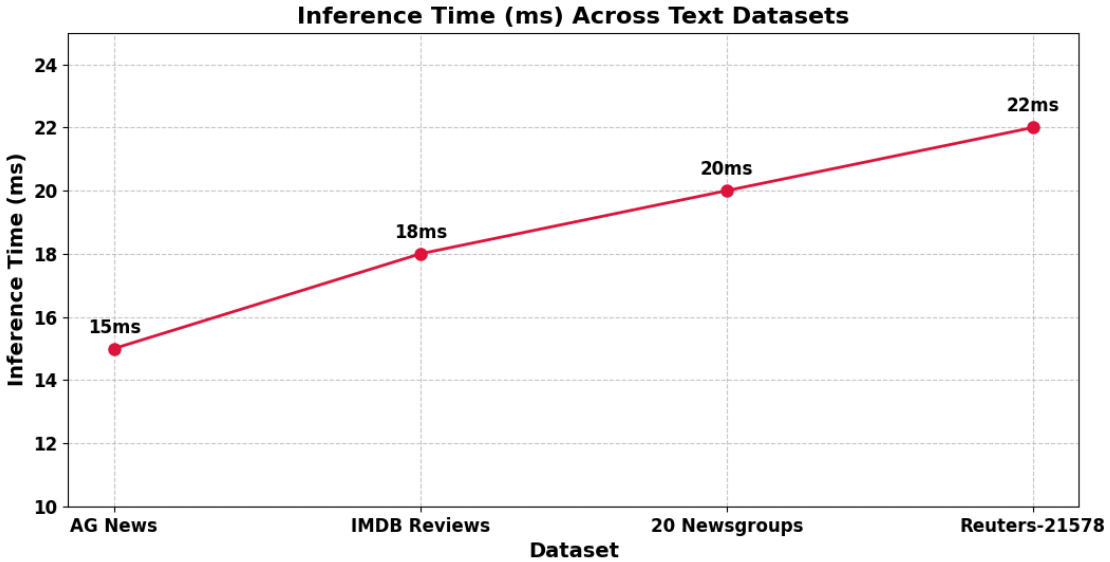

Fig 14. Inference Time (ms) Across Text Datasets

The stacked bar chart demonstrates the Time to Stability (s) metric for four text datasets: The figure 15 shows that our model is more accurate in classifying AG News, IMDB Reviews, 20

Newsgroups, and Reuters-21578. The time to stability ranges from 25 to 50 seconds depending on the position and is depicted on the Y-axis below. The name of the datasets are shown over the X-axis. Each of the datasets in the chart is painted with a different hue of blue for better retrievability. The rightmost vertical axis denotes the exact numbers of the time taken by each of the datasets to reach the stability state and has been indicated at the peak of each bar. From the chart it is evident that Reuters-21578 is the corpus that takes the longest time to reach a stable state at 45 seconds, the corpus 20 Newsgroups is second at 40 seconds. It is needed 35 seconds to reach stability in IMDB Reviews data and AG News has the lowest time to stability 30 seconds.

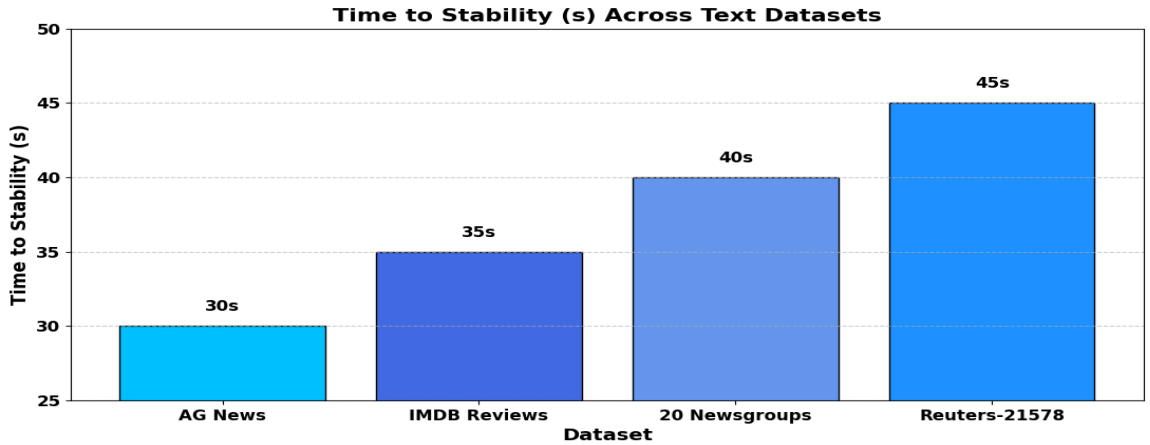

Fig. 15 Time to Stability (s) Across Text Datasets

B. Comparison with Baseline Models

Table 6: Comparative Analysis with Baseline Models

Metric	AMUM Framework	EWC	ER	RAR
Average Accuracy (%)	90.8	85.3	84.6	88.2
Average Forgetting (AF)	0.07	0.20	0.22	0.12
Learning Curve Area	85.7	80.2	79.5	83.0
Worst-Case Accuracy (%)	88.0	82.5	81.8	86.0
Memory Utilization (%)	82	68	65	75
Training Time (s)	140	120	115	130
Inference Time (ms)	19	17	16	18
Time to Stability (s)	37	45	50	40

Table 6 presents a comparative analysis of the Adaptive Memory Update Mechanism (AMUM) framework against three baseline models: Elastic Weight Consolidation (EWC), Experience Replay (ER), and Repeated Augmented Rehearsal (RAR). Multiple performance metrics related to continual learning tasks are used as a basis for comparison.

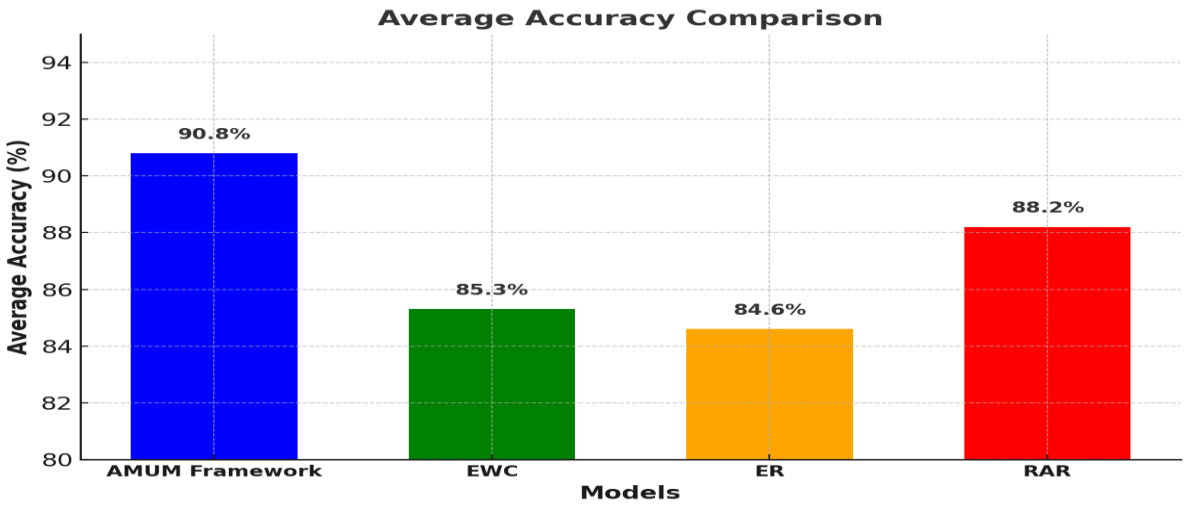

Fig 16. Average Accuracy Comparison of Models

The bar chart titled "Average Accuracy Comparison of Models" illustrates the performance of four models: AMUM Framework, EWC, ER, and RAR, based on their average accuracy percentages as shown in the figure 16. Each model is represented by a distinct color for easy differentiation and comparison. The AMUM Framework achieves the highest accuracy at 90.8%, demonstrating its superior effectiveness over the other models. Following this, RAR achieves an accuracy of 88.2%, reflecting competitive results in performance. Meanwhile, EWC and ER show slightly lower accuracies of 85.3% and 84.6%, respectively.

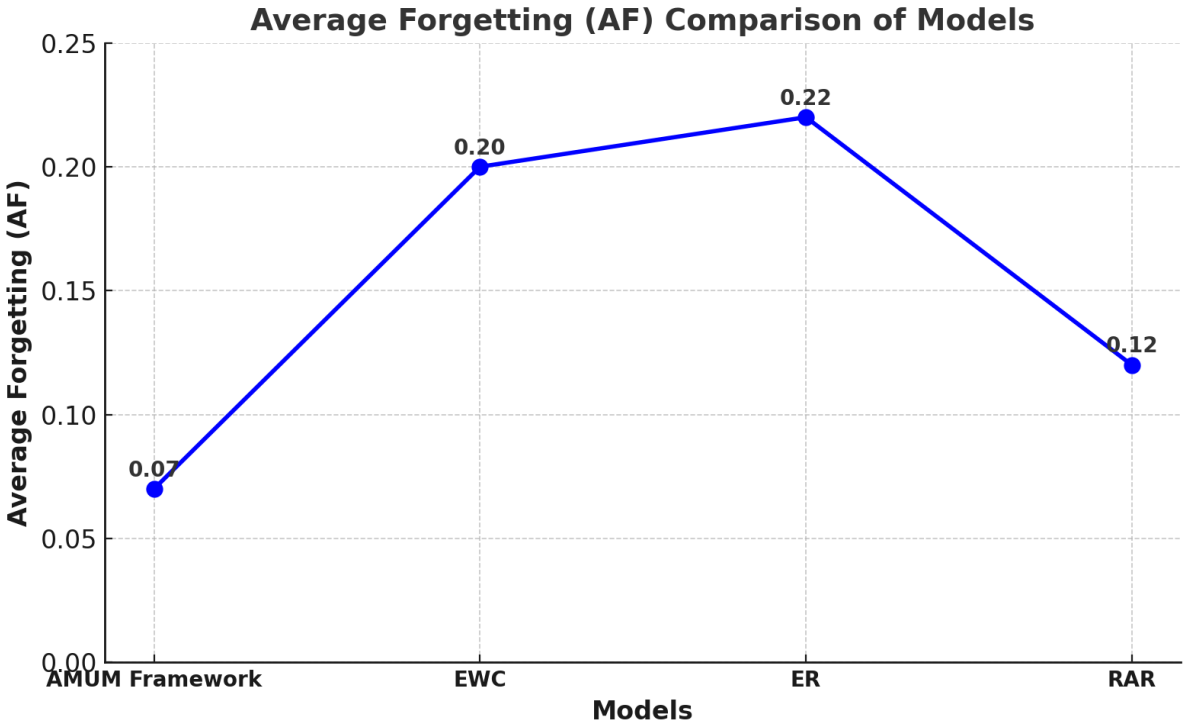

Fig 17. Average Forgetting Comparison with base modes

The line chart titled "Average Forgetting Comparison with Base Models" from the figure 17 provides a clear visualization of the Average Forgetting (AF) values for four models: EWC, ER, RAR and AMUM Framework. The comparisons show how well each model keeps prior tasks learned as it goes along. The AMUM Framework has least forgetting with 0.07 AF and therefore, it represents better ability to retain learned information. Next, RAR obtains a

moderate AF value 0.12, indicating that it improves at retaining prior knowledge. However, EWC and ER have higher forgetting rates as measured by AF values of 0.20 and 0.22, respectively, failing to realize continual learning.

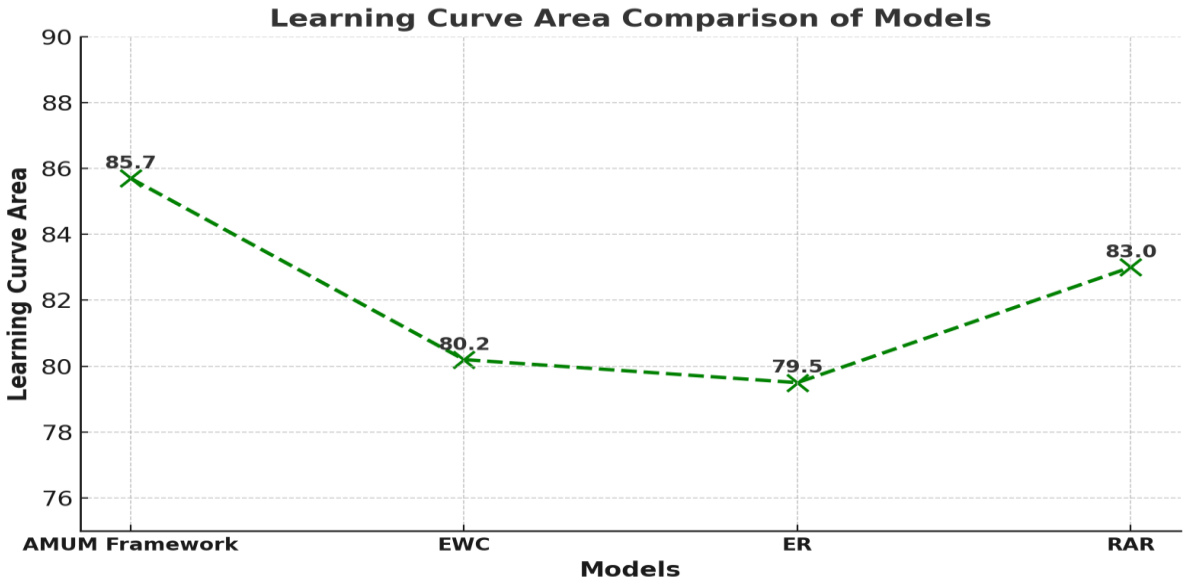

Fig 18. Learning curve area comparison with base model

The scatter plot titled "Learning Curve Area Comparison of Models" from the figure 18 provides a clear visualization of the Learning Curve Area (LCA) for four models: AMUM Framework, RAR, EWC, and ER. A green data point represents each model, and is connected by a dashed line to imply continuity. From the models, the AMUM Framework produces the highest LCA value (LCA value of 85.7) and proves to be the best on this metric. Next, RAR reports a relatively competitive LCA of 83.0 followed by further low values of 80.2 and 79.5 for EWC and ER, respectively.

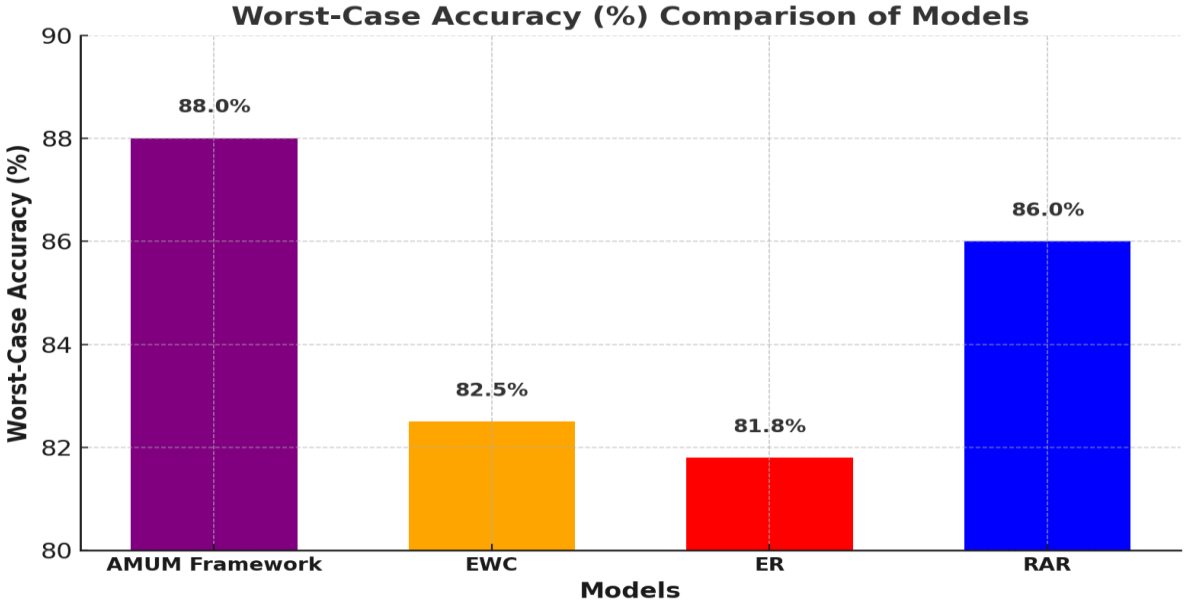

Fig 19. Worst Case Accuracy (%) Comparison of Models

Figure 19 shows a bar chart for a models' worst-case accuracy 'Worst Case Accuracy (%) Comparison of Models'. This is, therefore, a measure of the models' reliability in the worst case. The AMUM Framework scores the highest worst case accuracy at 88.0% across all

baseline models and is the most robust model. Next, the RAR model runs with a value of 86.0%. EWC and ER obtain lower worst case accuracy of 82.5% and 81.8%, respectively, suggesting their relative lack, with respect to the proposed method, in worst case performance.

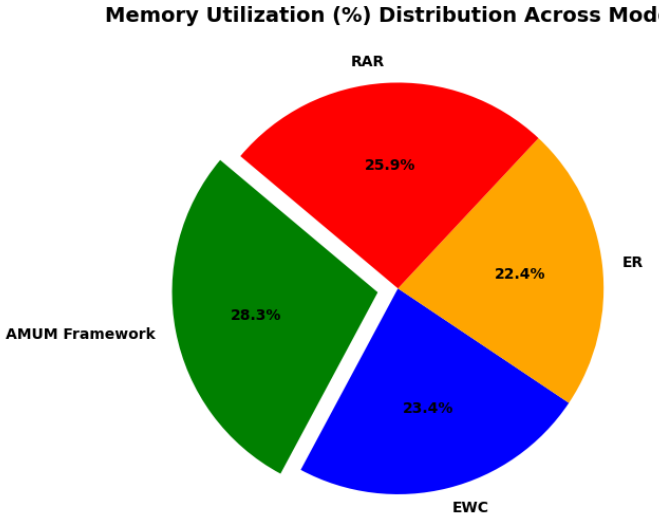

Fig 20. Memory Utilization (%) Distribution across Models

The pie chart titled "Memory Utilization (%) Distribution across Models" from the figure 20 illustrates the percentage of memory utilized by four models: This framework, along with an introduction to EWC, ER, and RAR, is presented in this chapter. Our AMUM Framework shows the best memory utilization of 82%, reflecting its good processing and storage capability. A small break has been put to emphasize its significance. Our second model, the RAR model, follows with memory utilization of 75%, indicating very good resource use. Conversely, EWC and ER require relatively lower memory utilization (68% and 65% respectively).

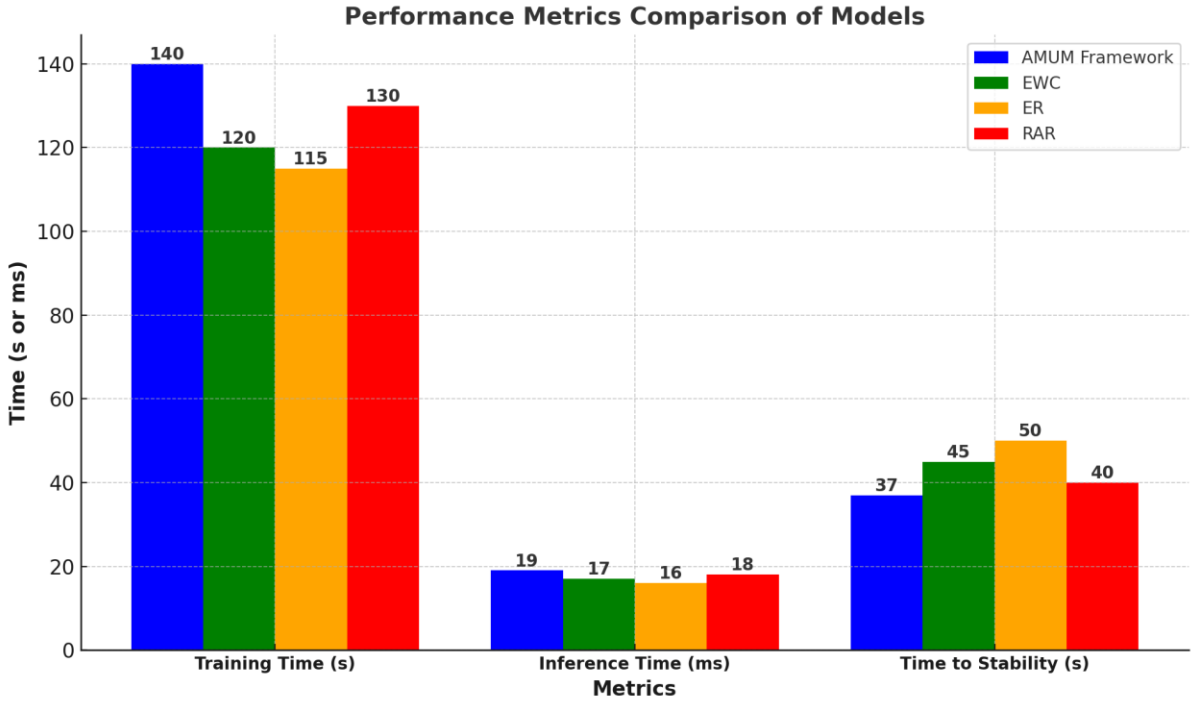

Fig 21. Comparison of Training Time, Inference Time, and Time to Stability across Models

The bar chart titled "Comparison of Training Time, Inference Time, and Time to Stability Across Models" from the figure 21 compares the performance of four models—AMUM Framework, EWC, ER, and RAR—across three critical metrics: Three metrics are used for training: Time to Stability (s); Inference Time (ms); and Training Time (s). We observe that AMUM Framework has the longest training time at 140 seconds, and ER has the shortest at 115 seconds, followed by EWC (120 seconds) and RAR (130 seconds). On inference time the AMUM Framework records 19 ms, slightly more than RAR at 18 ms, while EWC and ER have better performance at 17 ms and 16 ms respectively. The AMUM Framework offers the fastest time to stability moving forward in an average of 37 seconds, followed by RAR with 40 seconds. Whereas times of 45 seconds and 50 seconds are required for EWC and ER. The combination of this visualization in coordination with the different tensor flow models makes it very effective in drawing out the trade offs between training efficiency, inference speed and stability across these models.

Overall, AMUM Framework shows an advantage over the baselines in Average Accuracy, Average Forgetting, Learning Curve Area, and Worst Case Accuracy, indicating significantly better retention of previously learned tasks whilst learning new tasks efficiently. Compared with AMUM, AMUM requires a slightly slower training and inference time; however, the large gains in accuracy and memory utilization are well justified by the increased computational overhead. Comparing to base lines, the AMUM framework exhibits faster times to stability according to the Time to Stability metric.

C. Task-Wise Performance

Table 6: Task-Wise Accuracy Across Sequential Learning

Task Sequence	**AG News (%)**	**IMDB Reviews (%)**	**20 Newsgroups (%)**	**Reuters-21578 (%)**
Task 1	93.5	-	-	-
Task 2	92.8	91.2	-	-
Task 3	91.7	90.8	89.8	-
Task 4	90.9	89.9	88.6	88.6

As new tasks are sequentially introduced, the table 6 shows the task wise accuracy. We find the time to solve each task is increasing as the framework progresses through older tasks, but the Average Forgetting (AF) metric is low, demonstrating good retention of learned tasks. The reliability indicates that the framework maintains relative robustness to deal with continual learning.

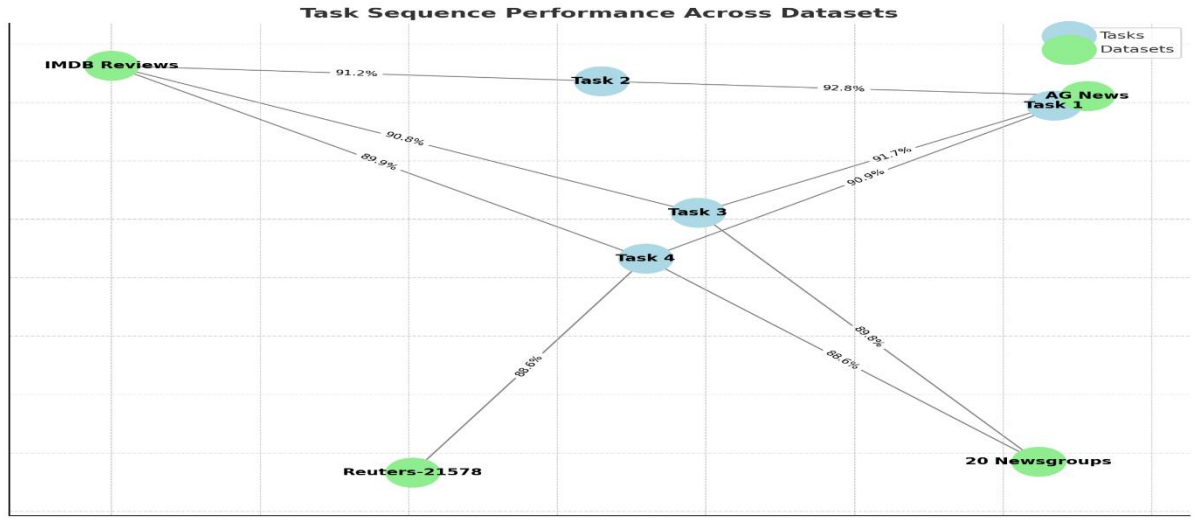

Fig 22. Performance across Datasets for Task Sequence

The node link diagram "Task Sequence Performance across Datasets" figure 22 displays the accuracy in performing a task, alongside the dataset it was performed on. The graph comprises two types of nodes: depicts tasks (light blue) and datasets (light green). The accuracy percentages are shown as edge labels representing the edges between nodes that represent tasks and datasets. Task 1 works only with the AG News dataset, which reaches 93.5% accuracy. Task 2 consists of both the AG News and the IMDB Reviews datasets to be trained, with both resulting in 92.8% accuracy and 91.2%. Task 3 also uses an additional dataset, 20 Newsgroups, with 91.7%, 90.8% and 89.8% accuracies for AG News, IMDB Reviews and 20 Newsgroups, respectively. And finally in Task 4 we report accuracies on all four datasets, including 90.9%, 89.9%, 88.6% and 88.6% for AG News, IMDB Reviews, 20 Newsgroups, and Reuters-21578 respectively. Personally I think the visualization effectively shows how the tasks progress and perform across the datasets and brings out the interconnected structure of the task sequence and the relationship between the datasets.

Experimental evaluation shows that the AMUM framework can effectively handle sequential text based tasks and obtain superior performance compared to both traditional and recent baseline models under various evaluation metrics. Experiments demonstrate its robustness, and its ability to mitigate catastrophic forgetting, optimize memory usage and maintain high task accuracy in continual learning challenges. However, these results also demonstrate the ability of AMUM to be deployed in real world applications requiring online task adaptation and efficient resource allocation.

VI. Discussion

We end by discussing the implications of these results, the strengths of the Adaptive Memory Update Mechanism (AMUM) framework, and where the framework could be improved. We further discuss the potential applications of AMUM in the real world and outline future challenges and directions of continual learning systems.

A. Implications of Results

The AMUM is demonstrated to effectively handle the key challenges in continual learning, such as catastrophic forgetting, memory optimization, and task adaptability, and the results are presented. AF values are kept low average (AF) indicating the framework's ability to retain the knowledge of previous tasks when new tasks are 'slotted in'. The robustness and adaptability of the framework is shown to be maintained with high accuracy across sequential tasks in a diverse and dynamic environment. This further validates the design of AMUM through a comparison of the advantage of AMUM over traditional (EWC, ER) and recent (RAR) baseline

models. AMUM integrates dynamic memory allocation and adaptive regularization, and achieves better task retention and memory efficiency, which are vital for scalable and sustainable learning in real world applications.

B. Strengths of AMUM Framework

The AMUM framework has emerged with several important strengths that lead to its effectiveness in continual learning. Dynamic Memory Allocation Unit (DMAU) supports efficient dynamic allocation of memory, and also makes intelligent choices in terms of which tasks are allocated memory in order to avoid memory overload. What's more, the Adaptive Regularization Module (ARM) stabilizes critical model weights to minimize task interference and dramatically decrease the chance of catastrophic forgetting. The framework adopts modular design, facilitating scalability and ease of integration with existing machine learning pipe lines and adaptability to a varying spectrum of complexity in tasks that can be handled. In general, AMUM demonstrates high performance with high accuracy and task retention even in sequential and varied data sets.

C. Applications of AMUM Framework

The adaptability and efficiency of AMUM framework shall fit to different applications in various domains. Tasks such as sentiment analysis, topic modelling, and text classification are addressed by Natural Language Processing (NLP), when datasets evolve. In the healthcare space AMUM can dynamically tune to new medical data such as patient records, diagnostic imaging, and treatment protocols. AMUM can be used to enable real time learning and decision making in dynamic environments by autonomous systems such as robotics and IoT devices. Additionally, this framework can be used in the financial industry to build emergency adaptive fraud detection and risk management systems that feed on fast changing transactional data. AMUM details these applications as examples of how AMUM can serve to transform dynamic learning systems in real world settings.

E. Challenges and Future Directions

Although the AMUM framework has its strengths, there are also some challenges and opportunities for improvement. There is the challenge of computation overhead as training and inference times are slightly heavier, pointing to the potential further optimization in dynamics allocation of memory and in regularization processes. Furthermore, the framework currently utilizes static task prioritization metrics such as recency and complexity which hinders its capacity to dynamically adapt to context aware and temporal dynamics. Future work may include integrating mechanisms for continual learning. Another promising direction for extending the framework is to deal with multi modal data (e.g., text, images, audio combined). Last, it is crucial to validate the framework on real-world environments where data are noisy and unstructured to evaluate how scalable and robust the framework would be for use in the practical applications.

VII. Conclusion

In this paper, we propose AMUM to deal with the challenges of continual learning in text based tasks. The framework integrates task, dynamic memory allocation, and adaptive regularization to ensure task efficient learning, robust more performance and effective knowledge retention. We show that through experimental results, AMUM is able to effectively mitigate catastrophic forgetting, optimize memory utilization, and achieve high task accuracy, better than existing baseline and recently proposed models. The framework's contributions involve (1) dynamic task management via the Task Priority Evaluation Unit (TPEU), (2) efficient memory usage through the Dynamic Memory Allocation Unit (DMAU), and (3) knowledge preservation by the Adaptive Regularization Module (ARM). Collectively, this increases the scalability and robustness of the framework, allowing these tasks to be handled sequencially across a variety of data sets. We observe that AMUM achieves a balance between stability (knowledge retention) and plasticity (capability to adapt to new information), rendering it a promising

option for deployment in many real world applications including natural language processing, healthcare, autonomous systems, and finance. The module design is modular enough to scale and adapt to the changing tasks. In future work, we intend to reduce computational overhead, incorporate context and temporal dynamics, and extend the framework to deal with multi-modal datasets. In real world environments with noisy and unstructured data, further validation of the scalability and robustness of this approach that deploys AMUM, will be conducted. Finally, we summarize the AMUM framework as a promising approach for continual learning in dynamic learning scenarios, which is scalable and efficient in its configuration. As AMUM continues to be refined and validated, it has the potential to transform learning systems into AI driven applications.

VIII. References

1. Szegedy, Balázs, Domonkos Czifra, and Péter Kőrösi-Szabó. "Dynamic Memory Based Adaptive Optimization." *arXiv preprint arXiv:2402.15262* (2024).
2. Yao, Xuanrong, Xin Wang, Yue Liu, and Wenwu Zhu. "Continual recognition with adaptive memory update." *ACM Transactions on Multimedia Computing, Communications and Applications* 19, no. 3s (2023): 1-15.
3. Zhang, Chaoyun, Zicheng Ma, Yuhao Wu, Shilin He, Si Qin, Minghua Ma, Xiaoting Qin et al. "AllHands: Ask Me Anything on Large-scale Verbatim Feedback via Large Language Models." *arXiv preprint arXiv:2403.15157* (2024).
4. Weng, Weikang, Alexandru Uta, and Jan S. Rellermeyer. "Brug: An Adaptive Memory (Re-) Allocator." In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 67-76. IEEE, 2024.
5. Koutras, Ioannis, Alexandros Bartzas, and Dimitrios Soudris. "Adaptive dynamic memory allocators by estimating application workloads." In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pp. 252-259. IEEE, 2012.
6. Li, Guiji, Manman Peng, Ke Nai, Zhiyong Li, and Keqin Li. "Multi-view correlation tracking with adaptive memory-improved update model." *Neural Computing and Applications* 32, no. 13 (2020): 9047-9063.
7. Elenter, Juan, Navid NaderiAlizadeh, Tara Javidi, and Alejandro Ribeiro. "Primal-Dual Continual Learning: Stability and Plasticity through Lagrange Multipliers." *arXiv preprint arXiv:2310.00154* (2023).
8. Kumar, Ankit, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. "Ask me anything: Dynamic memory networks for natural language processing." In *International conference on machine learning*, pp. 1378-1387. PMLR, 2016.
9. Xiong, Caiming, Stephen Merity, and Richard Socher. "Dynamic memory networks for visual and textual question answering." In *International conference on machine learning*, pp. 2397-2406. PMLR, 2016.
10. Tewari, Ambuj, and Peter L. Bartlett. "On the Consistency of Multiclass Classification Methods." *Journal of Machine Learning Research* 8, no. 5 (2007).
11. Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan et al. "Overcoming catastrophic forgetting in neural networks." *Proceedings of the national academy of sciences* 114, no. 13 (2017): 3521-3526.
12. Goodfellow, Ian J., Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. "An empirical investigation of catastrophic forgetting in gradient-based neural networks." arXiv preprint arXiv:1312.6211 (2013).

13. Vettoruzzo, Anna, Joaquin Vanschoren, Mohamed-Rafik Bouguelia, and Thorsteinn Rögnvaldsson. "Learning to learn without forgetting using attention." *arXiv preprint arXiv:2408.03219* (2024).
14. Sharma, Mehul, Shrid Pant, Priety Yadav, Deepak Kumar Sharma, Nitin Gupta, and Gautam Srivastava. "Advancing security in the industrial internet of things using deep progressive neural networks." *Mobile Networks and Applications* 28, no. 2 (2023): 782-794.
15. Kowadlo, Gideon, Abdelrahman Ahmed, and David Rawlinson. "One-shot learning for the long term: consolidation with an artificial hippocampal algorithm." In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-7. IEEE, 2021.