Incremental Caching of Large Language Model Requests

Shriram Piramanayagam

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

Organizations often use Large Language Model (LLM) APIs for tasks like customer support, which can be expensive and pose data privacy risks. To mitigate this, a neural caching framework that uses a smaller, locally run *student* model trained on the LLM's outputs has been proposed and studied by previous literature (Ramírez et al., 2023). However, this research performed a complete retraining of the student model to update it, which is inefficient. In this work, we explore a range of Incremental Learning (IL) techniques for efficient neural caching with a focus on classification tasks. IL allows models to adapt to new information without forgetting previously learned knowledge. Our experiments suggest that the IL method, Replay improves the computational efficiency of neural caching by reducing FLOPs and training time by approximately 40% in static data streams and 30% in dynamic data streams compared to complete retraining with minimal performance tradeoffs.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Shriram Piramanayagam*)

Acknowledgements

I would like to thank my supervisor Prof. Ivan Titov for his invaluable guidance and support throughout the dissertation process. I also want to express my gratitude to my PhD tutor, Guillem Ramírez Santos for his guidance throughout the process and patiently answering all my questions. Additionally, I want to express my gratitude to my family and friends for their love and support throughout this journey.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Large Language Models (LLMs) possess advanced capabilities in understanding natural language. They undergo pre-training on an extensive corpus of text data, during which they acquire knowledge of various language patterns and structures. After pre-training, these models are fine-tuned for tasks such as text classification, generation or question answering. However, the financial cost of training and maintaining LLMs is significant, making it inaccessible for many organizations. LLMs can perform tasks without prior task-specific training by understanding the context from the input, making them versatile and adaptable across various applications.

LLM subscriptions allow organizations to use LLMs through APIs, with charges applied per call. For instance, customer support chatbots in organizations leverage LLMs to understand user intentions and offer appropriate assistance (Ham et al., 2020). As time goes on, the prompts provided to the LLM may become repetitive. Moreover, this approach also reveals the organization's entire request stream to the LLM provider. To tackle cost and data privacy concerns, a new framework has been studied where a smaller, locally run model, called the *student* model, is periodically trained using outputs from the LLM, or *teacher* model (Ramírez et al., 2023). This method, referred to as *neural caching*, enables the student model to manage requests on its own, cutting costs while maintaining similar performance.

The key element of neural caching is a policy or instance selection criteria that determine whether a user request should be processed by the student or the teacher. Ramírez et al. (2023) shows the efficacy of Active Learning (AL) (Settles, 2009) based selection criteria for neural caching on a stationary or static (i.i.d) stream of requests.

However, this study updated the student model with complete re-training as new LLM annotated data arrived, which is suboptimal. Additionally, in real-world scenarios, the distribution of requests is dynamic and is likely to change over time (Cacciarelli and Kulahci, 2023; Gama et al., 2014). Incremental Learning (IL) explores methods that strike a better balance between performance and efficiency, where models adapt to new data without forgetting previous knowledge. Thus, in this work, we aim to study IL methods to perform neural caching efficiently for both static and dynamic data streams. We also aim to study the effect of various policies for neural caching on a dynamic stream of requests.

Our incremental neural caching process begins with a student model generating a response to an incoming query. The policy determines whether to use the student's response or to invoke an LLM. LLM responses are stored in a cache and later used to incrementally distil the LLM's knowledge to the student model. Figure 1.1 describes the incremental neural caching process.

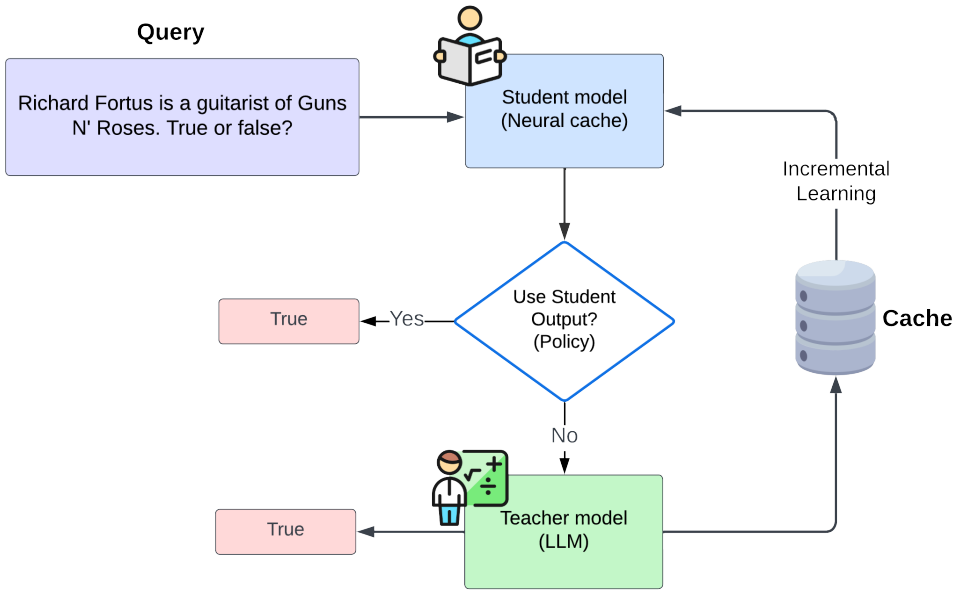

Figure 1.1: A single iteration of Incremental Neural Caching.

1.2 Objectives

The objectives of the project mainly include:

- To study the use of Incremental Learning (IL) methods in the context of neural caching for Large Language Model (LLM) requests, applying incremental online

knowledge distillation to reduce expensive LLM calls.

- To evaluate the efficiency and performance trade-offs of incremental learning approaches in neural caching across static and dynamic data streams.
- To evaluate and analyze the performance of various policies including AL-based policies for incremental neural caching.

1.3 Key Contributions

This research makes several key contributions to the field of incremental learning and neural caching.

- We introduce a novel application of incremental learning for neural caching of LLM requests.
- We show that incremental learning approaches offer significant efficiency gains with minimal performance trade-offs.
- Our findings reveal that the IL method, Replay is particularly effective for incremental learning in neural caching of static and dynamic data streams.
- The research empirically proves that AL-based policies improve performance over baseline policies for incremental neural caching of static and dynamic data streams.

1.4 Structure of the research

The dissertation is structured as follows:

1. **Introduction**: This chapter introduces the motivation behind the research on incremental neural caching and outlines the key objectives and contributions of the research.
2. **Background**: This chapter provides a review of relevant literature, including an overview of neural caching and various instance selection criteria, as well as continual and incremental learning methods.

3. **Description of the Work Undertaken**: This chapter details the conceptual design and implementation of the project. It covers the neural caching framework, the incremental learning methods employed, the datasets used, the implementation specifics of the experiments and the evaluation criteria.

4. **Experiments**: This chapter presents the experiments conducted on both static and dynamic data streams, outlining the motivation behind each experiment, followed by the results and their subsequent discussions.

5. **Conclusions**: The final chapter summarizes the research findings, discusses the limitations of the research, and offers suggestions for future work.

Chapter 2

Background

This chapter reviews the background literature relevant to the research. Section 2.1 introduces neural caching and explores various instance selection criteria discussed in the literature. Section 2.2 covers continual and incremental learning, along with an overview of IL methods.

2.1 Neural caching

The goal of neural caching is to enhance the system's online accuracy while effectively training the student model. The implementation of neural caching by Stogiannidis et al. (2023) uses a kNN classifier as the *student* model. This model is simple since it does not require retraining. As the dataset expands, the kNN classifier's requirement to access the entire dataset for each decision presents substantial memory challenges, highlighting the trade-off between simplicity and scalability in the model. Ramírez et al. (2023) assesses various active learning-based selection criteria for the policy, with Margin Sampling and Query by Committee demonstrating consistent benefits across tasks and budgets. Budget is the maximum number of API calls the student model can make to the LLM. The paper shows the efficacy of AL-based policies over static policies like Front-loading. In this context, the neural caching setup is an online form of Active Learning with Knowledge Distillation. Both Active Learning (Settles, 2009) and Knowledge Distillation (Bucilua, Caruana, and Niculescu-Mizil, 2006; Hinton, Vinyals, and Dean, 2015) aim to optimize the final accuracy of the student model, whereas the neural caching problem seeks to optimize the online accuracy of the entire setup.

Knowledge distillation (KD): KD (Bucilua, Caruana, and Niculescu-Mizil, 2006; Hinton, Vinyals, and Dean, 2015) involves training a smaller model on the larger

model's predictions to mimic it. Here, LLM's knowledge is continuously transferred to the *student* model.

Active Learning (AL) AL (Settles, 2009), primarily involves the selection of the most informative data samples for training, which human experts typically annotate. The neural caching framework uses a variant of active learning known as stream-based selective sampling. This method examines each consecutive unlabeled instance individually, allowing the *student* to assign a label or query the *teacher* for each data point.

2.1.1 Instance Selection Criteria

Active Learning (AL) metrics have already been shown to work well as instance selection criteria (policy) for neural caching (Ramírez et al., 2023). The remainder of this section outlines the AL metrics studied by Ramírez et al. (2023), which are followed in this work as well.

Front-loading (FR) involves using the entire annotation budget at the start by labeling all instances with the Large Language Model (LLM). After the budget is exhausted, the student model handles all subsequent queries independently.

Margin Sampling (MS) (Scheffer, Decomain, and Wrobel, 2001) focuses on selecting examples where the difference between the probabilities of the top two predicted labels by the student model is the largest. The margin is calculated as:

$$\text{Margin}(x_i) = \log P(y_i = k_1^* | x_i) - \log P(y_i = k_2^* | x_i) \tag{2.1}$$

where k_1^* and k_2^* represent the first and second most likely labels, as determined by the probability distribution $P(y_i|x_i)$ calculated by the student model. Recent evaluations by Schröder, Niekler, and Potthast (2021) have confirmed the efficacy of MS, particularly highlighting its superior performance with Transformer models (Devlin et al., 2019) in an offline, pool-based setting.

Prediction Entropy (PE) (Roy and McCallum, 2001) is one of the most commonly used uncertainty sampling strategies in active learning. The key idea is to select instances that have the highest entropy in their predicted output distribution, as these are likely to be the most informative for the model since the model is uncertain about these instances. Prediction entropy is calculated as:

$$\text{Entropy}(x_i) = -\sum_j P(y_i = k_j^* \mid x_i) \log P(y_i = k_j^* \mid x_i) \tag{2.2}$$

Query by Committee (QBC) selects instances based on the level of disagreement among a committee of models (Seung, Opper, and Sompolinsky, 1992). The committee consists of four previous student models in addition to the current student model, which is presumed to be the best performing. The key idea behind QBC is that instances, where committee members disagree, are likely to be informative, as they represent areas of uncertainty in the model space. The disagreement is measured by calculating the percentage of committee members who oppose the current student's response.

Coreset (CS) is an active learning approach that aims to select a subset of data points that approximates the full dataset while minimizing redundancy and maximizing representativeness (Sener and Savarese, 2018). To implement Coreset in neural caching setup, an encoder is used to obtain the embedding representation of the new instance. Next, the cosine similarity between the new input's embedding and the embeddings of previous examples is computed. If the similarity with the most similar past instance x_i annotated by the LLM falls below a specified threshold s, a new annotation is requested from the LLM. To obtain the embeddings, an average of the encoder representation across tokens is usually taken, as this has been proven effective in sentence embedding benchmarks (Ni et al., 2022).

2.2 Continual Learning

Continual learning (CL) allows machine learning models to adapt continuously to new data and enables learning from a stream of information. The main challenge CL addresses is catastrophic forgetting (McCloskey and Cohen, 1989), which occurs when a neural network loses old knowledge as it learns new information. CL is typically designed to address challenges in environments where data distribution changes over time, which leads to catastrophic forgetting in the model. Neural networks are especially prone to this issue, often overfitting to recent data at the expense of earlier knowledge (Parisi et al., 2019). The current study does not explicitly focus on these dynamic distributions. However, we speculate that the advantages of CL and Incremental Learning (IL) approaches, which mitigate catastrophic forgetting, would be beneficial for efficient caching of static and dynamic data streams.

There are two basic approaches to learning with a continuous stream of data (Verwimp et al., 2023). The first, incrementally training or fine-tuning a model only with new data, often leads to suboptimal results as models may overly adapt to the latest data. The second approach involves retraining a model on all accumulated data. However, this

is undesirable due to high computational and memory costs. Existing neural caching frameworks were only evaluated by retraining the student model from scratch as new data became available. Continual Learning (CL) explores methods that strike a better balance between performance and efficiency.

As noted by Harun et al. (2023), most continual learning techniques are not primarily designed to optimize computational efficiency. However, they demonstrate that it is possible to reduce training times significantly while maintaining comparable performance levels. Effective implementation of continual learning techniques could drastically cut both the financial costs and the substantial carbon emissions typically associated with retraining models from scratch (Amodei and Hernandez, 2023), without sacrificing accuracy.

2.2.1 Incremental Learning

Incremental learning is a crucial aspect of continual learning, where models adapt to new information without forgetting previously learned knowledge. This approach is particularly relevant in scenarios where data arrives sequentially and complete retraining is impractical due to computational or memory constraints (Parisi et al., 2019). Incremental learning is more narrowly focused on updating models on batches of new data incrementally in an efficient way, while continual learning encompasses a broader range of scenarios, including domain shifts, task adaptation and lifelong learning in non-stationary environments.

The primary goal of incremental learning is to prevent catastrophic forgetting and ensure the model can learn over time without requiring full retraining. However, in our neural caching setup, the emphasis is on maintaining reasonable online and final accuracy of the student model. While incremental learning may not always achieve the highest accuracy, its strength lies in its ability to adapt to new data efficiently, which is crucial in real-world applications. Incremental learning methods can be broadly classified as follows (Biesialska, Biesialska, and Costa-Jussa, 2020):

Regularization-based methods: These approaches add constraints to balance the performance in old and new tasks. These approaches involve adding an extra term to the loss function during training to penalize changes to model parameters that are crucial for previous tasks. A common implementation involves adding a quadratic penalty to the loss function, penalizing the variation of each parameter based on its contribution or ”importance” to performing old tasks. The importance can be computed

using the Fisher information matrix (FIM), such as in Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017). EWC is a parameter regularization-based approach for incremental learning. EWC helps neural networks preserve prior knowledge by slowing down updates to weights that are crucial for previous tasks. EWC supports continual learning in neural networks, allowing them to learn new tasks sequentially without forgetting previously learned tasks. EWC uses the following loss function:

$$L = L_B + \sum_i \frac{\lambda}{2} F_{ii}(\theta_i - \theta^*_{A,i})^2$$

where L_B is the loss for the new task, λ is a hyperparameter, F_{ii} is the diagonal of the Fisher Information Matrix, and $\theta^*_{A,i}$ are the optimal parameters for the previous task.

Rotated EWC (R-EWC) (Liu et al., 2018) improves EWC by rotating the parameter space to better diagonalize the Fisher Information Matrix. Additionally, numerous efforts have focused on designing better online importance measures. For instance, Synaptic Intelligence (SI) (Zenke, Poole, and Ganguli, 2017) estimates parameter importance by tracking their contribution to the total loss change. Memory Aware Synapses (MAS) (Aljundi et al., 2018) measures parameter importance based on output function sensitivity. RWalk (Chaudhry et al., 2018) combines the regularization terms of SI (Zenke, Poole, and Ganguli, 2017) and EWC (Kirkpatrick et al., 2017) to integrate their advantages. Interestingly, these importance measurements have been shown to approximate the FIM (Nguyen et al., 2018), despite originating from different motivations.

Rehearsal Methods: These methods maintain a memory of past data samples, which are periodically replayed to the model while learning new tasks. This helps the model retain knowledge from previous tasks. Variants of this approach include pseudo-rehearsal methods, which generate synthetic data resembling past experiences instead of storing actual data. iCaRL (Rebuffi et al., 2017) combines replay with distillation to maintain performance on old tasks while learning new ones. The paper presents a method for class-incremental learning that prevents catastrophic forgetting by combining representation learning with a nearest-mean-of-exemplars classification strategy. iCaRL stores a fixed number of exemplars from past classes and uses these to update the model as new classes are introduced. The approach integrates knowledge distillation and exemplar rehearsal to maintain accuracy across both new and previously learned classes.

LAMOL (Sun, Ho, and Lee, 2020) enables a language model to learn new tasks

while generating pseudo-data from previous tasks, effectively using these pseudo-samples to replay old tasks and maintain performance across multiple tasks. Masson D'Autume et al. (2019) uses an episodic memory model that performs sparse experience replay and local adaptation for CL in text classification and question answering.

Memory replay or simply Replay involves storing and replaying a subset of past data. By preserving a memory of past experiences, replay enables incremental learning while avoiding catastrophic forgetting of previously acquired knowledge (Hayes et al., 2020). To perform continuous learning without resetting the model and to preserve knowledge across the entire data stream, a replay buffer is used to store a subset of previously seen examples. The buffer size can fixed to a percentage of the total budget. When the buffer reaches its capacity, an eviction policy can used to remove older or less important examples. Random works well as the eviction policy based on previous research (Chaudhry et al., 2018; Hayes et al., 2020; Wu et al., 2019). The size of the replay buffer can be tuned to balance memory usage, computation time, and preservation of past task performance.

Architectural methods: These involve altering the model's architecture to prevent forgetting. One common strategy is to freeze the parameters related to earlier tasks while allowing new parameters or modules to be added for subsequent tasks. Progressive Neural Networks (Rusu et al., 2016) is an example of this strategy, where new neural "columns" are added for new tasks while freezing previous ones.

Having discussed the theoretical background, the next chapter will explain the design and implementation details of the research.

Chapter 3

Description of the work undertaken

This chapter outlines the conceptual design and implementation details of the research. Section 3.1 introduces the neural caching problem, followed by Section 3.2, which covers the incremental learning methods studied in this research. Section 3.3 describes the datasets used, while Section 3.4 provides implementation details for the experiments. Finally, Section 3.5 discusses the evaluation metrics used to assess the experiments.

3.1 Neural caching

In this work, we specifically focus on classification tasks rather than free text generation to address practical problems like routing user requests and answering factual questions similar to Ramírez et al. (2023). This focus allows us to use established Active Learning methods without modifications and avoids the complexities associated with evaluating text generation, as highlighted by Celikyilmaz, Clark, and Gao (2021). By doing so, we ensure our approach is both practical and effective for real-world classification challenges. Our neural caching system aims to generate class labels for a sequence of inputs $(x_1, \ldots, x_n)$ from the input space X. The student model is updated using Incremental Learning (IL) methods after every f processed request, where f represents the frequency of model updates. The proposed system simulates a stream of requests, processing each in real-time through either the student model or the LLM, based on the established policy. During the learning phase, access to ground truth is not assumed to replicate a fully automated online scenario. This setup facilitates the exploration of the trade-offs between performance and efficiency. The entire incremental caching process is illustrated in Figure 1.1, and the algorithm is detailed in Algorithm 1.

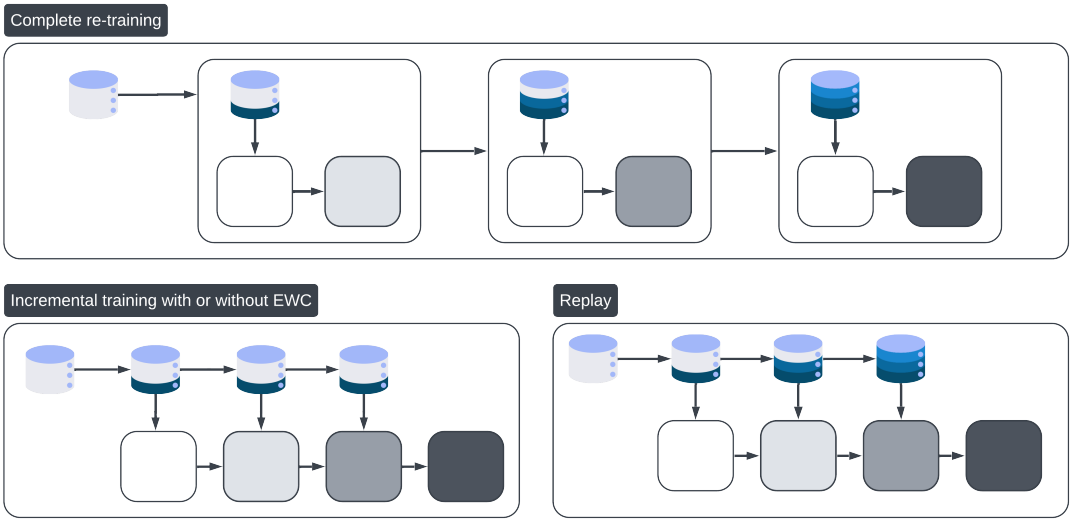

Figure 3.1: Visual comparison between complete retraining studied in Ramírez et al. (2023) and incremental learning methods studied in this research for incremental neural caching. The square boxes indicate the student model being trained on LLM annotated data with darker colour signifying better performance on validation data.

3.2 Incremental Learning Methods

We use incremental learning with Elastic Weight Consolidation and Replay for incremental neural caching.

Incremental learning (Finetuning): In this training method, the model is not reset to scratch during training. This means the model's parameters are continuously updated as new data arrives, rather than being reinitialized. As new data points are used to update the model, they are temporarily stored in a small cache. After the model is updated using those examples, they are quickly removed from the cache. This allows the model to learn from the most recent data without storing large amounts of historical data. When used with dynamic data streams, this method could be vulnerable to the *catastrophic forgetting* (McCloskey and Cohen, 1989) issue due to the lack of additional learning constraints. However, in a static data stream, the model may not suffer from the forgetting problem if the training set is sufficiently large, as the distribution of the training set would be closer to the true distribution. This method serves as our baseline for incremental approaches as it is the simplest form of incremental learning.

Elastic Weight Consolidation (EWC): EWC helps balance retaining previously learned knowledge while adapting to new data, ensuring consistent performance across all data, even when tasks remain unchanged. EWC was originally designed for scenarios

where tasks are changing, particularly in multi-task learning environments. However, we chose to employ EWC since it may help prevent the model from overfitting to recent data points, which is a risk in online learning scenarios. We assume a sequence of tasks exists, although, in our setup, we will use EWC to retain knowledge of the model from the previous training iteration. That is, we'll use EWC to protect knowledge from the previous version of the student model instead of a different task. At each retraining step, before updating the model, we compute the Fisher Information Matrix F for the current model parameters. The Fisher matrix represents the importance of each parameter to the model's current performance. The parameters of the new model are updated based on the matrix F.

We implemented a simplified version of EWC. In this approach, we don't maintain a history of past model parameters or keep track of multiple Fisher Information Matrices. Instead, we only use the most recent set of parameters and the corresponding Fisher Information Matrix. This streamlined version focuses on preserving knowledge from the immediately preceding training iteration, rather than accounting for multiple past states of the model. Since most of our experiments involved static data streams, the standard EWC, which stores the entire history of parameters and FIMs for all training iterations may be inefficient for our scenario.

Replay: In this training method, a replay buffer or cache is used to store a subset of previously seen examples to mitigate forgetting (Hayes et al., 2020). As new examples arrive, they are used to update the student model. Simultaneously, a batch of examples from the replay buffer is also used for training. This combination of new and replayed examples helps mitigate catastrophic forgetting.

The replay buffer is similar to the exemplar set in iCaRL (Rebuffi et al., 2017). iCaRL stores class prototypes for classification in incremental learning, while the replay method in this research doesn't organize examples by class and uses stored examples for retraining a language model. iCaRL is tailored for class-incremental learning, whereas the replay method is more versatile and applicable to various language tasks beyond class-based scenarios. Also, Masson D'Autume et al. (2019) uses sparse experience replay which is similar to our approach. However, it includes a local adaptation phase and uses a key network for retrieval, unlike our replay method, which lacks local adaptation and relies on random sampling. Additionally, the paper broadly targets various language tasks, whereas our work specifically focuses on caching LLM requests.

IL research typically focuses on class, task, or domain incremental learning (Delange et al., 2021). Since our study aimed to explore the impact of IL methods on static data

streams as well, many existing approaches were not suitable for our setup. We chose EWC as a representative of regularization-based IL methods since EWC is widely used in continual learning literature and has shown effectiveness in preserving knowledge across tasks. From the rehearsal-based IL methods, we chose to study Replay. We did not choose any architectural methods for IL since they are more suited for task-specific IL and the associated complexity in implementation and comparison with other methods. We leave the investigation of state-of-the-art IL methods in the context of neural caching for future research.

A visual illustration of the difference between this work and previous work by Ramírez et al. (2023) can be seen in Figure 3.1.

3.3 Datasets

We use the following datasets: ISEAR (Shao, Doucet, and Caruso, 2015), RT-Polarity (Pang and Lee, 2005), FEVER (Thorne et al., 2018), and Openbook (Mihaylov et al., 2018). These datasets are chosen since the LLM consistently achieved higher accuracy than the smaller model trained on 5000 gold labels, indicating that knowledge distillation (KD) would be useful as per Ramírez et al. (2023). Each dataset is split into training and test sets with an 80%-20% division, except for Openbook due to its smaller size. The class distribution is uniform across all datasets.

ISEAR (Shao, Doucet, and Caruso, 2015) comprises data collected over several years in the 1990s by a large group of psychologists from around the world. The dataset features reports on seven major emotions from nearly 3000 respondents across 37 countries on all 5 continents (classes: joy, fear, shame, sadness, guilt, disgust, anger; 7666 examples).

RT-Polarity (Pang and Lee, 2005) is a collection of movie review documents annotated for their overall sentiment polarity (classes: positive, negative; 10662 examples).

FEVER (Thorne et al., 2018) is a fact-checking dataset containing claims that can be verified using Wikipedia (classes: true, false; 6612 examples).

Openbook (Mihaylov et al., 2018) is a challenging question-answering dataset intended to evaluate the human understanding of a topic, similar to open-book exams. Each entry includes a multiple-choice question (classes: A, B, C, D) along with a fact that can aid in answering the question. The dataset consists of 5957 data points, with 5457 used for training and 500 reserved for testing.

3.4 Experiment Details

We conduct all our experiments using three random seeds, determining the order of examples and we report the average scores. For consistency, the frequency of updating the student model, designated by f, is set to 1000, and each query incurs a uniform cost, denoted by $c(x_i) = 1$. To mitigate the cold start problem, the initial model, S_0, is pre-trained using data points derived from the Large Language Model (LLM) with $N = 100$ for the ISEAR and RT-Polarity datasets, and $N = 1000$ for FEVER and Openbook, ensuring that S_0 performs above the baseline of random selection as per Ramírez et al. (2023).

$T5_{base}$ (Raffel et al., 2019) is used as the backbone for the student model. We freeze the model weights and add LoRA adapter layers for parameter-efficient fine-tuning without altering the underlying model weights (Hu et al., 2022). We fine-tune the student model by minimizing the cross-entropy loss between its predictions and the soft labels (log probabilities) provided by the teacher model (LLM) for each class. The budget range for a task (b_{task}) is be varied from 1000 to 3500 in increments of 500. The LLM annotated data accumulated in the cache is split into training and validation sets with a 90%-10% division before every training iteration. We use the LLM annotated dataset[1] released by Ramírez et al. (2023) as a starting point for the experiments. Refer to Appendix A for details regarding the experiment environment and hyperparameters.

3.4.1 Optimizer and learning rate scheduler

An optimizer is an algorithm that adjusts the parameters of a model to minimize the loss function and improve the model's accuracy during training. For all our experiments, we utilize the AdamW optimizer (Loshchilov and Hutter, 2019), which combines the benefits of Adam's (Kingma and Ba, 2017) adaptive learning rates with weight decay regularization. AdamW uses parameter-specific learning rate adaptation. AdamW's adaptive learning rates result in smaller updates for parameters that have consistently large gradients and larger updates for parameters with smaller or infrequent gradients. This behaviour is particularly beneficial for incremental learning scenarios. Parameters that are crucial for maintaining performance on previously seen data are likely to have had larger gradients in earlier training phases. As a result, these parameters will tend to have smaller learning rates in subsequent training phases, which helps to preserve their values and mitigate catastrophic forgetting to an extent.

[1] https://huggingface.co/datasets/guillemram97/cache_llm

The consistency in optimal hyperparameters across different learning setups (incremental, replay, and complete retraining) based on hyperparameter experiments in Appendix A suggests that AdamW's adaptive nature contributes to a more robust learning process. This stability is crucial for incremental learning, as it allows for consistent performance across different stages of the learning process without the need for frequent hyperparameter adjustments.

We use a learning rate scheduler along with the optimizer in our experiments. A learning rate scheduler adjusts the learning rate during training according to a predefined schedule to improve model convergence. The interaction between AdamW and a learning rate scheduler involves a layered adaptation process where AdamW adjusts parameter-specific learning rates based on gradient moments (Kingma and Ba, 2017), while the scheduler modifies the overall base learning rate. This creates a complementary dynamic where the scheduler manages global learning rate adjustments, and AdamW fine-tunes learning for individual parameters. In incremental learning, this combination can help balance retaining old knowledge with acquiring new information.

We compared a linear scheduler with a linear scheduler with hard resets. The linear scheduler gradually decreases the learning rate across iterations, while the hard reset variant resets the learning rate at each iteration. In our experiments, we found that both schedulers show similar final accuracy, with the linear scheduler having only a minor decrease in online accuracy. The absence of resets in the linear scheduler may contribute to a more stable learning process, leading us to choose the linear scheduler for all our experiments. Refer to Section A in the Appendix for the experiment results.

3.5 Evaluation

3.5.1 Performance

To report the performance of our model, we use average accuracy. We obtain average accuracy by dividing the corresponding Area Under the Curve (AUC) by the budget range. This gives a normalized view of the model's performance across varying budget constraints. We consider online accuracy in addition to final accuracy because online accuracy represents the average service quality provided to the user.

3.5.2 Efficiency

Efficiency metrics are crucial for evaluating the feasibility and scalability of the model, especially in resource-constrained environments. Similar to the performance metric, we obtain the average value of the efficiency metrics by dividing the Area Under the Curve (AUC) by the budget range. We use the following metrics to evaluate the efficiency of our models.

Training time:.Training time is the total duration required to train a model from start to finish, reflecting the computational cost and speed of the training process. In our neural caching setup, training time is the total amount of time spent for all the training iterations. It is intuitive and directly related to practical usability. Making fair comparisons between models using training time could be challenging in certain scenarios since the metric is sensitive to external factors such as hardware specifications, concurrent processes, and core utilization. However, these limitations have a limited effect on our setup since we use the same model and hardware for all the experiments.

Floating point operations (FLOPs): The FLOPs (Floating Point Operations) quantifies the total number of floating-point operations required to train or infer with a model, serving as an indicator of the computational workload and resource demands of the model. FLOPs offer a more standardized and hardware-agnostic estimate of computational work, correlating strongly with energy consumption and runtime while allowing for equitable comparisons across different research environments. This metric considers the granular work done at each time step, surpassing the limitations of asymptotic runtime analysis. However, it is important to note that FLOPs primarily focus on computational intensity, potentially overlooking other crucial factors such as memory usage. Schwartz et al. (2020) argue that by emphasizing FLOPs alongside accuracy in model evaluation, the field can foster the development of ”Green AI” - models that achieve high performance while minimizing computational resources and environmental impact.

We did not choose other efficiency metrics for our comparison due to the following reasons as per Schwartz et al. (2020). Carbon emissions are difficult to measure accurately and vary based on local electricity infrastructure, making them unreliable for comparing models. Electricity usage is correlated with emissions and can be estimated using GPU data, but it is hardware-dependent and not fully comparable across models. The number of parameters is hardware-independent and correlates with memory usage, but models with similar parameter counts can perform different amounts of work. Also,

the model used in our study has the same number of parameters across all experiments.

In our experiments, we found that the training time results closely resemble the FLOPs results, a pattern observed consistently across all our experiments. Therefore, we may use these metrics interchangeably to quantify efficiency for our experiments.

Algorithm 1: Pseudo-code for the neural caching algorithm with datasets (tasks) *Tasks*, the budget for the current task b_{task}, model update frequency f, cost per query c, and an initial student S_0. Call_LLM function calls the policy algorithm to decide if the LLM has to be invoked for the input query. The Trim function trims the cache (or replay buffer) if the buffer percent is less than 100% based on the buffer eviction policy.

$D_{online} \leftarrow \emptyset$
for *task in Tasks* **do**
  for x_i *in* X_{online} **do**
    if $i \ mod \ f == 0$ **then**
      if *Incremental* **then**
        $S_{i/f} \leftarrow \text{Update}(cache)$
        $cache \leftarrow \emptyset$
      end
      if *Replay* **then**
        $S_{i/f} \leftarrow \text{Update}(cache)$
        $Trim(cache)$
      end
      if *Retrain* **then**
        $S_{i/f} \leftarrow \text{Retrain}(cache)$
      end
    end
    $\hat{y}_i \leftarrow S_{i/f}(x_i)$
    if *Call_LLM*$(b, x_i, \hat{y}_i)$ *and* $b_{task} \geq c(x_i)$ **then**
      $\hat{y}_i \leftarrow \text{LLM}(x_i)$
      $b \leftarrow b - c(x_i)$
      $cache \leftarrow cache \cup \{\langle x_i, \hat{y}_i \rangle\}$
    end
    $D_{online} \leftarrow D_{online} \cup \{\langle x_i, \hat{y}_i \rangle\}$
  end
end
$D_{test} \leftarrow \{\langle x_j, S_{i/f}(x_j) \rangle | x_j \in X_{test}\}$
$Acc_{online} \leftarrow \text{Evaluate}(D_{online})$
$Acc_{final} \leftarrow \text{Evaluate}(D_{test})$

Chapter 4

Experiments

Based on the experiment design outlined in the previous chapter, this chapter presents the experiments and results. Section 4.1 covers experiments with static data streams, including a preliminary study on buffer eviction policy for the replay buffer, followed by an examination of the FR policy in Subsection 4.1.2 and AL-based policies in Subsection 4.1.3. Subsection 4.1.4 explores the impact of higher retraining frequency in static data streams. Finally, Section 4.2 discusses experiments with dynamic data streams.

4.1 Incremental neural caching with static data streams

4.1.1 Preliminary experiment: Buffer eviction policy study

Since the size of the replay buffer can be controlled, we performed experiments to study common eviction policies. We chose to compare the policies outlined below:

- **First-In-First-Out (FIFO):** Remove the oldest examples.
- **Random:** Remove examples from the buffer randomly.
- **Margin Sampling (MS):** Remove examples deemed less important based on the margin for maintaining performance.

The size of the replay buffer is set to be 25%, 50% and 75% of the budget. The experiments are conducted in the ISEAR dataset with retraining frequency $f = 1000$, FR as instance selection criteria and Replay as the incremental learning method. The results can be seen in Table 4.1 and Table 4.2.

Buffer percent	FIFO policy	MS policy	Random policy
25%	0.631 ± 0.003	0.628 ± 0.005	**0.637 ± 0.001**
50%	0.634 ± 0.005	0.634 ± 0.004	**0.641 ± 0.002**
75%	0.641 ± 0.002	0.641 ± 0.002	**0.642 ± 0.003**

Table 4.1: Online Accuracy (AUC) for eviction policy study (ISEAR dataset)

Buffer percent	FIFO policy	MS policy	Random policy
25%	0.588 ± 0.003	0.583 ± 0.006	**0.603 ± 0.003**
50%	0.593 ± 0.008	0.593 ± 0.006	**0.607 ± 0.004**
75%	0.606 ± 0.005	0.603 ± 0.006	**0.608 ± 0.001**

Table 4.2: Final Accuracy (AUC) for eviction policy study (ISEAR dataset)

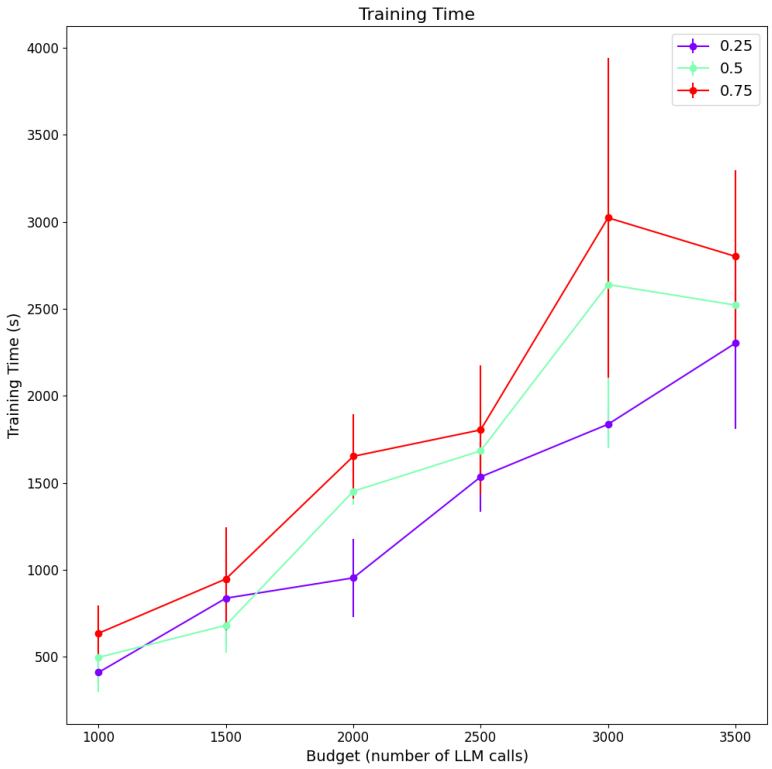

Figure 4.1: Training Time (s) for random eviction policy across budgets for various buffer percents

We can observe that for both online and final accuracy, the Random policy outperforms FIFO and MS policies. This validates the efficacy of the random eviction policy, which worked well in previous research (Chaudhry et al., 2018; Hayes et al., 2020;

Wu et al., 2019) as described in the background chapter. The choice of eviction policy appears to have a more significant impact on performance at smaller buffer sizes, with differences becoming less pronounced at larger sizes. The improvement in accuracy from 50% to 75% is smaller than from 25% to 50%, suggesting diminishing returns as buffer size increases. From the above discussion and the training time plot from Figure 4.1, we can confirm that the buffer percent of 50% provides a good trade-off between performance and efficiency. Also, we can notice that the deviation of the accuracy values for the random policy is smaller confirming its robustness. Hence, we use a buffer percent of 50% with a random buffer eviction policy in addition to a full (100%) buffer for replay experiments.

4.1.2 Incremental neural caching with Front-loading (FR) policy

We first study the performance and efficiency of IL methods using a simple Front-loading (FR) policy, since the training pattern of the FR is straightforward. The experiments are conducted in all datasets introduced in Section 3.3 with retraining frequency $f = 1000$. The training methods of the experiments include all the IL methods introduced in Section 3.2 along with Replay (50%) introduced in Section 4.1.1 and complete retraining. The performance and efficiency metrics outlined in Section 3.5 are calculated for these experiments. The online accuracy, final accuracy, FLOPs and training time are shown in Table 4.3, Table 4.4, Table 4.5 and Table 4.6. Figure 4.2 represents the performance and efficiency curves with respect to budgets averaged across four datasets for the front-loading strategy. Figure 4.3, and Figure 4.4 represent the online accuracy and total training time respectively across four datasets.

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	**0.646** ± **0.004**	0.725 ± 0.004	**0.880** ± **0.001**	**0.736** ± **0.002**	**0.747** ± **0.003**
Incremental	0.632 ± 0.003	0.718 ± 0.003	0.879 ± 0.001	0.731 ± 0.001	0.740 ± 0.002
EWC	0.632 ± 0.001	0.718 ± 0.004	**0.880** ± **0.001**	0.733 ± 0.001	0.741 ± 0.002
Replay (100%)	0.644 ± 0.004	0.717 ± 0.005	0.879 ± 0.002	0.732 ± 0.004	0.743 ± 0.004
Replay (50%)	0.641 ± 0.002	**0.729** ± **0.002**	0.879 ± 0.001	0.733 ± 0.002	0.745 ± 0.002

Table 4.3: Online Accuracy (AUC) for FR Strategy

From the results, we can observe that plain incremental learning has the lowest performance confirming that neural caching with incremental learning suffers from

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	**0.612** ± **0.003**	0.640 ± 0.004	**0.884** ± **0.002**	0.683 ± 0.006	**0.705** ± **0.004**
Incremental	0.593 ± 0.006	0.630 ± 0.002	0.879 ± 0.002	0.679 ± 0.005	0.695 ± 0.004
EWC	0.594 ± 0.006	0.634 ± 0.003	0.882 ± 0.002	0.680 ± 0.005	0.698 ± 0.004
Replay (100%)	0.609 ± 0.002	0.624 ± 0.006	0.882 ± 0.003	**0.684** ± **0.011**	0.700 ± 0.005
Replay (50%)	0.607 ± 0.004	**0.646** ± **0.005**	0.881 ± 0.001	0.683 ± 0.002	0.704 ± 0.003

Table 4.4: Final Accuracy (AUC) for FR Strategy

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	6.69 ± 0.87	5.95 ± 0.67	4.29 ± 0.61	4.90 ± 0.38	5.46 ± 0.63
Incremental	1.99 ± 0.18	3.32 ± 0.17	1.86 ± 0.15	1.72 ± 0.16	2.22 ± 0.17
EWC	2.07 ± 0.26	3.82 ± 0.36	1.89 ± 0.14	1.72 ± 0.12	2.38 ± 0.22
Replay (100%)	4.91 ± 0.41	4.88 ± 0.34	4.07 ± 0.28	3.15 ± 0.18	4.25 ± 0.30
Replay (50%)	4.29 ± 0.50	5.54 ± 0.70	3.43 ± 0.07	2.71 ± 0.21	4.00 ± 0.37

Table 4.5: FLOPS (E+15) (AUC) for FR Strategy

catastrophic forgetting confirming the previous literature discussed in the background chapter. EWC performs better compared to plain incremental learning. However, this comes at the cost of tuning the extra hyperparameter λ and additional computation involved for calculating the new loss term. Also, this method still has low performance compared to complete retraining. Complete retraining generally achieves the highest online and final accuracy across datasets. Replay methods, especially Replay (50%), often perform nearly as well, occasionally surpassing complete retraining at certain budget points, based on Figure 4.2. Replay (50%) has marginally better online and final accuracy compared to Replay (100%) across datasets.

From Figure 4.2, we can see that the training time results mirror the FLOPs results. We find this trend in all our experiments. Figure 4.4 illustrates that complete retraining generally demands the most training time across different datasets and budgets, although Replay exceeds complete retraining for lower budgets in the Openbook and RT-Polarity datasets. Incremental and EWC methods are significantly more efficient, using fewer FLOPs and less training time compared to other methods. They are also memory efficient compared to other methods since they discard old examples after training. Despite significant efficiency gains, incremental methods maintain performance close

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	2427 ± 245	1808 ± 201	1702 ± 229	2509 ± 168	2112 ± 211
Incremental	745 ± 69	1012 ± 53	777 ± 53	936 ± 91	868 ± 66
EWC	789 ± 88	1168 ± 104	786 ± 62	979 ± 72	931 ± 82
Replay (100%)	1743 ± 134	1469 ± 68	1657 ± 146	1646 ± 109	1629 ± 114
Replay (50%)	1593 ± 163	1687 ± 215	1310 ± 23	1440 ± 145	1508 ± 137

Table 4.6: Training Time (s) (AUC) for FR Strategy

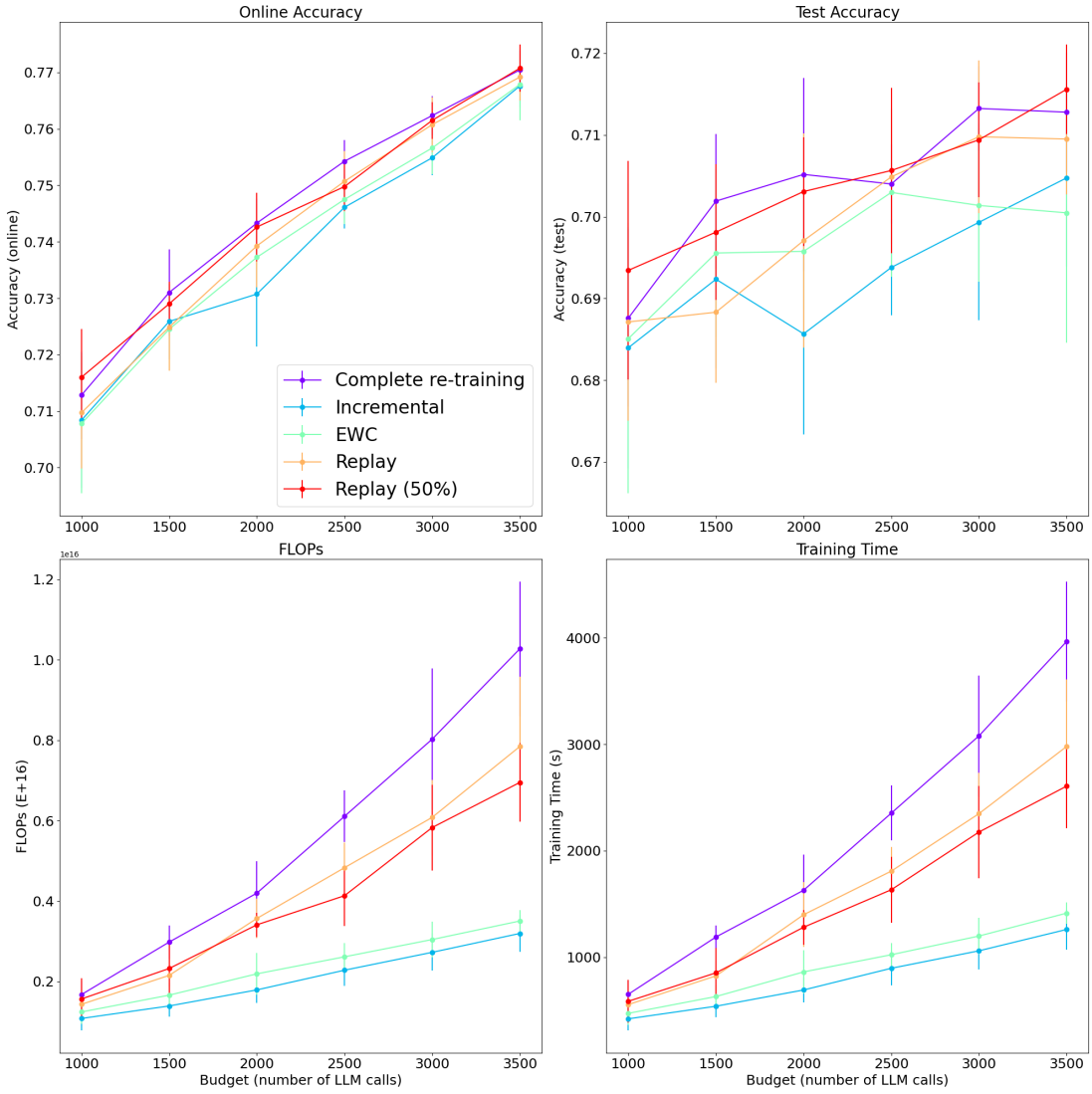

Figure 4.2: Performance and efficiency curves with respect to budgets for front-loading (FR) strategy. Error bars represent variance. The results have been averaged across the four datasets.

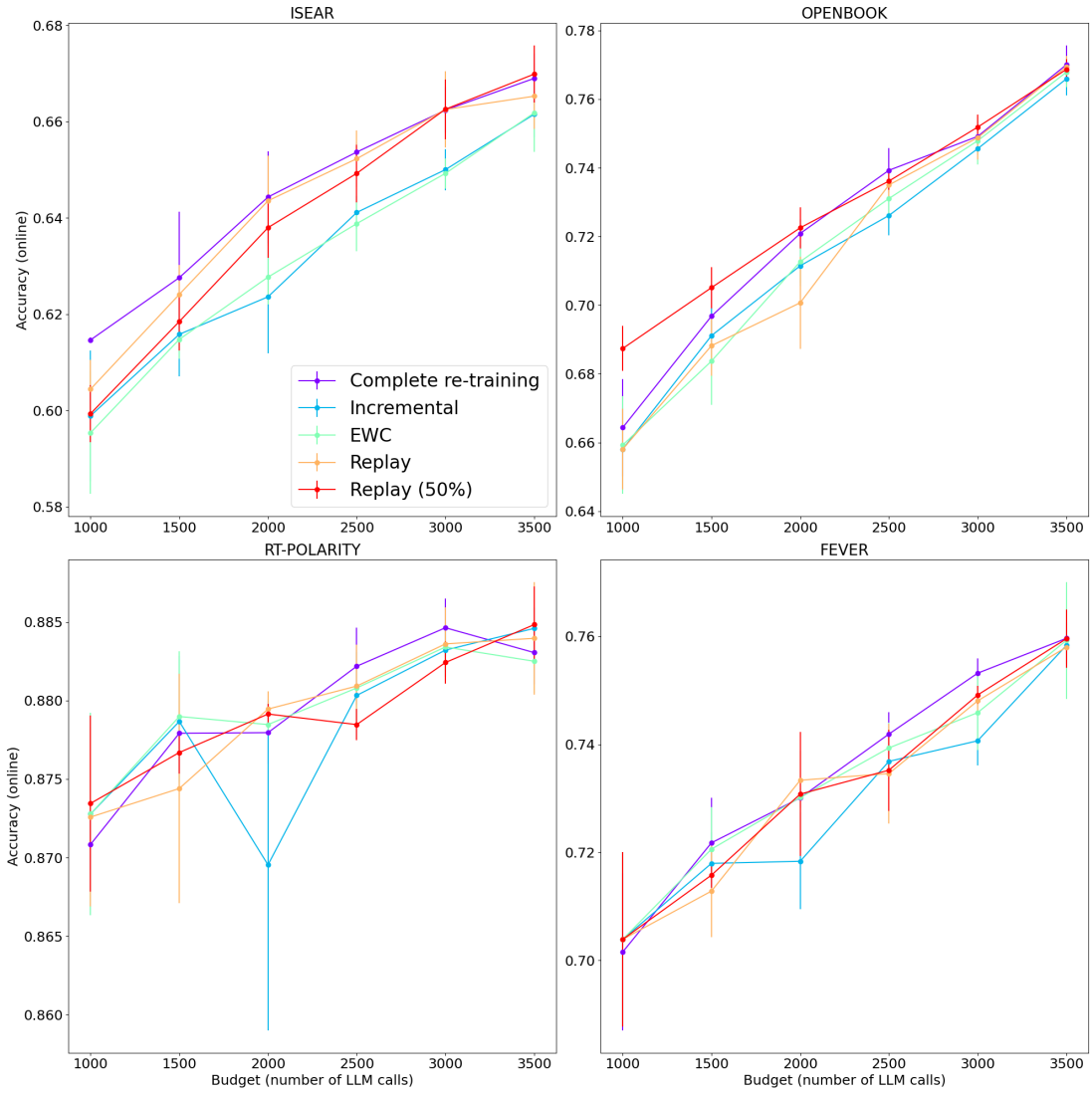

Figure 4.3: Online Accuracy curve with respect to budgets for Front-loading (FR) strategy across four datasets. Error bars represent variance.

to the best-performing approach (within 1-2% in most cases). Therefore, they are well-suited for stationary data streams where computational and memory efficiency are essential. The computational efficiency of Replay methods, both 100% and 50%, falls between complete retraining and incremental methods.

Across all methods, accuracy tends to increase with higher budgets, although the gains diminish over time, whereas efficiency tends to scale more linearly with budget, especially for complete retraining. This suggests that with larger budgets or datasets, the efficiency of complete retraining would decrease.

We can infer that there is a clear trade-off between performance and efficiency, with Incremental methods offering significant efficiency gains at the cost of some

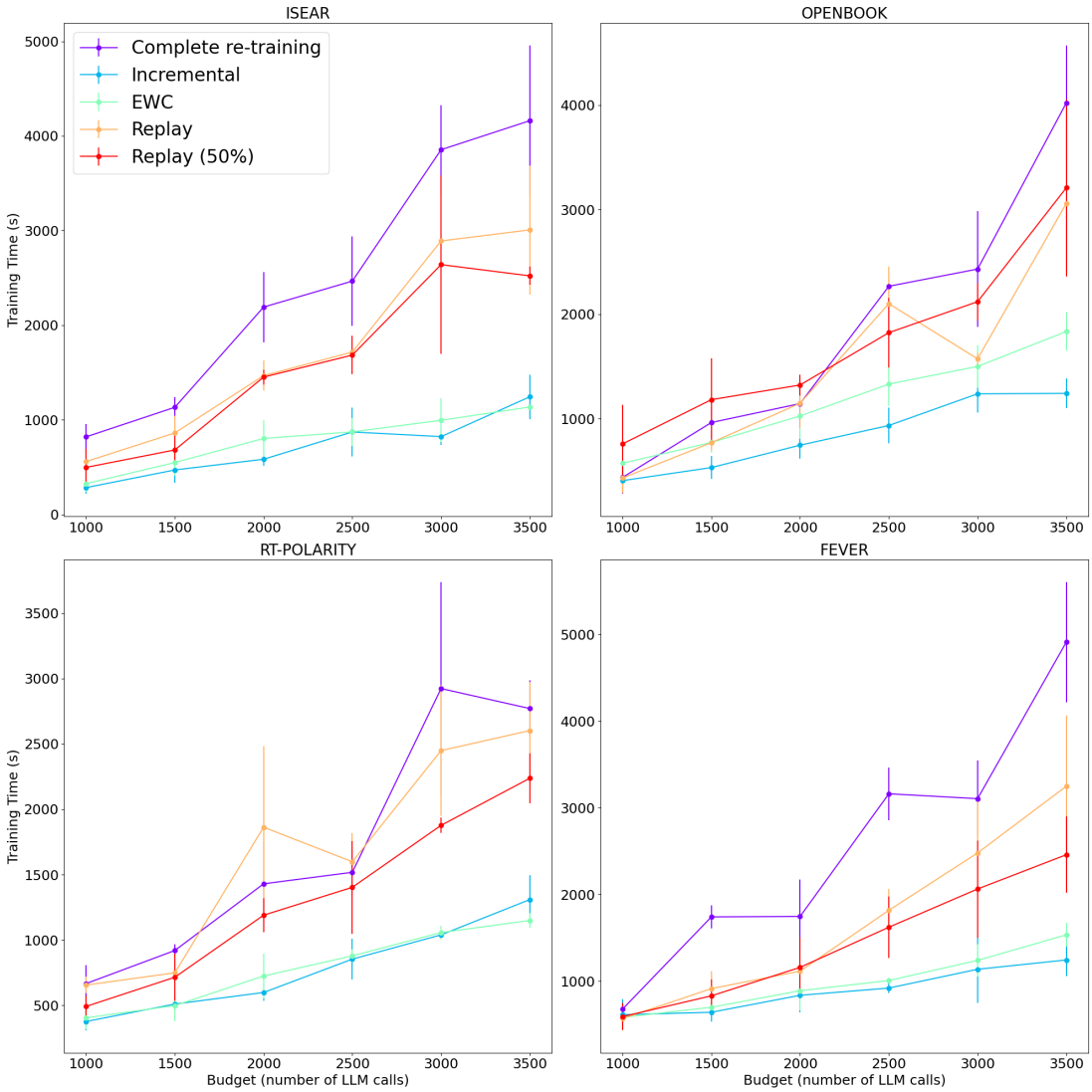

Figure 4.4: Training time curve with respect to budgets for Front-loading (FR) strategy across four datasets. Error bars represent variance.

performance. Figure 4.5 gives a comprehensive view of this trade-off. We can see that Replay (50%) offers a good balance between performance and computational efficiency. For scenarios where performance is paramount, complete retraining is a viable option, while incremental methods and Replay (50%) offer a good balance between efficiency and performance.

4.1.3 Incremental neural caching with AL policies

We repeat the incremental neural caching experiment explained in the previous section for all AL policies outlined in Section 2.1.1. In this section, we focus on the results of

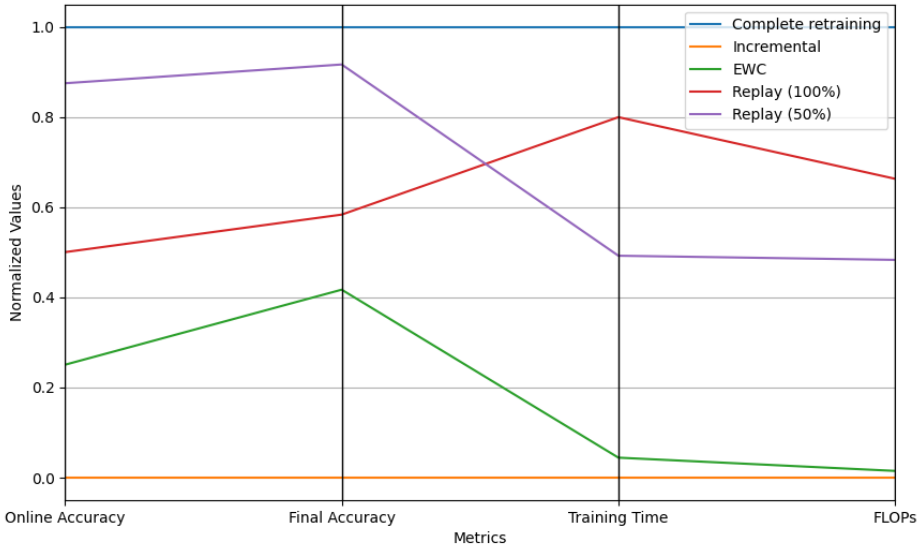

Figure 4.5: Comparison of training methods using min-max normalized performance and efficiency metrics for FR strategy averaged across four datasets

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	**0.665** ± **0.002**	0.727 ± 0.008	0.890 ± 0.001	0.748 ± 0.003	0.758 ± 0.004
Incremental	0.656 ± 0.001	0.734 ± 0.006	**0.893** ± **0.001**	0.748 ± 0.005	0.758 ± 0.004
EWC	0.656 ± 0.001	0.738 ± 0.006	**0.893** ± **0.001**	0.748 ± 0.006	0.759 ± 0.004
Replay (100%)	0.663 ± 0.003	0.730 ± 0.007	0.892 ± 0.000	0.746 ± 0.002	0.758 ± 0.004
Replay (50%)	0.662 ± 0.002	**0.758** ± **0.006**	0.892 ± 0.001	**0.748** ± **0.002**	**0.765** ± **0.003**

Table 4.7: Online Accuracy (AUC) for MS Strategy

the Margin Sampling (MS) strategy, as we observed consistent trends across all active learning (AL) policies. Detailed experimental results for all strategies are provided in Appendix A. The AUC across budgets of online accuracy, final accuracy, FLOPs and training time for MS policy are shown in Tables 4.7, 4.8, 4.9 and 4.10 respectively. Figure 4.6 shows the performance and efficiency curves with respect to budget with values averaged across the four datasets for MS strategy.

From the results, we observe results very similar to the results for FR strategy. However, there are a few key differences as discussed below. The online accuracy of all the training methods improved due to the use of AL-based policies. There is an increase in the number of training iterations with a lesser number of examples when AL-based policies are compared to the FR policy. This leads to an increase in FLOPs (Table 4.9) and training time (Table 4.10. Consequently, EWC does not improve over

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	**0.619** ± **0.006**	0.637 ± 0.003	**0.887** ± **0.002**	**0.683** ± **0.002**	0.706 ± 0.004
Incremental	0.600 ± 0.002	0.628 ± 0.007	0.882 ± 0.002	0.682 ± 0.006	0.698 ± 0.005
EWC	0.600 ± 0.001	0.633 ± 0.009	0.882 ± 0.003	0.674 ± 0.007	0.697 ± 0.006
Replay (100%)	0.615 ± 0.002	0.636 ± 0.007	0.884 ± 0.002	0.681 ± 0.005	0.704 ± 0.005
Replay (50%)	0.610 ± 0.003	**0.652** ± **0.002**	**0.887** ± **0.002**	0.683 ± 0.007	**0.708** ± **0.004**

Table 4.8: Final Accuracy (AUC) for MS Strategy

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	9.14 ± 0.71	8.90 ± 1.00	10.35 ± 1.20	8.75 ± 0.70	9.28 ± 0.93
Incremental	2.09 ± 0.16	3.07 ± 0.13	1.65 ± 0.13	1.88 ± 0.10	2.17 ± 0.13
EWC	2.09 ± 0.21	3.55 ± 0.20	1.65 ± 0.12	1.86 ± 0.10	2.29 ± 0.16
Replay (100%)	6.27 ± 0.17	6.61 ± 0.32	8.48 ± 0.38	5.21 ± 0.06	6.64 ± 0.26
Replay (50%)	5.53 ± 0.58	7.26 ± 0.23	6.87 ± 0.51	3.77 ± 0.23	5.86 ± 0.42

Table 4.9: FLOPS (E+15) (AUC) for MS Strategy

to plain incremental learning. Replay (50%) generally achieves the highest online and final accuracy across datasets. Replay (50%) has marginally better online and final accuracy compared to Replay (100%) across datasets.

Figure 4.6 illustrates that complete retraining generally demands the most training time across different datasets and budgets. Incremental learning methods (with or without EWC) drastically reduce FLOPs and training time compared to complete retraining. Despite large compute and memory gains, incremental methods maintain performance relatively close to the best-performing method (within 1-2% in most cases). Hence, incremental methods are suitable for stationary data streams where computational and memory efficiency are critical. The tradeoff between model performance and efficiency is observed in the case of AL policies as well. Consequently, we find that there is an increase in the computational efficiency improvement provided by Replay methods over complete retraining when compared with the FR strategy. The performance gap between incremental methods and complete retraining is smaller than the computational efficiency gap, indicating diminishing returns for additional computation. Replay methods, especially Replay (50%) offer better performance and computational efficiency compared to complete retraining. Our experiments confirm the results of Ramírez et al.

Training method	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	3447 ± 356	2682 ± 282	4096 ± 439	4419 ± 371	3661 ± 366
Incremental	883 ± 139	969 ± 46	718 ± 62	1011 ± 46	895 ± 83
EWC	952 ± 239	1137 ± 79	752 ± 70	1031 ± 49	968 ± 133
Replay (100%)	2337 ± 47	2008 ± 94	3358 ± 167	2704 ± 23	2602 ± 99
Replay (50%)	2054 ± 222	2191 ± 91	2662 ± 201	1979 ± 96	2222 ± 164

Table 4.10: Training Time (s) (AUC) for MS Strategy

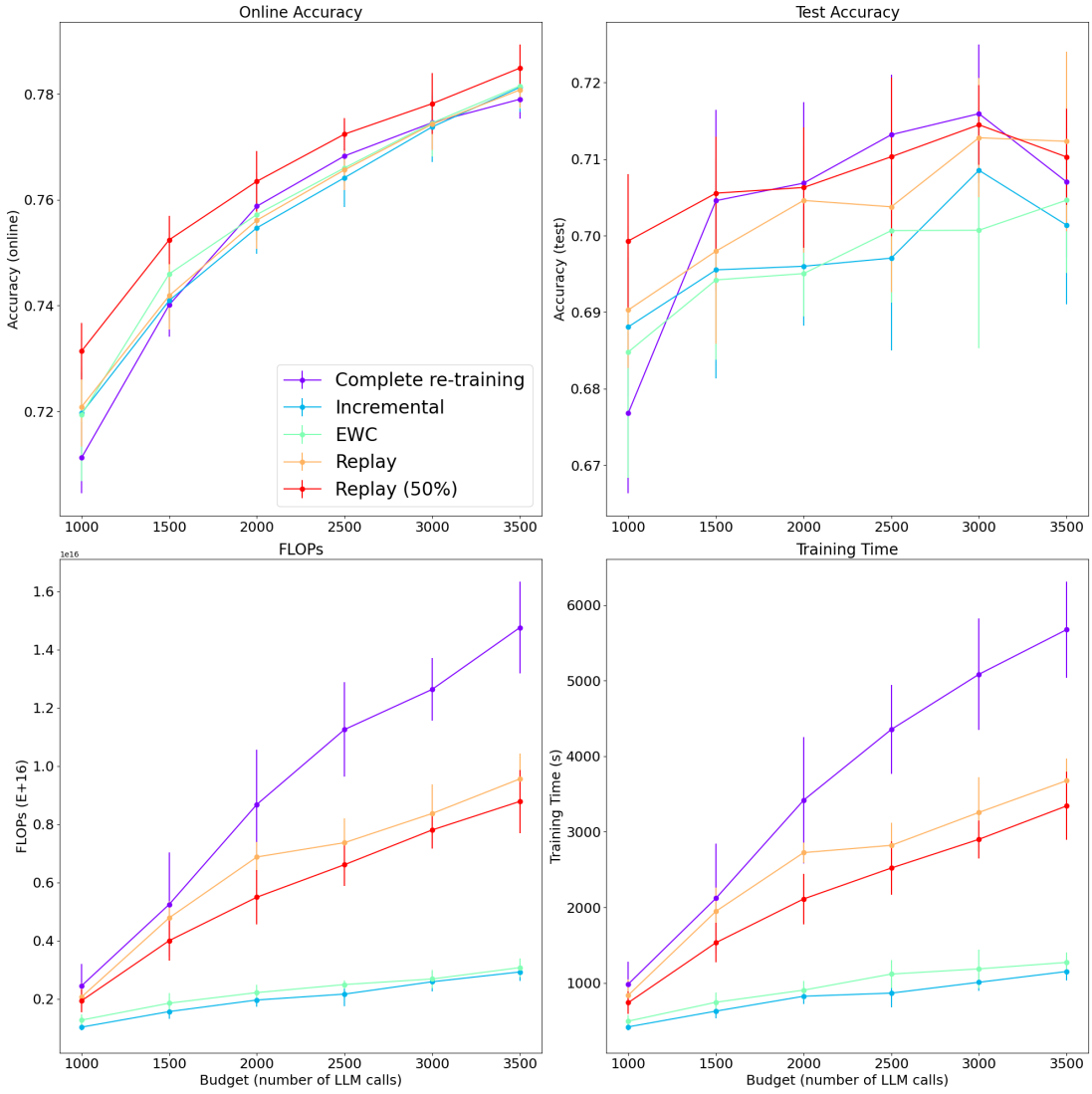

Figure 4.6: Performance and efficiency curves with respect to budgets for Margin Sampling (MS) strategy. Error bars represent variance. The results have been averaged across the four datasets.

(2023) that AL-based policies improve the performance over FR strategy. Additionally, our experiments confirm the efficacy of Active Learning (AL) policies in an incremental learning setup.

4.1.4 Higher retraining frequency f:

To assess the effect of an increased retraining frequency, we conducted the neural caching experiments again with a higher frequency of $f = 100$. The corresponding results are presented in Table 4.11. The online accuracy for MS remains relatively unchanged with the increased retraining frequency. However, a decline in online accuracy is observed for FR in both Incremental and EWC methods. EWC provides marginal performance improvement over plain incremental learning. The performance decrease in Incremental and EWC methods is similar to $f = 1000$ experiments. The memory and computational efficiency of Incremental and EWC methods increase with an increase in retraining frequency. The increase in training iterations does not increase the overall training time when compared to the results for retraining frequency of $f = 1000$.

Complete retraining and Replay methods consistently demonstrate the highest online and final accuracy. It is important to note that the Replay method incurs nearly double the FLOPs and training time compared to complete re-training across both MS and FR strategies. Thus, we can see that at higher retraining frequencies, complete retraining provides a more favourable trade-off between performance and computational efficiency relative to the incremental learning methods explored in this study. Incremental and EWC methods are suitable for memory-constrained systems.

When comparing the results with the results for retraining frequency of $f = 1000$, we can observe that increasing the frequency in a static data stream does not enhance performance. Consequently, it disrupts the favourable balance between performance and computational efficiency for complete retraining and Replay methods. However, it increases the efficiency of Incremental and EWC methods.

4.2 Incremental neural caching with dynamic data streams

We re-sampled the datasets in a class incremental way with samples from a single class grouped to simulate a dynamic data stream. We use the LLM annotated label for the re-sampling based on the order described in Section A. The experimental

Training method	Strategy	Online Acc.	Final Acc.	FLOPs (E+15)	Training Time (s)
Complete retraining	MS	**0.674** ± 0.001	0.621 ± 0.006	24.08 ± 0.83	9056 ± 226
	FR	0.644 ± 0.002	0.600 ± 0.004	17.62 ± 0.67	6624 ± 183
Incremental	MS	0.654 ± 0.005	0.592 ± 0.004	1.55 ± 0.04	854 ± 17
	FR	0.617 ± 0.004	0.565 ± 0.002	1.41 ± 0.03	690 ± 15
EWC	MS	0.657 ± 0.002	0.595 ± 0.004	1.55 ± 0.03	906 ± 22
	FR	0.617 ± 0.003	0.567 ± 0.002	1.54 ± 0.02	750 ± 25
Replay (100%)	MS	0.670 ± 0.002	0.620 ± 0.004	47.24 ± 1.0	18065 ± 251
	FR	0.657 ± 0.003	**0.629** ± 0.003	36.99 ± 0.93	13473 ± 197

Table 4.11: Performance and Efficiency metrics for neural caching with retraining frequency $f = 100$ (ISEAR dataset)

procedure outlined previously is repeated across different training methods and two strategies: Front-loading and Margin Sampling. The results can be seen in tables 4.12, 4.13, 4.14 and 4.15. From Table 4.12, we can observe that Margin Sampling consistently outperforms Front-loading across all training types in terms of online accuracy. Incremental and EWC methods achieve the best online performance across datasets with MS policy and perform well even with FR strategy.

For final accuracy from Table 4.13, complete retraining and Replay methods continue to perform well, but Incremental and EWC methods exhibit a significant drop compared to their online performance. This confirms that Incremental and EWC methods adapt strongly to recent data and forget the past examples. We can also note that our simplified version of EWC is unable to help the model retain its past knowledge. Also, MS strategy generally outperforms FR, except in Incremental and EWC methods where FR performs better. This shows that the MS policy helps the model adapt to new data and improves the online accuracy at the cost of final accuracy. Thus, Incremental and EWC methods would have poor accuracy when the data distribution changes to a distribution in the past.

The standard use case for EWC is in task-incremental learning, where a model learns a sequence of distinct tasks and is evaluated on all tasks, including earlier ones. While our simplified EWC implementation showed limited benefits in the dynamic stream scenario, it's important to note that EWC can be particularly effective in certain types of changing distributions, especially when the system encounters previously seen or similar distributions. While our simplified EWC was marginally beneficial for the

static stream scenario, the more challenging nature of class incremental learning may require a more sophisticated approach. A full implementation of EWC, maintaining multiple model parameter checkpoints and FIMs for each class and using a loss function that considers all previously learned classes, may be more effective in this dynamic scenario.

In terms of computational efficiency, complete retraining remains the most computationally expensive, particularly with MS, while Incremental and EWC methods are the most efficient, with Replay methods offering a balance between computational efficiency and performance. Training time trends mirror the FLOPS results. The impact of MS versus FR strategies is more pronounced in dynamic streams, and the efficiency trends remain consistent. MS is generally more effective but also more computationally demanding than FR, with strategy choice having a significant impact in dynamic scenarios. We can see that complete retraining performs well but doesn't dominate as much as in static streams, Incremental and EWC methods struggle with final accuracy, and Replay methods balance accuracy and computational efficiency. The dynamic stream highlights a stronger trade-off between online and final accuracy, especially for incremental methods, and the computational efficiency versus accuracy trade-off persists, with Replay methods offering a solid compromise. We can observe that replay methods provide better performance and computational efficiency when using dynamic data streams. This confirms the effectiveness of Replay in both static and dynamic data streams. We also find the performance gap between AL-based policies like MS and static policies like FR increases in dynamic data streams compared to static ones, supporting findings from previous research (Bifet and Gavaldà, 2007) discussed in the background section.

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	0.462 ± 0.001	0.543 ± 0.007	0.495 ± 0.000	0.642 ± 0.006	0.535 ± 0.003
	MS	0.531 ± 0.006	0.526 ± 0.009	0.794 ± 0.005	0.601 ± 0.003	0.613 ± 0.006
Incremental	FR	0.487 ± 0.004	0.534 ± 0.007	0.497 ± 0.000	0.712 ± 0.005	0.558 ± 0.004
	MS	0.552 ± 0.004	0.688 ± 0.007	0.844 ± 0.003	0.638 ± 0.005	**0.681 ± 0.005**
EWC	FR	0.488 ± 0.006	0.533 ± 0.007	0.497 ± 0.000	0.706 ± 0.005	0.556 ± 0.004
	MS	0.552 ± 0.005	0.688 ± 0.007	0.844 ± 0.003	0.638 ± 0.005	**0.681 ± 0.005**
Replay (100%)	FR	0.492 ± 0.008	0.550 ± 0.007	0.496 ± 0.001	0.645 ± 0.005	0.546 ± 0.005
	MS	0.544 ± 0.002	0.677 ± 0.016	0.742 ± 0.016	0.613 ± 0.002	0.644 ± 0.009
Replay (50%)	FR	0.499 ± 0.012	0.558 ± 0.011	0.497 ± 0.001	0.663 ± 0.007	0.554 ± 0.008
	MS	0.548 ± 0.003	0.684 ± 0.015	0.739 ± 0.013	0.618 ± 0.007	0.647 ± 0.010

Table 4.12: Online Accuracy (AUC)

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	0.443 ± 0.005	0.543 ± 0.006	0.502 ± 0.000	0.641 ± 0.002	0.533 ± 0.003
	MS	0.564 ± 0.013	0.574 ± 0.021	0.872 ± 0.009	0.682 ± 0.007	0.673 ± 0.014
Incremental	FR	0.414 ± 0.008	0.344 ± 0.008	0.503 ± 0.000	0.562 ± 0.006	0.457 ± 0.005
	MS	0.363 ± 0.016	0.240 ± 0.001	0.591 ± 0.017	0.500 ± 0.007	0.424 ± 0.011
EWC	FR	0.415 ± 0.011	0.339 ± 0.007	0.503 ± 0.000	0.564 ± 0.005	0.455 ± 0.006
	MS	0.355 ± 0.013	0.245 ± 0.001	0.591 ± 0.017	0.500 ± 0.007	0.423 ± 0.010
Replay (100%)	FR	0.469 ± 0.005	0.558 ± 0.006	0.503 ± 0.001	0.645 ± 0.011	0.544 ± 0.006
	MS	0.557 ± 0.007	0.644 ± 0.006	0.862 ± 0.002	0.661 ± 0.011	0.681 ± 0.007
Replay (50%)	FR	0.474 ± 0.011	0.552 ± 0.004	0.504 ± 0.002	0.644 ± 0.002	0.544 ± 0.005
	MS	0.556 ± 0.005	0.64 ± 0.011	0.865 ± 0.004	0.669 ± 0.007	**0.682 ± 0.007**

Table 4.13: Final Accuracy (AUC)

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	6.26 ± 0.36	9.14 ± 1.19	3.03 ± 0.01	4.04 ± 0.7	5.62 ± 0.56
	MS	9.40 ± 0.56	12.18 ± 0.45	7.02 ± 0.55	7.21 ± 0.7	8.45 ± 0.57
Incremental	FR	1.9 ± 0.1	3.82 ± 0.38	1.26 ± 0.01	1.50 ± 0.1	2.12 ± 0.15
	MS	1.97 ± 0.05	3.22 ± 0.28	1.26 ± 0.04	1.24 ± 0.1	1.93 ± 0.12
EWC	FR	1.9 ± 0.12	3.99 ± 0.35	1.26 ± 0.01	1.51 ± 0.1	2.16 ± 0.15
	MS	1.96 ± 0.03	3.20 ± 0.28	1.26 ± 0.04	1.24 ± 0.1	1.92 ± 0.11
Replay (100%)	FR	5.65 ± 0.46	7.58 ± 0.64	3.03 ± 0.01	3.50 ± 0.06	4.44 ± 0.29
	MS	7.16 ± 0.25	10.08 ± 1.03	6.61 ± 0.71	5.24 ± 0.23	7.27 ± 0.56
Replay (50%)	FR	4.59 ± 0.52	5.90 ± 0.47	2.49 ± 0.00	2.24 ± 0.23	3.81 ± 0.31
	MS	6.11 ± 0.48	7.60 ± 0.6	5.28 ± 0.2	3.74 ± 0.23	5.68 ± 0.38

Table 4.14: FLOPS (E+15) (AUC)

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	2210 ± 183	2689 ± 333	1190 ± 7	2131 ± 371	2055 ± 223
	MS	3328 ± 176	3817 ± 246	2836 ± 225	3719 ± 371	3425 ± 264
Incremental	FR	691 ± 32	1239 ± 155	519 ± 1	828 ± 46	797 ± 58
	MS	747 ± 21	1021 ± 84	525 ± 20	709 ± 96	742 ± 55
EWC	FR	719 ± 39	1296 ± 173	521 ± 9	870 ± 49	852 ± 68
	MS	768 ± 33	1032 ± 110	574 ± 13	753 ± 49	782 ± 51
Replay (100%)	FR	1928 ± 137	2250 ± 168	1173 ± 24	1862 ± 23	1803 ± 88
	MS	2506 ± 77	3049 ± 305	2629 ± 278	2421 ± 23	2651 ± 171
Replay (50%)	FR	1623 ± 218	1809 ± 139	972 ± 6	1259 ± 96	1416 ± 116
	MS	2093 ± 184	2452 ± 219	2087 ± 81	2019 ± 96	2163 ± 145

Table 4.15: Training Time (s) (AUC)

Chapter 5

Conclusions

This research confirms that incremental learning approaches offer promising directions for efficient caching research by providing substantial efficiency gains with manageable performance tradeoffs across various budgets and datasets. This is, to our knowledge, the first work that uses incremental online knowledge distillation for efficient caching of LLMs and reducing expensive calls to LLMs. Our work builds upon the research by Ramírez et al. (2023) by incorporating Incremental Learning (IL) methods, which offer greater efficiency compared to the complete retraining approach discussed in the original paper. Additionally, we extend the scope of the study to include efficient caching for dynamic data streams.

Incremental and EWC methods provide the most efficiency gains with minimal performance tradeoffs for static data streams. They are highly suitable for resource-constrained environments. However, these methods suffer from forgetting in dynamic data streams. Complete retraining and Replay perform the best across data streams. However, Replay increases computational efficiency by reducing FLOPs and training time by about 40% for static data streams and 30% for dynamic data streams compared to complete retraining.

While we've found EWC to be slightly beneficial for static data streams, we did not find it beneficial for dynamic data streams. Future research could investigate the effect of the standard EWC approach, which involves storing the entire history of parameters to preserve knowledge from all training iterations.

We find that AL-based policies enhance the performance and efficiency of incremental training methods compared to the static FR policy in both static and dynamic data streams. The performance gap between AL-based policies like MS and static policies like FR widens in dynamic data streams, confirming previous research (Bifet

and Gavaldà, 2007; Ramírez et al., 2023).

In conclusion, our research on incremental learning techniques shows a strong balance between performance and efficiency in neural caching of both static and dynamic data streams, aligning with incremental learning literature (Verwimp et al., 2023). This suggests that incremental learning approaches could enhance efficiency in diverse scenarios, opening up new avenues for future research.

Chapter 6

References

Aljundi, Rahaf, Babiloni, Francesca, Elhoseiny, Mohamed, Rohrbach, Marcus, and Tuytelaars, Tinne, 2018. Memory aware synapses: learning what (not) to forget. *Proceedings of the european conference on computer vision (eccv)*, pp.139–154 (cit. on p.9).

Amodei, Dario and Hernandez, Danny, 2023. *AI and compute* [Online]. Available from: https://openai.com/research/ai-and-compute [Accessed April 21, 2024] (cit. on p.8).

Biesialska, Magdalena, Biesialska, Katarzyna, and Costa-Jussa, Marta R, 2020. Continual lifelong learning in natural language processing: a survey. *Arxiv preprint arxiv:2012.09823* (cit. on p.8).

Bifet, Albert and Gavaldà, Ricard, 2007. *Learning from time-changing data with adaptive windowing*. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611972771.42. Available from: https://doi.org/10.1137/1.9781611972771.42 (cit. on pp.32, 35).

Bucilua, Cristian, Caruana, Rich, and Niculescu-Mizil, Alexandru, 2006. Model compression. *Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining* [Online], Kdd '06. Philadelphia, PA, USA: Association for Computing Machinery, pp.535–541. Available from: https://doi.org/10.1145/1150402.1150464 (cit. on p.5).

Cacciarelli, Davide and Kulahci, Murat, 2023. Active learning for data streams: a survey. *Machine learning* [Online], 113(1), pp.185–239. Available from: https://doi.org/10.1007/s10994-023-06454-2 (cit. on p.2).

Celikyilmaz, Asli, Clark, Elizabeth, and Gao, Jianfeng, 2021. *Evaluation of text generation: a survey* [Online]. arXiv: 2006.14799 [cs.CL]. Available from: https://arxiv.org/abs/2006.14799 (cit. on p.11).

Chaudhry, Arslan, Dokania, Puneet K, Ajanthan, Thalaiyasingam, and Torr, Philip HS, 2018. Riemannian walk for incremental learning: understanding forgetting and intransigence. *Proceedings of the european conference on computer vision (eccv)*, pp.532–547 (cit. on pp.9, 10, 21).

Delange, Mathew, Aljundi, Rahaf, Masana, Marc, Parisot, Sarah, Jia, Xu, Leonardis, Aleš, Slabaugh, Gregory, and Tuytelaars, Tinne, 2021. A continual learning survey: defying forgetting in classification tasks. *Ieee transactions on neural networks and learning systems*, 31(7), pp.2749–2765 (cit. on p.13).

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina, 2019. Bert: pre-training of deep bidirectional transformers for language understanding. arxiv. *Arxiv preprint arxiv:1810.04805* (cit. on p.6).

Gama, João, Žliobaitundefined, Indrundefined, Bifet, Albert, Pechenizkiy, Mykola, and Bouchachia, Abdelhamid, 2014. A survey on concept drift adaptation. *Acm comput. surv.* [Online], 46(4). Available from: https://doi.org/10.1145/2523813 (cit. on p.2).

Ham, Donghoon, Lee, Jeong-Gwan, Jang, Youngsoo, and Kim, Kee-Eung, 2020. End-to-end neural pipeline for goal-oriented dialogue systems using gpt-2. *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp.583–592 (cit. on p.1).

Harun, Md Yousuf, Gallardo, Jhair, Hayes, Tyler L., Kemker, Ronald, and Kanan, Christopher, 2023. *Siesta: efficient online continual learning with sleep*. arXiv: 2303.10725 [cs.CV] (cit. on p.8).

Hayes, Tyler L, Kafle, Kushal, Shrestha, Robik, Acharya, Manoj, and Kanan, Christopher, 2020. Remind your neural network to prevent catastrophic forgetting. *European conference on computer vision*. Springer, pp.466–483 (cit. on pp.10, 13, 21).

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff, 2015. Distilling the knowledge in a neural network. *Arxiv preprint arxiv:1503.02531* (cit. on p.5).

Hu, Edward J, shen, yelong, Wallis, Phillip, Allen-Zhu, Zeyuan, Li, Yuanzhi, Wang, Shean, Wang, Lu, and Chen, Weizhu, 2022. LoRA: low-rank adaptation of large language models. *International conference on learning representations* [Online]. Available from: https://openreview.net/forum?id=nZeVKeeFYf9 (cit. on pp.15, 43).

Kingma, Diederik P. and Ba, Jimmy, 2017. *Adam: a method for stochastic optimization* [Online]. arXiv: 1412.6980 [cs.LG]. Available from: https://arxiv.org/abs/1412.6980 (cit. on pp.15, 16).

Kirkpatrick, James, Pascanu, Razvan, Rabinowitz, Neil, Veness, Joel, Desjardins, Guillaume, Rusu, Andrei A, Milan, Kieran, Quan, John, Ramalho, Tiago, Grabska-Barwinska, Agnieszka, et al., 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13), pp.3521–3526 (cit. on p.9).

Liu, Xialei, Masana, Marc, Herranz, Luis, Van de Weijer, Joost, Lopez, Antonio M, and Bagdanov, Andrew D, 2018. Rotate your networks: better weight consolidation and less catastrophic forgetting. *2018 24th international conference on pattern recognition (icpr)*. IEEE, pp.2262–2268 (cit. on p.9).

Loshchilov, Ilya and Hutter, Frank, 2019. *Decoupled weight decay regularization* [Online]. arXiv: 1711.05101 [cs.LG]. Available from: https://arxiv.org/abs/1711.05101 (cit. on pp.15, 44).

Masson D'Autume, Cyprien de, Ruder, Sebastian, Kong, Lingpeng, and Yogatama, Dani, 2019. Episodic memory in lifelong language learning. *Advances in neural information processing systems*, 32 (cit. on pp.10, 13).

McCloskey, Michael and Cohen, Neal J, 1989. Catastrophic interference in connectionist networks: the sequential learning problem. In: *Psychology of learning and motivation*. Vol. 24. Elsevier, pp.109–165 (cit. on pp.7, 12).

Mihaylov, Todor, Clark, Peter, Khot, Tushar, and Sabharwal, Ashish, 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In: Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, eds. *Proceedings of the 2018 conference on empirical methods in natural language processing* [Online]. Brussels, Belgium: Association for Computational Linguistics, pp.2381–2391. Available from: https://doi.org/10.18653/v1/D18-1260 (cit. on p.14).

Nguyen, Cuong V, Li, Yingzhen, Bui, Thang D, and Turner, Richard E, 2018. Variational continual learning. *Arxiv preprint arxiv:1710.10628* (cit. on p.9).

Ni, Jianmo, Hernandez Abrego, Gustavo, Constant, Noah, Ma, Ji, Hall, Keith, Cer, Daniel, and Yang, Yinfei, 2022. Sentence-t5: scalable sentence encoders from pretrained text-to-text models. In: Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, eds. *Findings of the association for computational linguistics: acl 2022* [Online]. Dublin, Ireland: Association for Computational Linguistics, pp.1864–1874. Available from: https://doi.org/10.18653/v1/2022.findings-acl.146 (cit. on p.7).

Pang, Bo and Lee, Lillian, 2005. Seeing stars: exploiting class relationships for sentiment categorization with respect to rating scales. *Proceedings of the 43rd annual meeting on association for computational linguistics* [Online], Acl '05. Ann Arbor, Michigan: Association for Computational Linguistics, pp.115–124. Available from: https://doi.org/10.3115/1219840.1219855 (cit. on p.14).

Parisi, German I, Kemker, Ronald, Part, Jose L, Kanan, Christopher, and Wermter, Stefan, 2019. Continual lifelong learning with neural networks: a review. *Neural networks*, 113, pp.54–71 (cit. on pp.7, 8).

Ponti, Edoardo Maria, Sordoni, Alessandro, Bengio, Yoshua, and Reddy, Siva, 2023. Combining parameter-efficient modules for task-level generalisation. In: Andreas Vlachos and Isabelle Augenstein, eds. *Proceedings of the 17th conference of the european chapter of the association for computational linguistics* [Online]. Dubrovnik, Croatia: Association for Computational Linguistics, pp.687–702. Available from: https://doi.org/10.18653/v1/2023.eacl-main.49 (cit. on p.43).

Raffel, Colin, Shazeer, Noam, Roberts, Adam, Lee, Katherine, Narang, Sharan, Matena, Michael, Zhou, Yanqi, Li, Wei, and Liu, Peter J., 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *Corr* [Online], abs/1910.10683. arXiv: 1910.10683. Available from: http://arxiv.org/abs/1910.10683 (cit. on p.15).

Ramírez, Guillem, Lindemann, Matthias, Birch, Alexandra, and Titov, Ivan, 2023. Cache & distil: optimising api calls to large language models. *Arxiv preprint arxiv:2310.13561* (cit. on pp.i, 1, 5, 6, 11, 12, 14, 15, 28, 35, 36, 43, 45, 46).

Rebuffi, Sylvestre-Alvise, Kolesnikov, Alexander, Sperl, Georg, and Lampert, Christoph H, 2017. Icarl: incremental classifier and representation learning. *Proceedings of the ieee conference on computer vision and pattern recognition*, pp.2001–2010 (cit. on pp.9, 13).

Roy, Nicholas and McCallum, Andrew, 2001. Toward optimal active learning through monte carlo estimation of error reduction. *Icml, williamstown*, 2(441-448), p.4 (cit. on p.6).

Rusu, Andrei A, Rabinowitz, Neil C, Desjardins, Guillaume, Soyer, Hubert, Kirkpatrick, James, Kavukcuoglu, Koray, Pascanu, Razvan, and Hadsell, Raia, 2016. Progressive neural networks. *Arxiv preprint arxiv:1606.04671* (cit. on p.10).

Scheffer, Tobias, Decomain, Christian, and Wrobel, Stefan, 2001. Active hidden markov models for information extraction. *International symposium on intelligent data analysis*. Springer, pp.309–318 (cit. on p.6).

Schröder, Christopher, Niekler, Andreas, and Potthast, Martin, 2021. Revisiting uncertainty-based query strategies for active learning with transformers. *Arxiv preprint arxiv:2107.05687* (cit. on p.6).

Schwartz, Roy, Dodge, Jesse, Smith, Noah A, and Etzioni, Oren, 2020. Green ai. *Communications of the acm*, 63(12), pp.54–63 (cit. on p.17).

Sener, Ozan and Savarese, Silvio, 2018. Active learning for convolutional neural networks: a core-set approach. *International conference on learning representations* [Online]. Available from: `https://openreview.net/forum?id=H1aIuk-RW` (cit. on p.7).

Settles, Burr, 2009. Active learning literature survey (cit. on pp.1, 5, 6).

Seung, H. S., Opper, M., and Sompolinsky, H., 1992. Query by committee. *Proceedings of the fifth annual workshop on computational learning theory* [Online], Colt '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, pp.287–294. Available from: `https://doi.org/10.1145/130385.130417` (cit. on p.7).

Shao, Bo, Doucet, Lorna, and Caruso, David R., 2015. Universality versus cultural specificity of three emotion domains: some evidence based on the cascading model of emotional intelligence. *Journal of cross-cultural psychology* [Online], 46(2), pp.229–251. eprint: https://doi.org/10.1177/0022022114557479. Available from: https://doi.org/10.1177/0022022114557479 (cit. on p.14).

Stogiannidis, Ilias, Vassos, Stavros, Malakasiotis, Prodromos, and Androutsopoulos, Ion, 2023. Cache me if you can: an online cost-aware teacher-student framework to reduce the calls to large language models. *Arxiv preprint arxiv:2310.13395* (cit. on p.5).

Sun, Fan-Keng, Ho, Cheng-Hao, and Lee, Hung-Yi, 2020. {lamal}: {la}nguage modeling is all you need for lifelong language learning. *International conference on learning representations* [Online]. Available from: https://openreview.net/forum?id=Skgxcn4YDS (cit. on p.9).

Thorne, James, Vlachos, Andreas, Christodoulopoulos, Christos, and Mittal, Arpit, 2018. FEVER: a large-scale dataset for fact extraction and VERification. In: Marilyn Walker, Heng Ji, and Amanda Stent, eds. *Proceedings of the 2018 conference of the north American chapter of the association for computational linguistics: human language technologies, volume 1 (long papers)* [Online]. New Orleans, Louisiana: Association for Computational Linguistics, pp.809–819. Available from: https://doi.org/10.18653/v1/N18-1074 (cit. on p.14).

Verwimp, Eli, Ben-David, Shai, Bethge, Matthias, Cossu, Andrea, Gepperth, Alexander, Hayes, Tyler L, Hüllermeier, Eyke, Kanan, Christopher, Kudithipudi, Dhireesha, Lampert, Christoph H, et al., 2023. Continual learning: applications and the road forward. *Arxiv preprint arxiv:2311.11908* (cit. on pp.7, 36).

Wu, Yue, Chen, Yinpeng, Wang, Lijuan, Ye, Yuancheng, Liu, Zicheng, Guo, Yandong, and Fu, Yun, 2019. *Large scale incremental learning* [Online]. arXiv: 1905.13260 [cs.CV]. Available from: https://arxiv.org/abs/1905.13260 (cit. on pp.10, 21).

Zenke, Friedemann, Poole, Ben, and Ganguli, Surya, 2017. Continual learning through synaptic intelligence. *International conference on machine learning*. PMLR, pp.3987–3995 (cit. on p.9).

Appendix A

Experimental Details and Hyperparameters

Environment Details

For our research, we utilized GPU nodes equipped with NVIDIA Tesla V100 GPUs, each providing 16 GB of memory. Each GPU node is paired with dual-socket Intel Xeon E5-2695 v4 processors, offering a total of 36 cores per node and 256GB of RAM.

Student Model

We follow Ramírez et al. (2023) for the implementation of the neural caching process. We use the T5 implementation from Huggingface's transformers library, employing LoRA adapters (Hu et al., 2022), which are recognized for their parameter efficiency in few-shot settings. As suggested by Ponti et al. (2023), we incorporate a LoRA adapter into the query, key, value, and output weights in each self-attention layer of T5. The LoRA rank is set to $r = 16$ and scaled to $\alpha = 0.25$. The learning rate is $\eta = 5 \times 10^{-4}$, with a training batch size of $m = 16$ and weight decay $\lambda = 0.01$. These hyperparameters were validated based on experiments using the soft labels from the teacher by Ramírez et al. (2023). Additionally, the effectiveness of these hyperparameters has been validated for incremental learning.

Learning rate scheduler experiment

We chose to compare a linear scheduler with a linear scheduler with hard resets. In a linear scheduler, the learning rate decreases gradually across training iterations. On

the other hand, in a linear scheduler with hard resets, the learning rate is reset to the original learning rate for every training iteration.

Scheduler type	**Online Accuracy**	**Final Accuracy**
Linear with hard resets	0.629	0.587
Linear	0.625	0.580

Table A.1: Comparison of Online and Final Accuracy (AUC) for different scheduler Types (ISEAR dataset)

We ran the experiments in the ISEAR dataset with FR as policy, a retraining frequency of $f = 1000$ and plain incremental learning as the training method. The optimizer used in AdamW (Loshchilov and Hutter, 2019). The results can be seen in Table A.1. In terms of online accuracy, the linear scheduler shows only a minor decrease. For final accuracy, both schedulers are nearly identical. Since the linear scheduler has no disruptions from resets, it could lead to a more stable learning process. Hence, we chose a linear scheduler for all our experiments.

Hyperparameter experiments

An incremental neural caching setup is executed for three budget values 1000, 2000 and 3000. In this setup, a hyperparameter search for learning rate is performed and the results can be seen in Table A.2. From the results, we can observe that the optimal learning rate for incremental learning is the same as that of complete retraining.

Table A.2: Learning rate search: Incremental

Learning Rate	Online AUC		Final AUC	
	MS	FR	MS	FR
5e-3	0.626	0.609	0.537	0.552
5e-4	**0.640**	0.618	**0.578**	**0.578**
5e-05	0.626	0.608	0.561	0.563

Additionally, an incremental neural caching setup using replay is executed for three budget values 1000, 2000 and 3000 and a hyperparameter search for learning rate and weight decay is performed. From the results in Table A.3 and A.4, we can see that

Table A.3: Learning rate search: Replay

Learning Rate	Online AUC		Final AUC	
	MS	FR	MS	FR
5e-3	0.625	0.613	0.472	0.554
5e-4	**0.654**	0.634	**0.615**	0.600
5e-05	0.634	0.617	0.583	0.576

Table A.4: Weight decay search: Replay

Weight decay	Online AUC		Final AUC	
	MS	FR	MS	FR
0.1	0.651	0.636	0.608	0.601
0.01	**0.654**	0.634	**0.615**	0.600
0.001	0.651	0.635	0.605	0.602

the optimal learning rate and weight decay are the same as that of complete retraining. Based on the above experiments, we can observe that the optimal hyperparameters for incremental learning are similar to that of complete retraining even though the learning rate scheduler is reset before every training phase in case of complete retraining setup.

EWC

For EWC, a lambda value of $\lambda = 0.4$ is chosen based on a hyperparameter search on λ values on the ISEAR dataset with front-loading as a strategy. From the results in Table A.5, we can observe that the performance of FR improves and MS degrades for higher values of λ, hence we choose $\lambda = 0.4$.

AL policies

To apply the above AL metrics, in an online environment as a selection policy, the following thresholds are established: (PE $= 0.5$, MS $= 5$, QBC $= 4$, CS $= 0.9$) based on Ramírez et al. (2023). These values were chosen to ensure that the initial student model selects at least 50% of instances for LLM annotation on the RT-Polarity dataset. Only examples with a margin exceeding this threshold are selected until the budget is

Table A.5: EWC Lambda search

Lambda	Online AUC		Final AUC	
	MS	FR	MS	FR
0.2	0.655	0.632	0.595	0.595
0.4	0.656	0.632	0.600	0.594
0.6	0.656	0.633	0.600	0.594
0.8	0.654	0.635	0.592	0.597

depleted. For entropy, we first apply a softmax over the classes to normalize the values before calculating it.

Labels from the LLM

The labels are obtained by Ramírez et al. (2023) during May 2023. Due to the OpenAI API's limitation of returning only the top five most likely tokens, a bias of $b = 100$ was added to the tokens representing each class.

- ISEAR: 'joy', 'fear', 'anger', 'sadness', 'disgust', 'shame', 'guilt'
- RT-Polarity: 'positive', 'negative'
- FEVER: 'true', 'false'
- Openbook: 'A', 'B', 'C', 'D'

If a class did not appear among these five tokens, it was assigned a log probability of -100 in the dataset.

Prompts Used

The prompts used by Ramírez et al. (2023) when querying the LLM are as follows:

ISEAR: This is an emotion classification task. Only answer with one of: 'joy', 'fear', 'anger', 'sadness', 'disgust', 'shame', 'guilt'.

RT-Polarity: This is a sentiment classification task for movie reviews. Only answer with either 'positive' or 'negative'.

FEVER: This is a fact-checking task. Only answer with either 'true' or 'false'.

Openbook: This is a multiple-choice test. You are given a fact and a question. Only answer with one letter, providing no further output.

Incremental neural caching with AL policies

The experiment results for the incremental neural caching experiments conducted in Section 4.1.3 for all the AL policies are as follows.

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	0.646	0.725	0.880	0.736	0.747
	MS	0.665	0.727	0.890	0.747	0.757
	PE	0.657	0.718	0.887	0.734	0.749
	CS	0.649	0.734	0.888	0.724	0.749
	QBC	0.657	0.763	0.879	0.748	0.762
	Average	**0.655**	0.733	**0.885**	0.738	**0.753**
Incremental	FR	0.632	0.717	0.878	0.729	0.739
	MS	0.656	0.730	0.893	0.749	0.757
	PE	0.652	0.711	0.884	0.737	0.746
	CS	0.638	0.729	0.887	0.726	0.745
	QBC	0.650	0.758	0.879	0.748	0.759
	Average	0.646	0.729	0.884	0.738	**0.749**
EWC	FR	0.632	0.718	0.880	0.734	0.741
	MS	0.656	0.739	0.893	0.748	0.759
	PE	0.651	0.711	0.884	0.738	0.746
	CS	0.640	0.730	0.887	0.726	0.746
	QBC	0.650	0.758	0.879	0.747	0.758
	Average	0.646	0.731	0.885	0.739	**0.749**
Replay	FR	0.643	0.717	0.879	0.732	0.743
	MS	0.663	0.730	0.892	0.746	0.758
	PE	0.655	0.713	0.884	0.741	0.748
	CS	0.645	0.731	0.888	0.729	0.748
	QBC	0.652	0.758	0.879	0.747	0.759
	Average	0.652	0.730	0.884	0.739	**0.751**
Replay (50%)	FL	0.641	0.729	0.879	0.733	0.746
	MS	0.662	0.757	0.892	0.748	0.765
	PE	0.654	0.738	0.884	0.741	0.754
	CS	0.644	0.732	0.880	0.729	0.746
	QBC	0.653	0.758	0.888	0.749	0.762
	Average	0.651	**0.743**	**0.885**	**0.740**	**0.754**†

Table A.6: Online Accuracy (AUC)

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	0.612	0.640	0.884	0.683	0.705
	MS	0.619	0.637	0.887	0.683	0.707
	PE	0.620	0.656	0.888	0.676	0.710
	CS	0.615	0.657	0.886	0.683	0.710
	QBC	0.622	0.672	0.882	0.678	0.714
	Average	**0.618**	**0.652**	**0.885**	0.681	**0.709**†
Incremental	FR	0.593	0.624	0.879	0.677	0.693
	MS	0.599	0.623	0.882	0.689	0.698
	PE	0.598	0.634	0.882	0.682	0.699
	CS	0.596	0.641	0.882	0.675	0.699
	QBC	0.597	0.642	0.882	0.686	0.702
	Average	0.596	0.633	0.881	**0.682**	**0.698**
EWC	FR	0.594	0.635	0.882	0.680	0.698
	MS	0.600	0.633	0.882	0.674	0.697
	PE	0.594	0.636	0.881	0.686	0.699
	CS	0.600	0.642	0.880	0.670	0.698
	QBC	0.599	0.644	0.881	0.682	0.702
	Average	0.597	0.638	0.881	0.678	**0.699**
Replay	FR	0.609	0.624	0.882	0.684	0.700
	MS	0.615	0.636	0.885	0.681	0.704
	PE	0.611	0.634	0.885	0.680	0.703
	CS	0.607	0.655	0.885	0.681	0.707
	QBC	0.615	0.657	0.881	0.678	0.708
	Average	0.611	0.641	0.884	0.681	**0.704**
Replay (50%)	FL	0.607	0.646	0.881	0.683	0.704
	MS	0.610	0.653	0.887	0.683	0.708
	PE	0.609	0.654	0.886	0.683	0.708
	CS	0.605	0.649	0.882	0.678	0.704
	QBC	0.613	0.658	0.884	0.685	0.710
	Average	0.609	**0.652**	0.884	**0.682**	**0.707**

Table A.7: Final Accuracy (AUC)

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	6.69	5.95	4.29	4.89	5.46
	MS	9.14	8.91	10.35	8.75	9.29
	PE	11.52	14.63	7.32	8.59	10.52
	CS	6.89	11.81	8.64	7.77	8.78
	QBC	10.65	15.38	3.63	8.42	9.52
	Average	8.98	11.34	6.85	7.68	**8.71**
Incremental	FR	1.99	3.32	1.86	1.72	2.22
	MS	2.09	3.07	1.65	1.89	2.17
	PE	1.94	3.85	1.62	1.60	2.25
	CS	2.48	3.26	1.60	1.76	2.28
	QBC	2.00	3.48	1.75	1.99	2.31
	Average	2.08	3.20	1.69	1.79	**2.19**
EWC	FR	2.07	3.82	1.89	1.72	2.38
	MS	2.09	3.55	1.65	1.86	2.29
	PE	2.02	3.80	1.62	1.49	2.23
	CS	2.52	3.31	1.61	1.69	2.28
	QBC	1.99	3.38	1.79	1.95	2.28
	Average	2.14	3.57	1.71	1.74	**2.29**
Replay	FR	4.91	4.89	4.07	3.15	4.26
	MS	6.27	6.61	8.48	5.21	6.64
	PE	7.24	9.60	5.23	4.43	6.62
	CS	4.90	6.80	6.79	4.25	5.69
	QBC	7.52	8.70	3.98	5.10	6.32
	Average	6.17	7.32	5.31	4.43	**5.81**
Replay (50%)	FR	4.29	5.54	3.43	2.71	4.00
	MS	5.53	7.26	6.87	3.77	5.86
	PE	6.23	7.21	4.68	3.53	5.41
	CS	4.13	5.66	3.41	3.64	4.21
	QBC	5.95	6.78	5.95	3.79	5.62
	Average	5.23	6.49	4.87	3.49	**5.02**

Table A.8: FLOPS (E+15) (AUC)

Training method	Strategy	ISEAR	Openbook	RT-Polarity	FEVER	Average
Complete retraining	FR	2427	1808	1702	2509	2112
	MS	3447	2682	4096	4419	3661
	PE	4231	4541	2812	4754	4085
	CS	2550	3903	3524	4321	3575
	QBC	3841	5080	1579	4725	3806
	Average	3299	3603	2743	4146	**3448**
Incremental	FR	745	1012	777	936	868
	MS	883	969	718	1011	895
	PE	804	1188	671	874	884
	CS	910	1086	703	1003	926
	QBC	817	1145	739	1165	967
	Average	809	1016	713	985	**881**
EWC	FR	789	1168	786	979	931
	MS	952	1137	752	1031	968
	PE	875	1226	723	912	934
	CS	1008	1106	783	1078	994
	QBC	886	1212	796	1171	1016
	Average	902	1169	768	1034	**968**
Replay	FR	1743	1469	1657	1646	1629
	MS	2337	2008	3358	2704	2602
	PE	2654	2869	2054	2335	2478
	CS	1801	2158	2931	2346	2309
	QBC	2870	2749	1693	2845	2539
	Average	2281	2247	2339	2375	**2311**
Replay (50%)	FR	1593	1687	1310	1440	1508
	MS	2054	2191	2662	1979	2222
	PE	2411	2203	1919	1930	2116
	CS	1540	1739	1375	1936	1648
	QBC	2329	2261	2536	2193	2330
	Average	1985	2016	1960	1896	**1964**

Table A.9: Training Time (s) (AUC)