TRACER: Task-aware Risk-adaptive Architecture for Continual Edge leaRning

by

Md.Hasibul Islam
20301298
Mir Muhammad Fuad
20101546
A.B.M. Fahim Hasan Tanzin
21101257

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
February 2026

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Hasibul

Md. Hasibul Islam
20301298

Mir Muhammad Fuad

Mir Muhammad Fuad
20101546

Fahim

A.B.M. Fahim Hasan Tanzin
21101257

Approval

The thesis/project titled " " submitted by

1. Md.Hasibul Islam (20301298)
2. Mir Muhammad Fuad (20101546)
3. A.B.M. Fahim Hasan Tanzin (21101257)

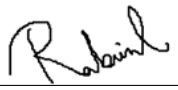
of Fall, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science in Spring 2026.

Examining Committee:

Supervisor:

Dr. Md. Khalilur Rahman
Professor
Department of Computer Science and Engineering
Brac University

Co-Supervisor:



Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi
Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

Current computer-vision architectures are being deployed as long-lived services, especially on edge and on-device platforms, where input distributions change with changes in environment, users, sensors, and class frequencies. In these cases, to achieve sustainable performance, continual learning is required. Also, we need to keep in mind that the process needs to be feasible under strict constraints like latency and memory. Previous experience demonstrates that device-centric measures of deployment efficiency should be used instead of proxy metrics like FLOPs, and that tail latency (e.g., p95) is a more constrained measure of deployment efficiency than mean latency. At the same time, full neural architecture search (NAS) is generally too costly to integrate into a repeated learning loop, motivating restricted, hardware-aware search strategies.

This thesis presents TRACER: Task-aware Risk-adaptive Architecture for Continual Edge leaRning, a deployability-oriented continual learning pipeline that keeps a fixed feature backbone and repeatedly selects and adapts a lightweight MLP classifier head. The system follows a restricted design space with a NAS-inspired controller and a Net2Net-optimized evolutionary population so that it can adapt efficiently. The stability between tasks is ensured through risk-aware exemplar rehearsal (high-risk samples are prioritized) and knowledge distillation. Experiments on Split CIFAR-100 (10 tasks x 10 classes) and CIFAR-10 (5 tasks x 2 classes) report class-incremental (CIL) and task-incremental (Task-IL) performance. Our proof-of-concept implementation has a final CIL mean accuracy of 0.8145 and average forgetting of 0.0341, and TIL has a final mean accuracy of 0.970 on CIFAR-10. Also, in CIFAR-100, we got 0.6136 final CIL mean accuracy, and TIL has a final mean accuracy of 0.9055 with 0.0451 forgetting. These results, which are derived from a single deterministic run, demonstrate that Lagrangian-relaxation-based constraint-aware head selection, combined with risk-sensitive stabilization, provides a practical accuracy-feasibility trade-off for continual learning under explicit latency targets.

Keywords: Continual Learning; Class-Incremental Learning; Task-Incremental Learning; Edge AI; Latency Constraint; Neural Architecture Search (NAS); Reinforcement Learning (RL); Exemplar Replay; Knowledge Distillation (KD).

Acknowledgement

We would like to extend our heartfelt appreciation to all those who have contributed, directly or indirectly, to the successful accomplishment of this thesis.
First and foremost, we express our deepest gratitude to Almighty Allah for granting us the strength, patience, and countless blessings that enabled us to carry out this research. Without his infinite mercy and blessings, this accomplishment would not have been possible.

We are sincerely thankful to our supervisor, Dr. Md. Khalilur Rahman, for his constant guidance, valuable insights, and unwavering support throughout this research. We are equally grateful to our co-supervisor, Dr. Md. Golam Rabiul Alam, for his continuous encouragement, thoughtful suggestions, and mentorship during the entire course of this work. Their expertise and constructive feedback played a vital role in shaping the direction and quality of this research.

Finally, we would like to thank the Department of Computer Science and Engineering, BRAC University, for providing us with the necessary facilities, resources, and academic environment that made this study possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Computer vision has been shifting off the train once, deploy forever model, and towards systems that are more like long lived services. This is particularly accurate in edge and on-device applications: smart cameras, assistive, embedded monitoring, and mobile vision applications, where the data stream is ultimately varying. Environments change, there is a modification of lighting conditions, sensors become old, and there is drift of class distributions. Within such realities, it is not only a matter of getting good benchmark accuracy on a single occasion; but it is also a matter of maintaining valuable performance in a dynamic world. This is a wider conceptualization to explain the reason why continuous learning has become central in applied vision research [11], [12], [18].

But there is one way of continuing learning, which is called the catastrophic forgetting range. Neural networks which learn new tasks tend to overwrite internal representations, decision boundaries which were previously significant in other tasks. This issue is further enhanced in class-incremental learning (CIL), the system is required to make predictions in the presence of all the classes observed to date, but no task indicators. According to the survey work, the ongoing performance of learning should be viewed through the prism of maintaining its levels throughout time, not merely using the latest performance of the task [11], [12].

The situation is more constrained with edge deployment. Although a given learning approach can make teaching more accurate, it can be of no use when it exceeds the latency threshold. The most prominent practical observation made by the edge literature is that proxy performance metrics such as FLOPs are not predictive of efficiency. The speed of inference is a strongly measured value, and it is very much dependent on the implementation details such as operator kernels, memory bandwidth, and runtime scheduling. It is due to this that metrics such as high-percentile latency (p95) are treated as deployment-centric metrics but are increasingly being viewed as the metrics that actually matter in the real world [3]. This is vital when updates arrive repetitionally; even quality improvement can push a system slowly to infeasible performance space.

This is the strain which drives us to work. Platform-aware schemes (such as NetAdapt) believe that the device should be within the optimization loop [9]. Hardware-conscious NAS techniques put latency as the first-class goal[10]. However, it is impractical to implement full NAS too often in a continuous learning process due to

high cost. One-shot and weight-sharing ideologies, on the other hand, indicate that limiting search space can conserve value, as well as regulate, cost [8], [14]. There are these hints of a direction: maintain the contact to the costly representation fixed (a fixed backbone) and change a small part (the head) in a manner that will be structured.
Constant studies alone through research lead to a similar approach. Powerful baselines are based on rehearsal and distillation combinations to stabilize updates [6], [7], [13]. Moreover, feature level replay views focus on the utility of adapting to lower dimensional space[17], [20]. This is just a natural fit with a fixed-backbone architecture in which features are reuseable and where the cost of adaptation is put into a lightweight head.
These realities come to an overlap in our thesis system. We investigate such a compromise between a NAS inspired but purposefully decorated backbone of more strongly realistic search through the selection of lightweight head variations within a p95 time limit, risk-aware continual learning stabilizers to reduce forgetting.nts.

1.2 Research Problem

When deployment is taken seriously, the research problem arises. In the case of an edge-use CL system, the CL system cannot be evaluated solely based on learning new classes; it must be deployable. The proxies are not known to reliably reflect constraints, as made clear by the literature. Hardware-aware NAS works indicate that ultimately the latency measurement is what counts tail latency (p95/p99) is a key requirement. [3]. This is reinforced in the serving literature, which makes tail latency (p95) one of the critical requirements of interactive pipelines[3]. The former half of the issue is to make updates safe in terms of the ability to tail latency.
The second one is computational feasibility. Full architecture search is too expensive to be repeated. In any CL loop in which information propagation is repeated, the system must have an adaptation mechanism whose cost of exploration is minimal. Other ways to experiment with architectures, including Net2Net (transformations with function preservation) [5]., can give a means of widening layers or deepening the network without retraining afresh, but such are rarely used in CL pipelines.
In the meantime, noise is brought about by continuous learning. Sequential training comes along with the short update windows and non-stationary distributions. When the only criterion of selection is the mean accuracy, then the selected configuration may be shaky–it may crash at difficult data subsets. It is here that risk-sensitive concepts come in. CVaR-type targets (Conditional Value at Risk) target tail outcomes as opposed to averages [1], [4]. In order to manage this, our system uses a CVaR-like reward value strategy that puts weight on doing better on more challenging validation batches where keeping them healthy to distributional changes.
Another type of modeling tension is a modeling tension: the most evident stage of the classifier is the stage of drift and bias. Although this is alleviated by the replay and distillation [6], [7], [13], it generally presupposes a fixed architecture. To optimize towards the feasibility we must have a change in capacity (e.g. by making the head wider) that does not destabilize the representation.
The research problem may be put as follows: How can a continual learning system be class-incrementally accurate on sequential tasks and explicitly constraints by a tail-latency constraint, with an adaptation mechanism small enough to be re-executed

on a repetitive basis? To answer this, this thesis postulates a limited, Net2Net enhanced mechanism of head-selection to be embedded in continual learning.

1.3 Research Objectives

The major goal is to come up with an ever-evolving learning pipeline that can be sustained at a specific latency-constrained setting as it learns sequentially.
Latency Integration Latency We embed deployment-driven signals, especially p95 latency, in the process of making selections using Lagrangian relaxation or a closely related constraint-based device [3].
Efficient Exploration: Ensure efficiency Before exploration: Make adaptation Mask selection: Use Net2Net-style mutations[5]. instead of an entire backbone search .
Stabilization: Decrease the forgetting through risk-conscious rehearsal (with excessive emphasis on high-loss samples) and knowledge distillation, post-process survey outcome of recognized CIL arranges [11], [12], [18].

1.4 Scope and Limitation

This thesis is limited by particular design decisions related to datasets, latency model and architecture impairments. To start with, the analysis concentrates on classifying images in case of class-incremental learning on the basis of Split CIFAR-10 benchmark (5 tasks $\times$ 2 classes). Although this standard protocol enables one to examine the forgetting and stability in detail, it should be noted that the results can be taken as a sign that the procedure is possible in a specific environment but not as the reason to consider that the behavior will be the same under large-scale datasets like ImageNet. Second, the system uses a strict feature backbone, to enhance performance and minimize catastrophic interference; thus, job loads that cause substantial changes to the feature extractor itself are beyond the capability of this work to design.
More so, the reported measures of feasibility are based on a simulation profile of Jetson that is correlated. It is to be understood that the computation of latency values is based on scaling observed GPU inference times by device-specific values which are required by published hardware benchmarks, and not by physically updating the model to a specialized edge device. Although, such method fits the literature of hardware-aware NAS, milliseconds that are reported are not to be interpreted at all as certain assurances that you will be able to implement a certain physical setup but are to exist as a demonstration of feasibility under specific conditions. Lastly, we admit a complexity trade-off, the complexity of using a controller-population mechanism over lower complexity fixed-architecture baselines. It does not offer the best possible solution as a universal solution, the paper is an experiment with space, changing its constraints as necessary to guarantee the hard conditions.

Chapter 2

Literature Review

Continual learning (CL) is a learning algorithm that resolves the issue of machine learning systems operating in evolving settings where the data are received in a sequential fashion and the distribution underlying the data may change over time[22], [27]. Compared to traditional training approaches based on offline training, a model is optimized on a still dataset and all conditions are held constant (training and testing conditions), continual learning focuses on the capacity to absorb new data and retain beneficial information on the knowledge that has already been learned [22]. The most important challenge in this system is catastrophic forgetting, in which learning a new model with new data can significantly reduce the performance of the model on old data, and this introduces a tradeoff between plasticity (learning new information) and stability (not losing old capabilities)[22], [27]. Since real deployments tend to introduce new limitations like finite memory, computational, and extreme runtime costs, continual learning studies have identified voluminous amounts of techniques which endeavor to maintain performance whilst being executable within realistic constraints[15], [19], [25].
In classification-oriented continual learning, scientists typically investigate the situations where the set of classes is enlarged through a series of learning stages, and similar variations in which either the task can vary by field or the organization of the label space[13], [22], [27]. These situations have promoted generalized procedures and values that summaries the general performance and retention conduct, such as average accuracy throughout the learning path, and explicit forgetting scores[22], [25]. Concurrently, variations in assumption, e.g. knowledge of task boundaries, task identification provided in the inference process, the ability to store past information or replay it, may significantly alter the difficulty of the task and the conclusions made based on the empirical findings[13], [21], [22]. Consequently, survey and benchmarking have begun focusing more on such survey designs, fair comparisons and reporting of cost of resources as well as accuracy [22], [33], [34].
This literature review is inclined to concentrate on the ongoing developments that are the most related to the continual image and multimodal learning within the conditions of realistic constraints [32], [33], [35]. It begins by summarizing the views of the surveys formalizing constant learning problem environments and practices of evaluation [22], [27], [33]. It then compares the representative methodological families such as replay based methods that relies on revisiting past experience [13], [17], [23], [24], distillation and regularization methodologies that restrict model drift, as well as parameter efficient varieties of adaption methods (such as prompting and

modular updates) that aims to reduce the number of interferences but limit the volume of trainable state. The review also reports the study on edge-oriented efficiency, such as device-conscious optimization, memory-conscious learning, and latency models to facilitate the practical execution on resource-constrained environments since several contemporary systems have limitations on deployment as well. The combination of these works offers the conceptual and empirical basis, which is required to comprehend the present trends and trade-offs of continual learning research, and which inspires the methodology options discussed in later chapters [22], [33].

2.1 Related Works

A survey by De Lange et al., [22] formulates continual learning into the problem of reusing a classifier across a series of tasks in a manner that does not require retraining the model, does not involve catastrophic forgetting when training on new data like with neural networks, etc. The paper does not discuss all the continual learning environments as equals, but it rather concentrates on task-incremental classification when the tasks come in a sequence and are separated by distinct task demarcation. It is in this range that the authors supply a systematic taxonomy of the currently extant approaches and suggested a framework on how a continuing learner reasons on how to maneuver the stability-plasticity trade-off across time. They also summarize a massive experimental paper compared between 11 state-of-the-art continual learning variants and 4 baseline variants on a variety of benchmarks, such as Tiny ImageNet, a large-scale unbalanced iNaturalist condition, and a stream of recognition dataset. According to the study, not only the algorithmic family is important to the results, but also practical and experimental issues of model capacity, weight decay and dropout regularization, as well as the sequence of task presentation are significant. Besides accuracy focused comparisons, the survey also talks of methods in terms of operational restrictions and makes qualitatively contrasting descriptions in the terms of what it requires in terms of memory, computation time, and storage requirements, which also aids in understanding trade-offs that are not apparent based on performance scores.

Zhou et al. [27] survey continuing learning in the age of pretrained model (PTM), as they claim that most real world systems have to learn with streaming data and cannot afford catastrophic forgetting and that new PTMs encourage a transition where training does not required starting with a blank slate, but instead a model is used to make changes by building itself around a strongest initial pretrained model. The authors classify the current approaches to continual learning that use PTM into three parts namely prompt based, representation based and model mixture based and how each of these parts exploits the structure and generalization of PTM differently with advantages and disadvantages. Besides being a definition source, benchmarks, and protocols, the survey offers an empirical research that compares representative possibilities within these categories and points out practical problems that may have impacts on the interpretation of comparisons such as the issues of fairness within an evaluation arrangement. In the case of experiments, they standardize the backbone by pre-training a Vision Transformer on ImageNet21K and test on seven datasets, namely: CIFAR100, CUB200, ImageNet-R, ImageNet-A, ObjectNet, OmniBenchmark and VTAB, which they select partially because a

number have significant domain gaps compared to ImageNet-style pretraining. They provide performance report on last-stage accuracy and average accuracy between stages and they present general conclusions like strong PTM representations may result in weak baselines being surprisingly competitive. Lastly, the paper provides future trends, such as continual learning of large language models, going beyond single-modality recognition, education under constrained computational needs, and benchmarks that are more reflective of truly new knowledge as opposed to what PTMs might have already learned.

Shim et al. [23] introduce Adversarial Shapley value Experience replay (ASER) as an online class-incremental continual learning method, inspired by the need to execute image-based deep learning though limited memory footprint, power, and computation on personal and embedded devices and still learn constantly on streaming data without catastrophic forgetting. The paper is about one of the most difficult problems that the data are received as a stream (a sample is only viewed once), tasks pose new classes with time progression, and the evaluation is done in the single-head mode where the model is expected to predict on all the classes that have been seen but the identity of the task is not known during the inference. In this replay-based environment, the authors view memory management as an important bottleneck and propose that the selection of what samples to replay among the others stored must satisfy two opposing requirement, namely stability (ineffective boundaries of decisions made in previous classes) and plasticity (boundaries of the decision being induced in new classes). To overcome this, ASER is an algorithm to map memory candidates into an adversarial perspective of the Shapley value in the latent space of the model that samples of an important assistance to hold previous boundaries and at the same time, the sample is adversarially placed to the current incoming sample, besides applying a Shapley-like approach to retrieval, it applies a similar strategy to update memories. The outstanding technical element is that it employs effective KNN-Shapley value calculation so that the data contribution approximation of the selection is possible within an online learning loop, in which the precise Shapley value computation would be otherwise prohibitive. It is tested on common continual learning tasks as Split CIFAR-10, Split CIFAR-100 and Split miniImageNet and compared to a set of common baselines including the regular version of experience replay, as well as variants which utilize regularization, and regularization-free memory-based methods, and evaluated by or averaged accuracy and averaged forgetting. In these experiments, the authors state that ASER (and its mean-variant ASER‘) can outperform competitively or better, with especially the higher the complexity of datasets and the smaller the memory buffer, and that more complex retrieval/update scoring attracts more training-time costs compared to more simple replay baselines.

Smith et al. [31] suggest Adaptive Memory Replay as a strategy that improves on the modern foundation model, where the large pretrained models require updating when new data is received, but replaying all the previous data in an unthoughtful way may be too costly and too compute-intensive. Contrary to a lot of earlier continual learning research that operating under a limited replay buffer assumes that past data is expensive to store, this paper suggests an alternate and more realistic regime: the cost of storage is now inexpensive enough to store all the old data, but

calculation is the real constraint, and then the question to ask is what samples to replay in order to meet a tight training-time constraint. In order to apply this concept, they construct replay selection as a non-stationary multi-armed bandit allocation problem: past information is categorized into clusters (in their implementation, clusters represent previous tasks), the mechanism approximates which clusters experience most current forgetting andolly they use Boltzmann sampling to sample replay data in a manner that responds to the current task and is updated by new beforehand this enables adaptation to the current task and improved adaptation to the changing model. Notably, they also define a zero-cost protocol that is supposed to ensure that training-time overhead is kept low by substituting some of the new-task samples with chosen replay samples such that the total number of examples processed is held constant. In both vision continual pre-training experiments, and language-model continual pre-training experiments, the article shows that adaptive replay achieves lower end self-logical loss and forgetting compared to full-memory iid replay baselines, and that the 0-cost version keeps most of these gains with no sacrifice to training efficiency.

Wang, Herranz and van de Weijer present the online learning algorithm ACAE-REMIND [24] that processes non-IID data streams in which each sample is only observed once (except stored exemplars) based on the practical application of the technique to devices with limited resources like mobile applications and robots. The paper builds upon the finding that feature replay can be more effective than raw image replay, and that, thanks to significantly less memory usage, a feature-replay system can be made to be feature replication more than feature replay, the paper reconsiders the case of compressed feature replay and suggests that one limitation with previous-style feature-replay systems is that they need to rely on either a fixed or minimally trainable backbone, and hence may be incapable of learning representations based on subsequent tasks and may equally have an adverse effect on distinguishing between familiar and unfamiliar classes. To overcome this, ACAE-REMIND adds an auxiliary classifier auto-encoder (ACAE) and product quantization (PQ) to REMIND pipeline such that replay can be executed at intermediate network layers with high level of compression so that more layers that follow the replay point can be trained simultaneously without incurring high expense in replay memory. The general process is outlined as follows: the model is initially primed on the initial task, followed by the ACAE being trained using a reconstruction task and an auxiliary classification loss that ensures that discriminative content is retained by compression and eventually PQ is trained to encode compact representations of the input in form of integer codes that are easily replayed during the online stream. In online learning, the technique stores the latent codes in a pool that has been compressed, samples the representations that have already been stored, and recreates the intermediate features using the decoders and updates the learnable part of the network based on a standard classification loss whilst the previous modules stay fixed. The authors assess ImageNet-Subset, CIFAR-100, CIFAR-10, and include the popular continual learning measures of average classwise accuracy and post-task accuracy, and evaluate several task schedules of few and many-step task splits to assess the different challenges of online class-incremental learning. Their experimental study involves comparisons to both the online continual baselines and popular offline incremental ones and also includes ablabes, which show that the aux-

iliary classification loss and the selection of replay layer can affect performance with various task granularities and memory budgets.

Dark Experience Replay (DER) is a rigorous and deliberately unsophisticated baseline of continual learning, as suggested by Buzzega et al. [13], based on the finding that much of the standard experimental procedure is pegged on task delineation and offline training sessions that are unlikely to match the realities of a streaming environment. They model their scenario of interest as General Continual Learning (GCL) in which the data stream can not be neatly divided into tasks and domain and class distributions can either shift slowly or jolt suddenly, making it challenging to be able to pick up or count on task transitions during training. In this context, DER is based on the simple concept of experience replay along with knowledge distillation, only that the labels are replaced with the logits of the model to which the buffered samples are stored and, finally, with later, the current model is encouraged to align its actions with the logits of that previous model. In order to be compatible with unclear boundaries and endless memory the technique adopts reservoir sampling to fill the buffer periodically, such that a limited footprint can be maintained, and without needing any a priori information about the length of the stream or the identity of the tasks in the stream. Another addition that the authors make is DER++, which incorporates an additional term that learns on top of the replayed samples ground-truths to alleviate the problems that may occur when the older logits are not so much trustworthy, and yet the replay framework remains the same. They test DER/DER++ in various tasks of never-before seen learning including Task-IL, Class-IL and Domain-IL, benchmarks formed by MNIST variants (Permuted/ Rotated) and its variants in sequential learning protocols (Sequential MNIST and CIFAR-10) and sequential learning of tiny images (Sequential Tiny ImageNet) and compare against well-known architectural, regularization, and rehearsal methods including PNN, EWF, SI, LwF, ER, MER, GEM/A-GEM As a more accurate model of the properties of GCL, they also introduce MNIST-360, a stream with changes of class combined with smooth distribution drift (through increasingly large rotations) that specifically encourages methods reliant on task-boundary cues. In addition to the results of accuracy, the article presents the practical analysis of correlating performance with memory footprint and comparing wall-clock training time of the methods based on replay and focusing on the idea that the proposed baseline must be viable under the limitation of memory and online-learning.

Prabhu, Torr, and Dokania [21] offer GDumb (Greedy Sampler and Dumb Learner) as a fairly naive baseline that raises the question of whether the new achievements in continual learning as it applies to classification might be owed in part to simplistic assumptions and test conditions instead of solutions of real usefulness. This paper initially suggests a broad generalization of continual learning where marked examples come in as a stream, that the set of observed classes may widen with time, and that a new learner should be capable of classifying not only the input to products in known classes but also an understanding that some test products may lie outside the learned label space - relating continual learning to the concepts of open-set recognition. It then breaks down popular variants of continual learning by more precisely defining what such variants rely on to simplify them, including: disjointed task boundaries, access to task identity during inference in task-incremental tasks,

and the differences between online and offline learning due to the possibility of revisiting samples all argue that minor modifications on such assumptions can have drastic impacts on problem difficulty and reduce cross-paper comparisons to unreliability. It is in this context that GDumb is described as a minimum-assumption, a greedy approach for storing a fixed number of samples in the stream as it aims to make the memory balancing in the classes and when the evaluation is needed, it will train a classifier on fresh memory on-the-fly as the particular stream progresses in contrast to continuously retraining the model on-the-fly. The method also helps to support class-incremental evaluation as well as task-incremental as an inference-timeies masking action over a set of output chances so that the same stored memory is examined at various continual-learning arrangements. The authors find that, on a large range of continual-learning formulations and benchmarks (splits of MNIST, CIFAR-10/100, SVHN, Tiny ImageNet under varying memory constraints), such a simple strategy achieves state-of-the-art performances in many cases — by large margins in many cases - without having been designed to take advantage of the specific mechanisms that more complex continual-learning algorithms can employ. Their interpretation of these findings is that evaluation practices, metric selections, and underlying assumptions may sometimes lead the continual-learning benchmarks to be easier to solve than they are supposed to be, and that they should be formulated more precisely and with stronger baselines in order that the reported gains be more indicative of progress on practically significant continual learning.

The suggestion by Douillard et al. [16] is that image incremental (class-incremental) learning can be achieved using PODNet (Pooled Outputs Distillation Network), based on the fact that many existing methods continue to face difficulties controlling catastrophic forgetting in long series of tasks, particularly when the tasks themselves are small, and presented in a large number of steps. The paper assumes incremental learning to be the limited environment in which the learner is able to access all information on the present task but it is required to maintain the information on the previous task with limited memory of exemplars belonging to previous lessons, the latter being associated with some practical considerations, including privacy, as well as memory and compute capacity constraints that prevent complete retraining. To tackle forgetting with those constraints, PODNet proposes two fundamental concepts: first, a distillation loss that not only limits the development of representations at the final embedding, but also in the rest of the convolutional architecture, by using pooling-based summaries of feature maps to balance rigidity (remembering old classes) and plasticity (learning new classes); and second, a Local Similarity Classifier (LSC) that the representation of each class by a set of proxy vectors, in order to be more effective at capturing the intra-class variance and insensitive to changes in the distribution The authors test PODNet in a small-tasks regime that had only been studied by previous literature (partially) and testing experiments on up to 50 incremental steps with a hard constraint on memory per-class. Performance is compressed by a dominant average incremental accuracy, point out accuracy after each phase in the entire learning programme, and findings are given on CIFAR100, ImageNet100, and ImageNet1000, and compared to placed baselines, including iCaRL, UCIR, and BiC. In this analysis, the paper records evidently better gains when compared with earlier techniques such as improvements with increased steps which at the same time adheres to the fact that the method is also effective in varying mem-

ory budgets as the relative improvements could increase as the amount of memory available to each class diminishes.

The study by Merlin et al. [25] reviews the replay-based continual learning through an implementation and evaluation lens, where the the author stated that, although the replay-based approach is commonly ranked among the most useful options in practice in alleviating catastrophic forgetting, the majority of crucial design-related aspects, including the size of the buffer, the time-dependent weighting of replay samples, and augmentation, are not always studied across the literature. An empirical investigation that forms the main contribution of the paper is in lieu of the hypothesis of introducing a new replay algorithm, which is in fact a structured empirical study aimed at creating practical recommendations on replay system construction that provides a superior efficiency-efficacy trade-off. The authors compare several popular rehearsal methods, including standard Replay, GDumb, iCaRL, and GSS, on the open-source Avalanche framework, and test such strategies in a new classes scenario on common continual learning tasks, including Split-MNIST, Split-CIFAR-10, Split-TinyImageNet, as well as CORe50-NC, using standard architectures. A large portion of their analysis is devoted to memory size, in which they experiment canonically with the size of a buffer, and found that the size of training required to be useful in a variety of tasks is not merely a function of dataset size, but also of factors such as difficulty of tasks and the number of classes involved, whose results indicate that quite small portions of a training set can still be usable to provide useful accuracy in a variety of tasks. They also address task-level replay weighting policies and discover that the balanced strategy is the most robust in general, and strategies that prioritize recent and middle-distance tasks are able to be competitive as well as provide an insight into which past events can be most useful in the learning process. Lastly, this paper also examines data augmentation on replay samples, citing its ability to prove the most productive in cases where memory is extremely low and ends by offering advice on how a replay system should be configured and compared to achieve the reported performance results being due to not just the algorithm selection, but rather in addition to the key implementation decisions which have a convincing impact on continual learning outcomes.

Wang et al. [26] introduce DualPrompt as a no-rehearsal continual learning architecture, which trains a sequence of tasks with one model 6) and does not catastrophically forget or store prior examples due to the practical constraints of rehearsal buffers relating to privacy and limited memory. The algorithm is based on the premise that there is a powerful pretrained-transformer backbone that is stored frozen and constantly adapted by training a small number of additional parameters known as prompts which are high-level playing-field-instructions to reuse pretrained representations instead of re-learning features all the way to zero. DualPrompt derives inspiration in Complementary Learning Systems, and argues that separate prompt pools are suboptimal since they do not explicitly differentiate between the knowledge that is transferable, encoded in a general prompt space called a General Prompt (G-Prompt), and the task-specific instructions embedded in one task-specific prompt space called the Expert Prompt (E-Prompt). To enable class-incremental inference where the identity of tasks is not known at test time, this method identifies each task with an E-Prompt by assigning it a learnable task key and in training

involves a matching loss to ensure that task keys represent their tasks. Another point that has been highlighted in the paper is that the location and placement of the prompts in the transformer are important: the paper examines prompting the prompts to various levels in multi-head self-attention architectures and compares prompting abilities, and finds that such architectural decisions can have significant impact on continual learning performance. To assess it, the authors use the challenging class-incremental training and evaluation with defined tasks boundaries during training according to which they evaluate performance in terms of standard continual learning measures such as mean accuracy and forgetting on standard benchmarks such as Split CIFAR-100 as well as add the Split ImageNet-R as a more difficult benchmark that stresses methods that do not rely on rehearsal. They cluster baselines in terms of being rehearsal-buffer size and elaborate on how representative of rehearsal-based techniques can deteriorate exponentially as the buffer becomes smaller, as a way to encourage rehearsal-complete techniques which are not based on the need to store past information. Altogether, DualPrompt is framed an emphasis on continual learning as a lightweight, prompt-based alternative that decouples task-invariant and task-specific communication in a frozen pre-trained model and is demonstrated by experiments to provide an explanation of prompt design choices and able to measure the difficulty of rehearsal-free continual learning.

To make on-device learning practicable even given the severe constraint of edge-memory Cai et al. [34] suggest Tiny-Transfer-Learning (TinyTL), contending that the limiting factor during training is not the number of parameters to be trained but the activation memory needed to run backpropagation. The paper finds, based on a direct computation of backward computation that there are two key reasons why in order to compute these weight gradients it is important to store the intermediate activations, but that in order to compute bias-related terms it is unnecessary to store the large activations in memory and this accounts for its choice of doing the training by simply freezing weights in the pre-trained feature extractor and updating only the Brian-related modules. Since the simplest form of bias-only adaptation is too limited, this paper presents a memory-efficient lite residual module that trains to optimize intermediate feature map refinements by learning small residual changes with no storage of activations by using a combination of design choices like downsampling and group convolutions. In transfer-learning configurations, the approach is found to be superior to standard methods of transfer learning, guided by pretrained models to a variety of downstream vision tasks, where results are measured against popular transfer baselines and the training-memory footparency of the training process is quantified. In addition to the quantifiable core bias/lite-residual methodology, the paper also talks about feature extractor adaptation with a once-for-all network such that the frozen backbone itself could be chosen to be more fit to any particular target dataset, where pretraining performance is not reliably predictive of transfer performance and various tasks can have preferences over different backbones. Lastly, the authors point out that TinyTL can be made to be competent even with extremely tiny batch sizes, with subsequent discussions both minimizing memory to hide as small footprints as caches, and they establish that results as a step towards practical continual or local adaptation on edge devices without the need of cloud retraining.

Minhas et al. [36] Goes into the problem of neuromorphic continual learning in

an embedded AI systems, concern with the way that spiking neural networks can continuously adapt to changing environments without catastrophic forgetting, a challenge particularly when devices have strict constraints on latency, energy, and memory. The paper notes that although memory replay is a popular and effective method for retaining old knowledge on continual learning operations, the state-of-the-art approach of a neuromorphic replay requires long timesteps coupled with repeated compression-decompression procedures that might cause significant overhead to deploy on a tightly constrained platform such as agents in mobile robotics. To investigate this limitation, the authors go on to provide a case study demonstrating that aggressively sampling timesteps less can have a detrimental effect on accuracy, studying "a fundamental tension between enhancing efficiency and maintaining learned knowledge in spiking network replay systems." In response, they propose Replay4NCL, which is a replay-based methodology that reduces the data size of the latent data stored in memory and replays that data with smaller timestep settings during training to reduce processing latency and energy use while adding compensating adjustments so that failing to have small spikes will not collapse learning quality. implicatively represent three coordinated design steps: timestep optimization to minimize the compute and latent size, parameter adjusting to match better the reduced spike regime and a latent replay insertion strategy-where where to get the replay data into the network with maximum retention and learning effectiveness. The evaluation is performed in a class incremental scenario on Spiking Heidelberg Digits dataset in surrogate gradient trained spiking neural network along with not only accuracy but also latency, latent memory size, and energy consumption, to reflect the embedded system constraints. Experiments throughout the paper found that Replay4NCL is able to maintain old knowledge while continuing to learn new activities, and are reporting large system-level improvements in terms of both efficiency and accuracy when compared with the baseline experiment studied.

Liu et al. [35] take a focused survey on continual learning for vision–language models, developing the idea that adapting large pretrained vision–language models to non-stationary, open-ended streams brings failure patterns in addition to failure patterns traditionally associated with unimodal continual learning approaches. The paper identifies three types of vision-languagely specific degradation, namely a cross-modal feature drift, shared module interference, and zero-shot capability erosion. Building on these issues, the authors suggest a method to organize existing methods, as a problem-driven taxonomy, in three broad families of solutions: multimodal replay approaches; cross-modal regularization approaches; and parameter-efficient adaptation approaches, poems and adaptations. Beyond the categorization of the various methods, the survey unifies the ways in which this area is assessed, summarizing some common metrics on continual learning as well as vision-language relevant metrics, e.g., zero-shot transfer and zero-shot degradation, pointing out retrieval specific metrics in televised retrieval tasks. As well, it identifies the benchmark landscape and maps it into repurposed unimodal, adapted multimodal, and native visionyntual driving and endogenous vision-landmark landscapes with a focus on the problem that common splits may bury alignment break-downs and case for stronger benchmarks efforts that probe across-modal alignment and compositional generalization in continual scenarios.

Shi et al. [30] Sentinel Surveys conducted Motivation Data Continual Learning Ideas Adapted to Large Language Models (arXiv, July 11 2023), Motivated by the need to keep statically pre-trained models effective under changing data distributions with continuous growth, improving incomplete generality to the world, and preventing a devastating forgetfulness from catastrophic forgetting of values domain(s) of earlier training, we survey approaches that focus on an easy to implement strategy. Two complementary notions of continuity highlighted by them are vertical continuity, which describes an endless adaptation from general capabilities to more specialized ones, and horizontal continuity, which represents an endless adaptation in competing with and adapting to time and domains. Building upon that framing a, the paper structures the problem of modern continual learning for large language models into three types of learning, termed continual pre-training, domain adaptive pre-training and continual fine-tuning, to clarify the distinction between choices and challenges with respect to conditional continual adaptations on study choices on where conditional continual adaptations are placed into the language model pipeline. In addition to discussion oriented to the method, the survey also consolidates evaluation protocols and available data sources for studying continual learning with large language models, underlining the role of having consistent experimental setups for interpreting continual learning progress across different studies. Finally, the authors present wider looking open questions and express how constant pre-training, adaption, and fine-tuning of large language models is still comparatively under explored.

Gao et al. [29] propose Consistent Prompting, a rehearsal-free continual learning approach based on a frozen pre-trained Vision Transformer and is inspired by the fact that current prompt-based continual learners can exhibit inconsistent behavior in the training and test stages. The paper states two types of concrete inconsistency: classifier inconsistency and prompt inconsistency. To fix the issue of classifier inconsistency, the method describes a classifier consistency learning, where the current task prompt is exposed to every classifier seen during training, and a regularization mechanism is introduced, which discourages older task classifiers from stemming higher logits than the current task classifier by a margin. To address the problem of prompt mismatch, for robustness improvement, another technique is also added to address prompt consistency learning, i.e., prompt in-service: the current task classifier is optimized when training classification, and random prompts are drawn from the pool, prompting the classifier to be robust when the prompt is not well selected at the test time. Because in random prompt sampling it is possible to under-train the current task prompt, the authors propose an auxiliary classifier to help supervise the current prompt better. In addition they propose a multi-key prompt selection strategy because there is a potential for multiple keys to represent each task, and report on improved prompt selection accuracy compared to single key mappings. The approach is evaluated in class-incremental learning without task identity at test time on four benchmark datasets with different metrics including last accuracy, average accuracy and average forgetting, and from the experiments, the improvements from using than other prompt-based rehearsal-free baselines are reported. The authors also note that the accuracy of prompt selection is relatively low, suggesting that Augustine could be made further gains by improving prompt selection.

Yu et al. [32] provide a comprehensive survey that contextualizes multimodal continual learning as a unique and growing research area of continual learning with a focus on the fact that we can easily overlook modality-specific failure modes when employing unimodal strategies on multimodal data. The paper serves as a motivating force for multimodal continual learning: While static training leads to successful results in practice, real trains can have data that changes over time, we argue that multimodal environments further increase the challenges of continual learning since models have to not only maintain task knowledge but also maintain cross-modal alignment and fusion behavior. A great contribution of it is that it articulates four challenges: modality imbalance, complex modality interaction, high computational costs, degradation of a zero-shot capability pre-trained when continual fine-tuning disrupts multimodal foundations. To organize the field, the survey has defined multimodal continual learning environments and situations and provided a taxonomy of methodologies that fall into regularization-based, architecture-based, replay-based and prompt-based methods and its relation to the practical constraints such as the growth of parameters. Beyond methods, it gives a summary of evaluation metrics used for continual learning, but stressing multimodal-related concepts such as zero-shot transfer and dangers of negative forward transfer. Finally, it gives a summary of available datasets and multimodal benchmarks and ends with some forward-looking points involving more modality coverage, more interaction modeling, better parameter-efficient solutions, and maintenance of pre-trained capabilities.

Dequino et al. [28] Rehearsal-based continual learning for spiking neural networks in resource-constrained edge settings, In: In: Proceedings of the 24th International Conference on World Wide Web. Association for Computing Machinery. (2020), pp. 1085-1100. In the original paper on their results, the authors stated that though replay is often one of the most accurate strategies for continual learning in conventional deep networks, it is difficult to use in spiking networks because storing and replaying past data can be prohibitively memory-intensive on small devices. To overcome this deficit, the paper presents a memory-efficient latent replay implementation of spiking-network continual learning (where the model is divided into two parts: the front part is frozen, and the back part is trainable) for which rehearsal is done with stored latent activations generated from a selected intermediate layer. Due to the fact that spiking network latents are time sequences of spikes in many timesteps, the authors recognize the storage of replays as a growing bottleneck upon learning more classes in sequence. They propose a lossy compression of the sequence of spikes in the time-domain, where they chunk the sequences and compress them to fit in memory, followed by a decompression operation which is expected to maintain the time scale of the network during the rehearsal. The technique is evaluated on the Spiking Heidelberg Digits dataset with a recurrent Spiking Neural Network as follows: Artificial neural network is trained with surrogates and backpropagation through time, Also in experiments we choose both sample-incremental and class-incremental setup, and a more challenging multi-class-incremental routine. Across these evaluations, the paper says, latent replay is able to sustain good accuracy while also minimizing forgetting, which also demonstrates that applying the proposed compression offers large memory–accuracy trade-offs and consequently shows how compressed latent replay makes replay style continual learning more feasible for spiking neural networks on the edge.

Overall, the studies reviewed in this chapter indicate that the choice of algorithmic strategy used to perform continual learning- replay, distillation, or parameter-efficient adaptation overexpression continual learning performance based on the assumptions made regarding data access, task structure, and evaluation protocol [21], [22], [25]. At the same time, a certain amount of recent work also emphasizes that more practical limits like e.g. the memory allowable and runtime or latency aspects can determine to a great extent which approaches are feasible outside of controlled benchmarks [15], [19], [34], [37]. These findings taken together provide the impetus to explore continual learning methods both empirically with regard to effectiveness and from the perspective of deployment-oriented trade-offs, laying the context for the methods and design choices presented in the following chapter [27], [33].

Chapter 3

Proposed Methodology

3.1 Design Process

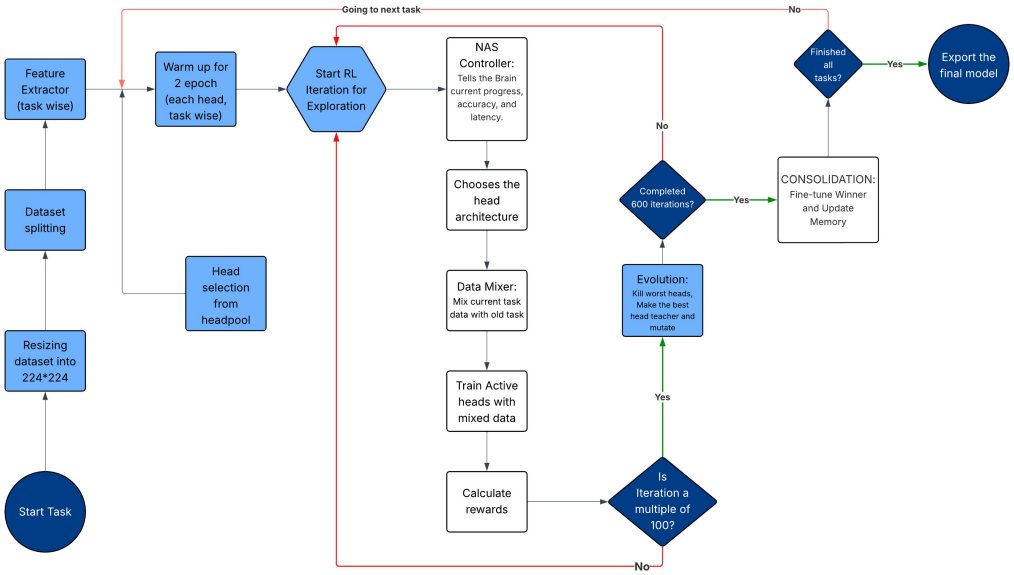


Figure 3.1: Proposed TRACER continual-learning workflow.

In this thesis,we suggest TRACER (Task-Aware Risk-Adaptive Architecture to Continual Edge LeaRning) a latency-aware continual learning architecture to learn class-incremental in Split CIFAR-100 and is verified as well in Split CIFAR-10, a smaller continual stream due to its size. The key problem in this context is that the model should continue to learn new classes in a series of tasks and still retain what it previously learned without forgetting, but it must be useful enough to be used in dealing with edge deployment where inference time constraints are limited. In order to overcome these limitations, the TRACER divides the network into two parts. The first component has the ResNet-18 backbone which is pretrained on ImageNet but kept frozen throughout the tasks to ensure that there are no changes in the features representation and that training takes place and is time-efficient. The second component is a task-adaptive MLP classifier head. Instead of a fixed-head architecture, TRACER keeps a population of candidateheads and iteratively enhances them with enhancements through a course of reinforcement learning combined with

evolutionary mutations. Once the search process is done, we pick the most suitable best head and refine it and proceed to the next task.
The system comes out as a repeatable pipeline in each task. First,CIFAR-100 is divided into ten tasks each containing ten classes and training/validation splits are deterministically constructed to make the experiments reproducible. The images are then processed by the frozen backbone once to give a feature vector with 512 dimensions, and the stored features are used to create a dataset in a Z-space that all the candidate heads can be trained on quickly without repeatedly running the backbone. Short warm-up phase is then used to initializes and ranks seed heads which forms the starting population of main search loop. In the RL-controlled stage, training the controllers is determined by a single controller choosing what head to train and another controller choosing the training hyperparameters.A new variant of heads is formed at a regular time interval by mutation and is then accelerated by using Net2Net transfer of weights. During the search, latency is considered as a hard constraint with the help of a Lagrangian penalty to prevent infeasible models. Finally, when the task is finished, the best head is further refined by the fine-tuning method, the exemplar memory is updated according to the risk-aware allocation plan, and the model is tested at the end of each task in both CIL setting and Task-IL oracle setting.

3.2 Design Model Specification

The section presents the design specification of TRACER (Task-Aware Risk-Adaptive Architecture to Continuous Learning of Edges). TRACER is designed to perform continual learning on edge devices, with the learner required to continually add new classes incrementally and still be reliable on the old classes and maintain viable inference latency.Its architecture is based on a modular concept, a frozen backbone generates stable feature representations, and a searchable lightweight MLP head offers task-adaptive capacity. Risk-awareness with selection (preferring robust validation performance instead of only average performance) and latency-awareness (penalizing architecture that exceeds the edge budget) are some of the design decisions of this model.

3.2.1 Problem Setting and Task Stream

TRACER is tested in the class-incremental learning (CIL) environment over Split CIFAR-100. The dataset consists of 100 classes that are divided into M=10 sequential tasks each with 10 new and non-overlapping classes. The model is trained at task t with the support of only samples of the subset of classes of the current task but should make predictions over all the classes learned until this point during inference. This environment is difficult since learning of new classes can override classes of the old in decision boundaries resulting in catastrophic forgetting.
Let $\mathcal{C}_t$ denote the set of classes introduced at task t. Tasks are disjoint and the seen-class set expands over time:

$$\mathcal{C}_{\leq t} = \bigcup_{i=1}^{t} \mathcal{C}_i, \quad \text{with} \quad \mathcal{C}_i \cap \mathcal{C}_j = \ (i \neq j). \tag{3.1}$$

The training dataset for task t is $D_t = \{(x_i, y_i)\}$ where labels belong only to the current classes ($y_i \in \mathcal{C}_t$). After completing task t, CIL evaluation requires prediction over the full seen set $\mathcal{C}_{\leq t}$ without access to task identity. In addition to CIL, TRACER also reports Task-IL results as an oracle reference, where task identity is provided at inference time and prediction is restricted to $\mathcal{C}_t$. The combination of reporting both protocols helpts in isolating two causes of difficulty: cross-task confusing (CIL) and within-task discriminating (Task-IL). This task-stream to definition correspondingly suits the target deployments environment of TRACER: this model has to continually expand its class knowledge along with tasks and still ensure its accuracy on previous tasks under limited memory and latency conditions.

3.2.2 Backbone Feature Extractor (Frozen ResNet-18)

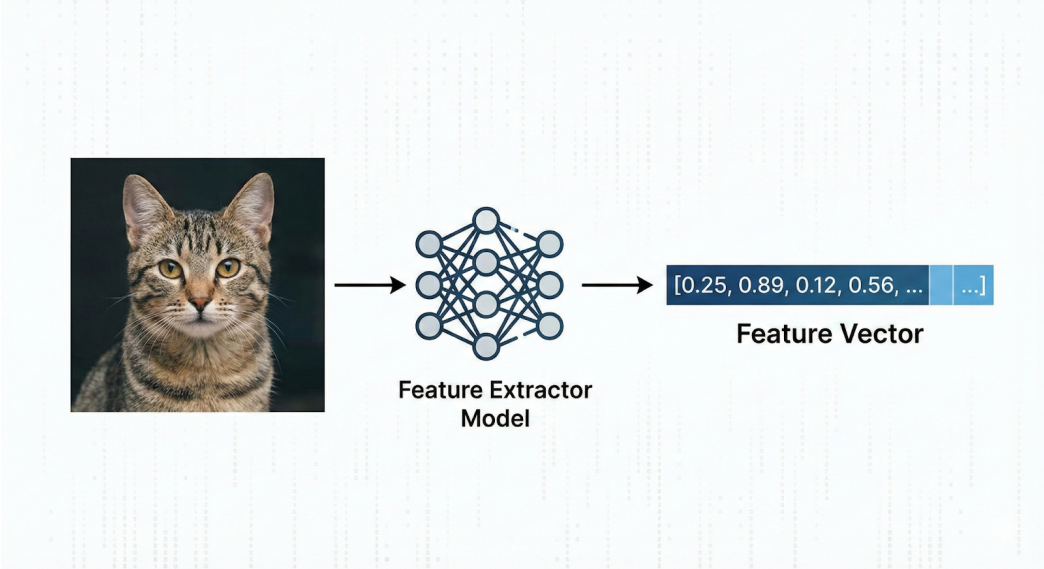


Figure 3.2: Process of frozen backbone feature extraction for the cached "Z-space" dataset.

TRACER applies a ResNet-18 backbone that is allowed to use a fixed model feature extractor but after ImageNet pretraining. The purpose of the backbone is to map every input image into a compressed form which is shared throughout continual learning tasks. The main design decision is that the backbone remains frozen about all tasks, i.e. its parameters do not change in the course of continual learning.This stabilizes the space of representation, minimizes the computation and eliminates the tendency to cause the feature drift that tends to increases forgetting in the event of a continual encoder update.
TRACER rescales images to 224x224 to have the best correspondence with ImageNet-pretrained backbone. Following the convolutional processing, the global average pooling process produces a 512 dimensional feature vector:

$$z = B(x), \quad z \in R^{512}. \tag{3.2}$$

Because the backbone does not change over time, TRACER can compute these features once and reuse them repeatedly during head training and search. Since these features do not change with time, TRACER is able to calculate them once and

reuse them many times in head training and search. This is valuable to efficiency: TRACER trains a lot of candidate classifier heads per task and caching features prevents the unnecessary retransmission of raw images through the backbone on the back-end every time there is a search update. This leads to the fact that the compute-heavy backbone spends minimal time, and the lightweight head can be trained repeatedly in low cost.

3.2.3 Searchable Classifier Head (MLP Family)

Besides frozen backbone aspects, TRACER has a searchable MLP classifier head as its adaptive part. Instead of using a particular fixed classifier architecture throughout a flow of tasks, TRACER has a population of candidate heads and the task searcher seeks a combination of heads that best fits the current task subject to latency constraints. This design is inspired by two facts of the reality of continual edge learning: various tasks can have varying capacity requirements and edge deployment needs to regulate model complexity in order to maintain inference times fast.
Each candidate head takes the backbone feature $z \in R^{512}$ as input and produces logits over the full CIFAR-100 label space:

$$\ell = H(z), \quad \ell \in R^{100}. \tag{3.3}$$

TRACER has head search space that is clearly defined, based on three architecture factors, which we appled in our implementation:

- **Depth:** this is the number of hidden layers, ranging, of course, between shallow (with zero hidden layers of a linear head) and deeper heads with multiple hidden layers.
- **Width:** the hidden units per layer are chosen from a discrete set $\{64, 128, 256, 384, 512\}$, which controls the capacity–latency trade-off.
- **Dropout:** dropout probability is selected from $\{0.0, 0.2, 0.5\}$ and applied within hidden blocks for regularization.

The hidden layers consist of 5 layers, each of which has a standard structure (*Linear* $\rightarrow$ *ReLU* $\rightarrow$ optional *Dropout*), and the last layer is divided into 100 logits. Even though an activity is only performed at a limited number of classes at a specific time, TRACER still inherits a single 100-way output and uses a masking (following subsection) to implement the incremental learning protocol. This maintains a similar head definition across tasks as well as eases evaluation. In general, the searchable head gives TRACER the ability to achieve adjustable capacity: this model can be left lightweight to do easy tasks or add more complex head to heavier tasks or higher-risk class subsets, still within the limits of the latency budget due to the constraints-aware selection process described later.

3.2.4 Logit Masking for CIL and Task-IL

TRACER uses a logit masking technique to make training as well as evaluation dynamic according to continual learning protocols. Since all 100 classes are generated as logits by all the heads, the model should avoid allocating probability to

invalid classes at a particular stage. One way of accomplishing this is logit masking (where invalid logits are masked with a large negative constant prior to softmax application), to make invalid classes get close to zero probability.
In the CIL setting after learning task t, the valid set is $\mathcal{C}_{\leq t}$ (all classes learned so far). In the Task-IL setting for task t, the valid set is $\mathcal{C}_t$ (only the classes of that task). On both these instances, masking allows predicting in the correct label space without modifying the model architecture. This is necessary in the assessment of fairness as it is the process that provides the model to be considered only on those classes that have already seen the model contemplated in the task stream.

3.2.5 Training Objective: Masked Cross-Entropy + Knowledge Distillation

To memorize the desired task but not to forget them, TRACER trains a single candidate head with a joint task consisting of masked cross-entropy and knowledge distillation. The backbone is frozen thus optimization only provides the parameters of the classifier head.
Masked cross-entropy trains to learn current task classes are done in the incremental label space, logits are masked to valid seen-class set and the classification loss is computed, ensuring consistency with CIL protocol.Nonetheless, using cross-entropy alone, has a tendency to steer the model towards the most recent classes and induce the logits of the older classes to drift.
In response to this, TRACER equally practices knowledge distillation through an extension of a frozen teacher head stored at the end of the last task. The teacher offers soft performance goals to previously learned classes, as it paves the way to an influence of the new head to maintain the old decision boundaries despite its adaptation to new data. The use of distillation especially in class-incremental-larging learning makes sense since it regularizes behavior of the model on old classes without necessarily having complete access to old training data.
The overall training objective comes in a concise form as:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_{KD}\,\mathcal{L}_{KD}. \tag{3.4}$$

Here, λ_{KD} controls the stability–plasticity balance: larger values prioritize retention of old knowledge, while smaller values prioritize faster adaptation to the new task. Through this loss design, TRACER is able to acquire new classes, but it suppresses performance decreases on previously performed tasks.

3.2.6 Latency Measurement and Constraint

As TRACER is edge-deployment focused, efficiency is treated as a core constraint rather than an afterthought. For each candidate head, TRACER measures inference latency using the composed model $f(x) = H(B(x))$ under a consistent evaluation setup (fixed input size and batch configuration).Latency is used to summarize the worst-case responsiveness; to do this, the 95 th percentile ($p95$) of measured inference time per image is used. This tail-focused metric is more suited to edge applications than average latency since systems of deployment should not lose responsiveness to variability.

TRACER establishes an authorised minimum threshold of latency execution by specifying a target figure with a small tolerance difference to engage measurement error. Certain heads of candidates are deemed viable when their $p95$ latency is not more than this value. To make it a part of model selection TRACER calculates a scalar that measures non-negativity of the violation:

$$g = \max(0,\ LAT_{p95} - LAT_{allowed}). \tag{3.5}$$

This violation is used in search to sanction too slow architectures. Consequently, the TRACER is a systematic enhancement of heads with good validation performance at the same time being able to deploy within the edge latency budget.

3.2.7 Reward Function

TRACER is based on a reward-oriented selection process that directs search process across candidate heads and training settings. The reward will be geared towards the key demands of continual edge learning: high performance on the task at hand, low rates of forgetting previous tasks and observing the latency constraint. TRACER tests a candidate head based on validation data after accommodating a brief update window, and provides a reward. The first element of this reward indicates justification performance towards the task at hand. This score would be calculable in a risk-adaptative environment of TRACER whereby a tail-centered estimate (CVaR style) can be applied to prefer candidates with better scores both on average and those who are consistent on challenging batches or classes. Another approach used to prevent forgetting is that TRACER additionally adds a stability signal which measures the consistency of performance on previously learned classes, usually computed based on replay memory validation or cached old-task validation capabilities. This is so that they search without choosing new heads that attain some short-term benefits at the cost of old knowledge. The latency penalty is incorporated via a form of Lagrangian penalty: the candidates violating the latency threshold are penalised with a negative score based on the extent of violation. The strength of this penalty is determined by a dual variable that is gradually modified in the course of training to ensure that the consistent violations lead to the strengthening of the constraints. Lastly, TRACER has a small entropy regularization factor that prevents over-confident prediction early when learning and stabilizes the dynamics on incremental training.
The reward is represented in a small format:

$$R = (1 - \gamma)\, acc_{now} + \gamma\, acc_{old} - u \cdot g - \beta H, \tag{3.6}$$

where acc_{now} measures current-task validation performance, acc_{old} measures retention on old classes, g is the latency violation from Section 3.2.6, u is the adaptive penalty weight, and H denotes predictive entropy. This reward design allows TRACER to prefer candidate heads that are accurate, stable, and feasible for edge deployment.

3.3 Data Collection

The TRACER exploits CIFAR-100 [2] and CIFAR-10 in appraising continual learning with an edge constraint. Both data sets consist of 32x32 scale RGB photos. There are 100 classes (CIFAR-100) 50,000 train and 10,000 test in CIFAR-100, whereas there are 10 classes (CIFAR-10) 50,000 train and 10,000 test in CIFAR-10. These benchmarks enable the use of TRACER both on a large class-expansion setting (CIFAR-100) and on a smaller and controlled setting of incremental setting (CIFAR-10).

3.3.1 Dataset and Task Splits

To ensure continual learning, each datasets are transformed into task stream through splitting the label space into class subsets. Split CIFAR-100: It is created with M=1 consecutive M=tasks, and 10 non-overlapping classes are introduced with each task, and thus the number of valid classes grows with each task. The CIFAR-10 is set into a smaller incremental stream, divided into 2 classes per task, with M=5 allowing the study of the stability and efficiency of TRACER in case of a more simple label space.
Task construction and task splitting is deterministic to achieve reproducibility. Each of the classes is then shuffled with a fixed seed and separated into training and validation subsets (e.g., 400/100 per class, in CIFAR-100), and the official test is left untouched and used to evaluate the models. Once every task has been completed TRACER examines test sets of all tasks that have been learned up to this point: CIL accuracy is calculated on the union of seen classes and Task-IL accuracy is calculated with oracle task identity being restricted to the classes of the current task.

3.3.2 Preprocessing

Since TRACER takes an ImageNet-pretrained ResNet-18 backbone, all CIFAR images are preprocessed to fit the desired format of the backbone. Despite CIFAR-10/100 images have a size of 32×32, it is rescaled to 224×224 and sent to frozen backbone. The use of separate training and evaluation transforms ensures that TRACER generalizes better and does not compromise test time behavior in any way.
To train, all images are processed with a resize of 224×224, and then random horizontal flipping done as an augmentation technique that is lightweight and can be used on CIFAR object categories. Lastly, images are normalised by the standard ImageNet mean and standard deviation so that the backbone statistics are compatible with the pretrained statistics.
To be evaluated (validated and tested), the transform is deterministic: all images are resized to 224×224 and normalized by the same ImageNet mean/std without any random transformation. This is to ensure that reported metrics are based on model performance, and not stochastic preprocessing effects.

3.3.3 Feature Caching ("Z-space Dataset")

Given that the ResNet-18 backbone of TRACER is frozen, its output characteristics do not change with training steps within a task. TRACER takes advantage of this by caching backbone embeddings at once and then training and searching their entire classifier-heads in this space. This greatly helps in reducing computation and the re-calculation of numerous candidate heads is made practical.

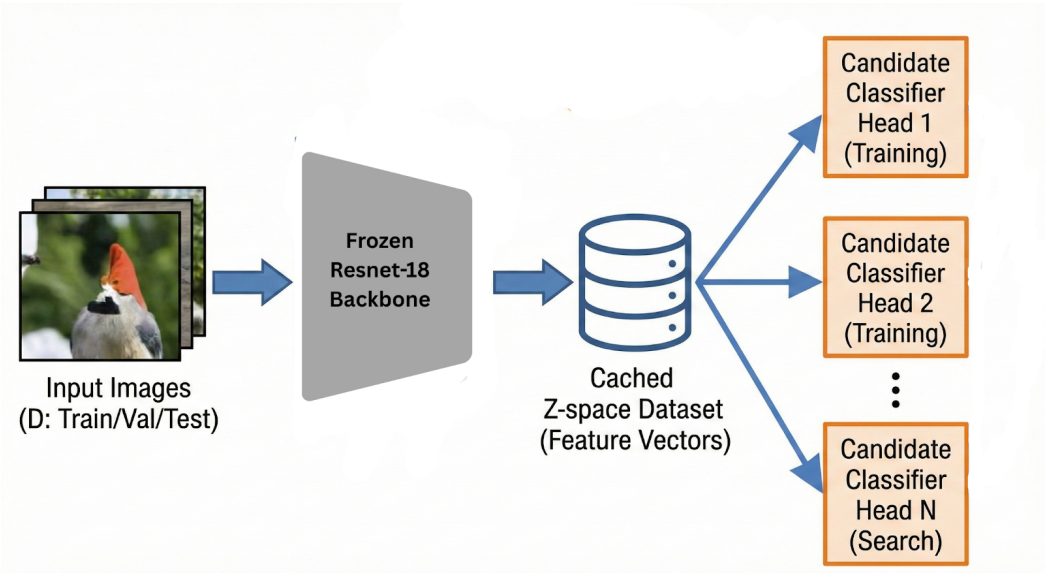


Figure 3.3: Frozen backbone feature extraction and cached Z-space dataset in TRACER.

In practice, in both split (train/validation/test), a single forward of images using the backbone is used to store feature vectors. TRACER then constructs a cached Z-space dataset:

$$D^Z = \{(z_i, y_i, idx_i)\}, \quad \text{where } z_i = B(x_i). \tag{3.7}$$

Here, z_i is the 512-dimensional backbone embedding, y_i is the class label, and idx_i is the sample identifier used for reproducible bookkeeping across tasks.
Once the architecture has been cached, the architecture search and continual training loop is run on $(z, y)(z, y)(z, y)$ pairs, that is, candidate MLP heads can be trained and trained comparisons done without iterating ResNet-18. Consequently, TRACER can analyze high densities of head confugurations in a very efficient way whilst maintaining the backbone constant and stable during the constant learning stream.

3.4 Implementation of Selected Design

3.4.1 Reproducibility Settings

TRACER experiments have also been rendered reproducible through fixes of random seeds in Python, NumPy, and PyTorch and sharing the same seed throughout construction of tasks, by randomly shuffling datasets, and generally by fixing the sampling of exemplars. To promote the use of deterministic execution, it is possible

to enable PyTorch deterministic algorithms and set cuDNN to act deterministically. The workers of DataLoader are additionally seeded in a deterministic manner to ensure that one can never repeat a given batch of information without the order remaining unchanged with each run, as well as that any stochastic preprocessing can also remain consistent.

3.4.2 Training Hyperparameters

TRACER trains only the classifier head while keeping the ResNet-18 backbone frozen. Unless otherwise stated, head optimization uses AdamW with weight decay 1×10^{-4}, batch size 64, and gradient clipping at 1.0. Every task starts with a short stage of warm up of the seed heads, then a stage of search controlled by RL (600 iterations, 8 steps of gradient update in each selection). The learning rate and distillation settings are selected at random by a small predetermined grid, and the replay batches intermix samples (although balance) between the current task and examples at the exemplar memory. Once a search has been done, fine-tuning that is of short duration (equalized fine-tuning, consolidation, and balanced fine-tuning) is applied to the chosen head so that it is more stable and less recency biased.

3.4.3 Warm-up Stage (Per Task)

TRACER then starts every task with a warm-up which will quickly assess seed heads to develop a good starting population to search. First, the cached feature splits for the current task are prepared, namely D^Z_{train} and D^Z_{val}, so that all warm-up training runs directly in Z-space (with the backbone excluded from the loop).
TRACER next performs training with all seeds of partner MLP heads of a fixed number of update steps: namely, 200 steps per head in Task 1, and 50 steps per head in all subsequent tasks. Using the same latency-aware reward used in the main search, both of the heads are scored on the validation split. This reward ranks heads, and the 4 best maintained to start the population.
Warm-up relies on constant replay mixture in order to make starting conditions uniform across tasks: examples of current tasks will always be represented, players of mini-batches are about 35 percent chosen out of the exemplar memory. This warm-up phase gives the stable and competitive set of initial heads with keeping computation small until the RL controlled evolutionary search takes place.

3.4.4 RL Search Loop (Main NAS + Continual Learning Component)

Along with the RL-based selection, TRACER deploys an evolutionary refinement in the search to keep on increasing the improvement of the head population. Each 100 RL evaluations, the algorithm picks the parent head that has the highest average reward in the recent evaluations. TRACER also studies the validation performance and finds the worst-performing (bottom-K) classes to be made task-conscious, as those are the most risky areas that may need extra capacity.
After warm-up, TRACER runs an RL-controlled head search for `RL_ITERS=600` iterations. In each iteration, the selected head is trained for `TRAIN_STEPS_PER_PICK=8`

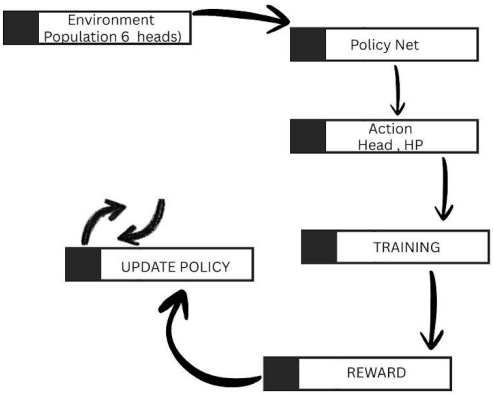


Figure 3.4: RL-based head search loop in TRACER.

gradient steps on mixed replay batches, allowing many candidate architectures to be evaluated efficiently within one task.

Two controllers drive this process. The head controller selects which head to update from `MAX_HEADS=16` slots, where only the first N active heads are selectable (active mask). A latency feasibility mask excludes heads with $p95 > LAT_{allowed}$; if no head is feasible, selection falls back to all active heads. Exploration is maintained using a temperature schedule $7.0 \rightarrow 0.9$ and an epsilon floor of 0.10. To stabilize early behavior, the first selections follow a round-robin bootstrap (about $4\times$ the population size) before the controller fully exploits reward signals.

The hyperparameter controller selects one training configuration from a grid of 48 combinations: KD weight $\{0.45, 0.60, 0.70\}$, KD temperature $\{2.0, 3.0\}$, learning rate $\{7 \times 10^{-4}, 5 \times 10^{-4}\}$, and replay ratio $\{(0.65, 0.00, 0.35), (0.60, 0.00, 0.40)\}$ for (current, error-buffer, exemplar), with error-buffer fixed at 0. Both controllers use the same 21-D state vector (search progress, reward/latency/memory statistics, and last-head one-hot). Policies are updated with REINFORCE, using an EMA baseline (0.9) and entropy regularization (0.02) to reduce variance and preserve exploration.

3.4.5 Evolutionary Operations (Mutation + Net2Net)

In the chosen parent, TRACER produces 2 heads of an offspring on each cycle with architecture mutations. These mutations consist of widening/narrowing hidden layers with fixed step size 64 units of width, adding/deleting hidden layer (subject to 4 maximum hidden layers), and dropout adjustment. Following the mutation, children will be initiated by transferring the weight matrices between parents through Net2Net style weight transfer, where applicable overlapping parts of the weight matrices of the parent weight matrix will be transferred to the offspring weight matrix. This also can be greatly faster than training through random initialization as well as enable the search process to explore structural change without reversing all of the previously known information.

TRACER manages the size of the population by keeping it constant and eliminating

evidence when a candidate is demonstrated to be dead following every evolutionary cycle. Once offspring are inserted and evaluated, the population is reduced back to `POP_SIZE=6` by keeping the highest-reward heads. Finally, TRACER supports risk-driven mutation, in which a mutation operator is biased based on widening of layers, and enhanced depth on risky classes recognized, prompting the search to spend increased diligence on the tougher or failure-biased areas of a task development.

3.4.6 Equalized Fine-Tuning, Consolidation, and Balanced Fine-Tuning

Once the RL search is complete, TRACER is used to optimize the search by performing a post-search refinement pipeline in order to produce a stable champion head and decrease the effect of the latest classes. To optimize the top-3 heads of the search stage, Equalized Fine-Tuning (EQ) will be first performed: the heads are optimized over 100 update steps by applying a constant hold back replay mixture (60% current-task samples, 40% exemplar replay). This is a controlled fine-tuning mechanism that makes the final option not dependent on dissimilar training histories in search. The heads are again checked on validation reward and the head that has performed best is chosen.
Then the selected head is fixed under consolidation training of 300 steps once more with 0.60/0.40 mixed batches. The phase enhances the decision limit of the chosen head of the new task as well as by repetition the old-class knowledge through exemplar.
Lastly, TRACER retrains the model by 150 steps to do Balanced Fine-Tuning (BFT) with exemplar-centered batches, to resolve recency bias. In BFT, TRACER uses the teacher of the last task as knowledge distillation on older class output, helping in retaining old logits and to tune the classifier using a more balanced replay distribution.

3.4.7 Risk-Aware Exemplar Allocation

TRACER also updates an exemplar memory with the fixed budget of 2000 samples after every task. The allocation is risk-conscious: example classes that are not accurately modeled by the model are allocated more, hence giving replay better results with a limited memory.
For every seen class $c \in \mathcal{C}_{\leq t}$, TRACER estimates robustness using a CVaR-style tail accuracy on the validation set, denoted $Acc_{CVaR}(c)$. Risk is defined as:

$$risk(c) = 1 - Acc_{CVaR}(c). \tag{3.8}$$

These risk values are converted into per-class exemplar quotas k_c by proportional allocation while enforcing a strict budget:

$$\sum_{c \in \mathcal{C}_{\leq t}} k_c = 2000. \tag{3.9}$$

Since quotas are to be whole numbers, TRACER operates on a largest-reminder scheme to ensure that the total has to be 2000.
Class-wise updating of memory then follows: older exemplars are truncated when a class reaches its old quota and classes which are undersized are superset by random

sampling of the current-task training features D^{Z}_{train}. This generates a budget-precise replay buffer, which automatically focuses on storage on riskier classes.

3.4.8 Evaluation Protocols and Metrics

TRACER is tested on all tasks completed to date after every task (t) to measure the learning progress as well as forgetting. Evaluation of the split (test) of each task is done and the performances of the tasks are reported using two standard continual learning conditions.

Class-Incremental Learning (CIL), does not have task identity at inference time. Therefore, during the consideration of after task (t), predictions are drawn regarding the combination of all the observed classes, ($\mathcal{C}_{\leq t}$) using logit masking. TRACER records per-task accuracy vector of the tasks (1) to (t), as well as the cumulative accuracy of task (t), which captures the overall performance data as the class space increases.

Task-Incremental Learning (Task-IL) makes an inference time (oracle) assumption of task identity known at inference time. Each task is evaluated with logits restricted to its own class set ($\mathcal{C}_i$). TRACER reports Task-IL per-task accuracies and mean-to-date Task-IL accuracy as an upper-bound estimate which separates within-task discrimination performance.

TRACER measures catastrophic forgetting by working out the per-task forgetting. Where Let $A_{t,i}$ refers to the accuracy of task (i) at post learning level of task (t). Forgetting in task (i): is the decay of the best past performance in task (i) to its final performance following learning the last task T:

$$F_i = \max_t A_{t,i} - A_{T,i}. \tag{3.10}$$

This measure characterizes the deterioration of performance on prior tasks with the acquisition of subsequent tasks, and is a complement to the aggregate accuracy, but the metric actually measures stability in the long run.

Algorithm 1: TRACER : Latency-aware continual head search

Input: Tasks $\{T_t\}_{t=1}^{M}$; frozen backbone B; head space $\mathcal{H}$;
exemplar budget B_{total}; latency target LAT_{target}; slack δ
Output: Champion heads $\{H_t^\star\}_{t=1}^{M}$
$LAT_{\text{allowed}} \leftarrow LAT_{\text{target}}(1+\delta)$
$\mathcal{M} \leftarrow \emptyset$
$u \leftarrow u_0$
for $t \leftarrow 1$ **to** M **do**
  Build deterministic splits for T_t
  Cache features $Z_t \leftarrow \{(B(x), y)\}$
  Initialize population $\mathcal{P} \leftarrow$ SeedHeads($\mathcal{H}$); warm-up; keep top heads
  if $t > 1$ **then**
    $H^{teacher} \leftarrow H_{t-1}^\star$
  else
    $H^{teacher} \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** I **do**
    Select $H \in \mathcal{P}$ and hyperparams a (RL controllers)
    Train H on mixed batches from Z_t and $\mathcal{M}$ using
     $\mathcal{L}_{CE} + \lambda_{KD}\mathcal{L}_{KD}(H^{teacher})$
    Measure acc_{now}, acc_{old}; measure LAT_{p95}
    $g \leftarrow \max(0, LAT_{p95} - LAT_{\text{allowed}})$
    $R \leftarrow (1-\gamma)acc_{now} + \gamma acc_{old} - u \cdot g$
    $u \leftarrow \text{clip}(u + \eta g)$; update RL policies with reward R
    if $i \bmod E = 0$ **then** `// every E steps`
      Mutate best head + Net2Net transfer; prune $\mathcal{P}$
  $H_t^\star \leftarrow$ best head in $\mathcal{P}$; consolidate + balanced fine-tune on $\mathcal{M}$
  Update memory $\mathcal{M}$ with risk-aware budget-exact allocation ($\sum k_c = B_{\text{total}}$)
  Evaluate CIL on $\mathcal{C}_{\le t}$ and Task-IL on $\mathcal{C}_t$

Chapter 4

Result Analysis

This chapter analyzes TRACER in the state of unending learning and edge-driven deployment restrictions. Split CIFAR-100 (10 tasks) is reported to be the primary stream and Split CIFAR-10 (5 tasks) is reported to be a compact stream on analysis of latency constrained behavior. We present the results of performance under the class-incremental, and task-incremental learning (CIL/Task-IL), and we also discuss the forgetting, latency of performance (p50/p95), constraint satisfaction, and the utilized head architectures.

4.1 Performance Evaluation

4.1.1 Split CIFAR-100 (10 tasks)

Mean-to-date accuracy (CIL vs Task-IL)

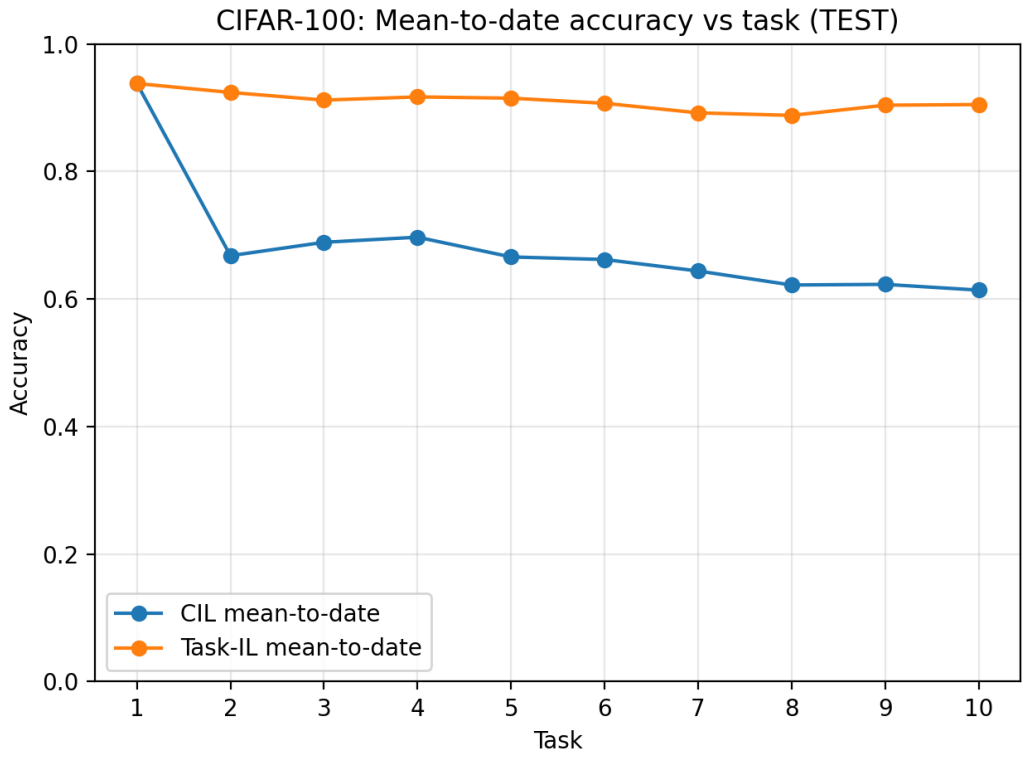


Figure 4.1: Split CIFAR-100: mean-to-date accuracy vs. task (CIL vs. Task-IL).

Figure 4.1 provides results of mean-to-date accuracy on the 10-task stream. Mean-to-date is an average of all the tasks that have been learned to date but calculated after completion of a specific task. TRACER also obtains a final cumulative mean accuracy of 0.6136 under CIL. At the beginning of the work, the curve is largest and becomes smaller with each other course that is offered. Such a drop is foreseeable since CIL expects the classifier to partition a successively larger label space, and also allows the classifier to continue to perform on former tasks. In Task-IL, performance is still significantly better throughout the stream since the identity of the task is known during inference thus it is not required to compete during cross-task classification. Such a distinction between Task-IL and CIL is a conventional demonstration that the overpowering challenge of CIL is cross-task confusion and interference, as opposed to the impossibility of learning one task at the same time.

Per-task performance summary (CIL and Task-IL)

To supplement the trend plot, the final results of training per-task performance are summarized by means of per-task accuracy plots.

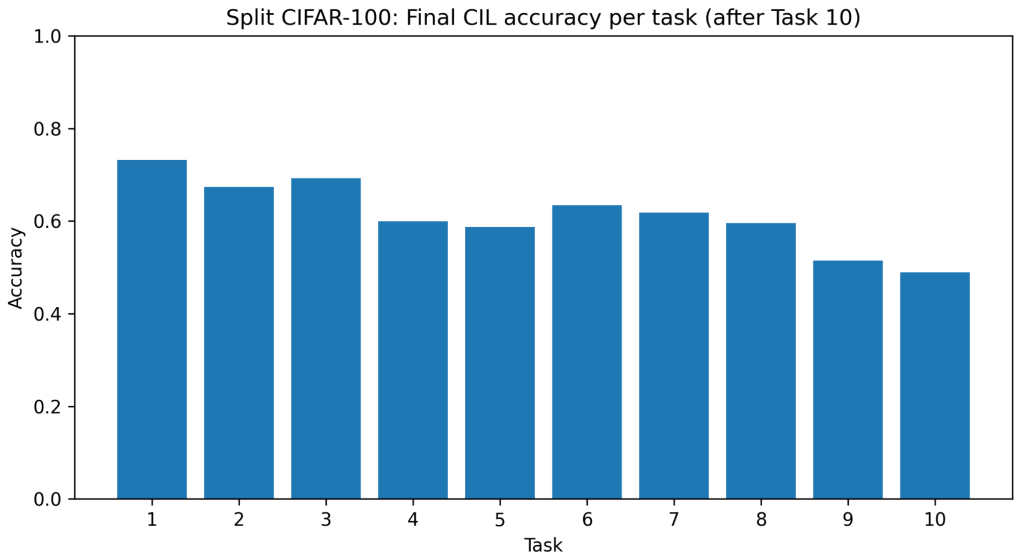


Figure 4.2: Split CIFAR-100: final CIL accuracy per task (after Task 10).

Figure 4.2 represents a summary of the final CIL accuracy of every task upon completion of the stream. The findings indicate that there are better performances in later tasks and relatively poor performances in earlier tasks, which is in line with constant learning that involves more interference with tasks at earlier times.

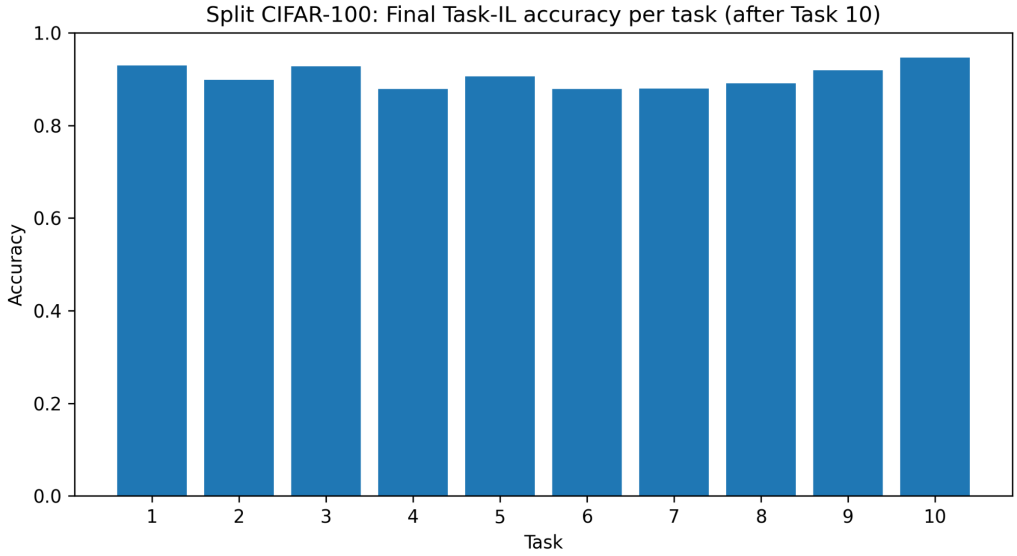


Figure 4.3: Split CIFAR-100: Task-IL accuracy per task (after Task 10).

Figure 4.3 lists the Task-IL per-task accuracy of which the identity of the task is known upon inference. Under such an environment, TRACER performance is characterized by high accuracy per task.
In the 10 tasks, the stable results in the final Task-IL per-task accuracies are:

$$[0.929, 0.898, 0.928, 0.879, 0.906, 0.879, 0.880, 0.891, 0.919, 0.946],$$

with a final Task-IL mean of 0.9055. The fact that concept separations between Task-IL and CIL are large is not surprising and reflects the fact that a large percentage of the CIL drop cannot be attributed to the inability to learn each task in isolation, as opposed to the cross-task class confusion.

Forgetting (CIFAR-100)

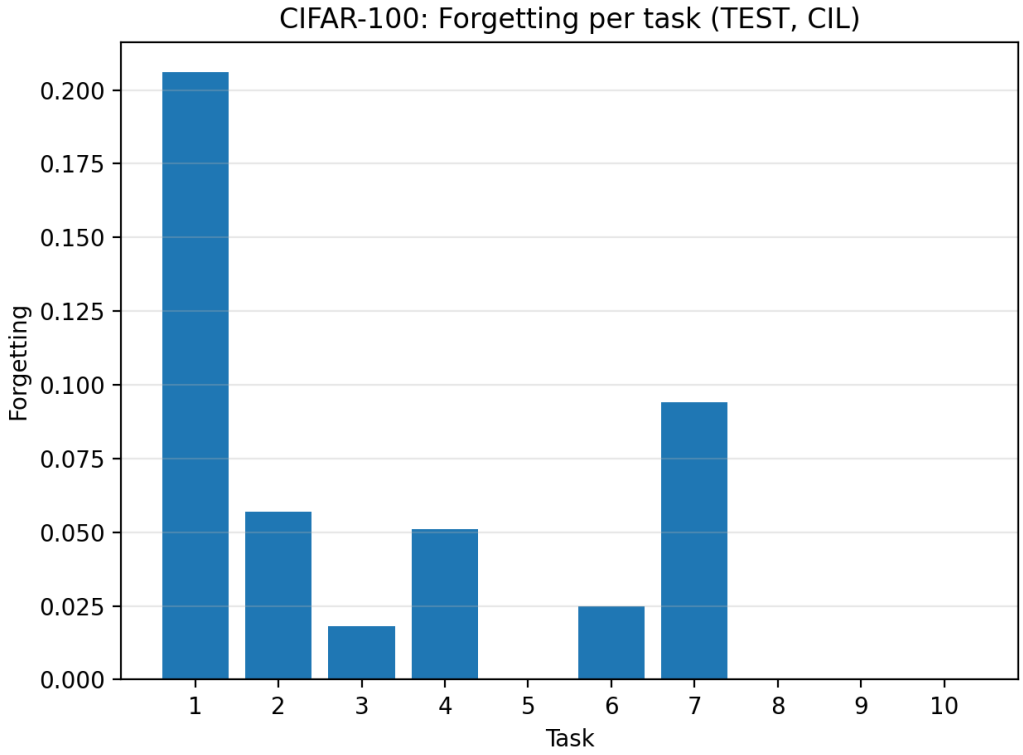


Figure 4.4: Split CIFAR-100: forgetting per task (CIL).

Figure 4.4 gives the forgetting under CIL. The peaks in forgetting are greater in early tasks, and lesser in late tasks, which is common due to the greater subsequent updating of early tasks. The values of forgetting are observed as follows:

$$[0.206, 0.057, 0.018, 0.051, 0.000, 0.025, 0.094, 0.000, 0.000, 0.000].$$

This suggests that TRACER restricts extreme forgetting in tasks initially; and moderate forgetting in the initial tasks during full CIL assessment.

4.1.2 Split CIFAR-10 (5 tasks): Relaxed vs Tight

Here we discuss the Split CIFAR-10 dataset using two latency regimes, slack and tight constraints. These findings demonstrate the presence of class incremental learning (CIL) as well as task incremental learning (Task IL) and evidence of forgetting and latency behaviour.

CIL mean-to-date accuracy (relaxed vs tight)

Relaxed latency regime

Mean-to-date accuracy (CIL vs Task-IL). This figure 4.5 indicates the mean accuracy at the end of every task in Class-Incremental Learning (CIL) and Task-Incremental Learning (TIL). TIL remains at a high level throughout all of the tasks and rnds at 0.970, which provides evidence of strong performance when the identity of the task is known at inference. Conversely, CIL tends to decline as additional classes are introduced, and all in the Task 3 due to the fact that the model is expected to partition the range of all observed classes with no task assignments; before recovering to some degree and reaching a final B CIL mean of 0.8145.

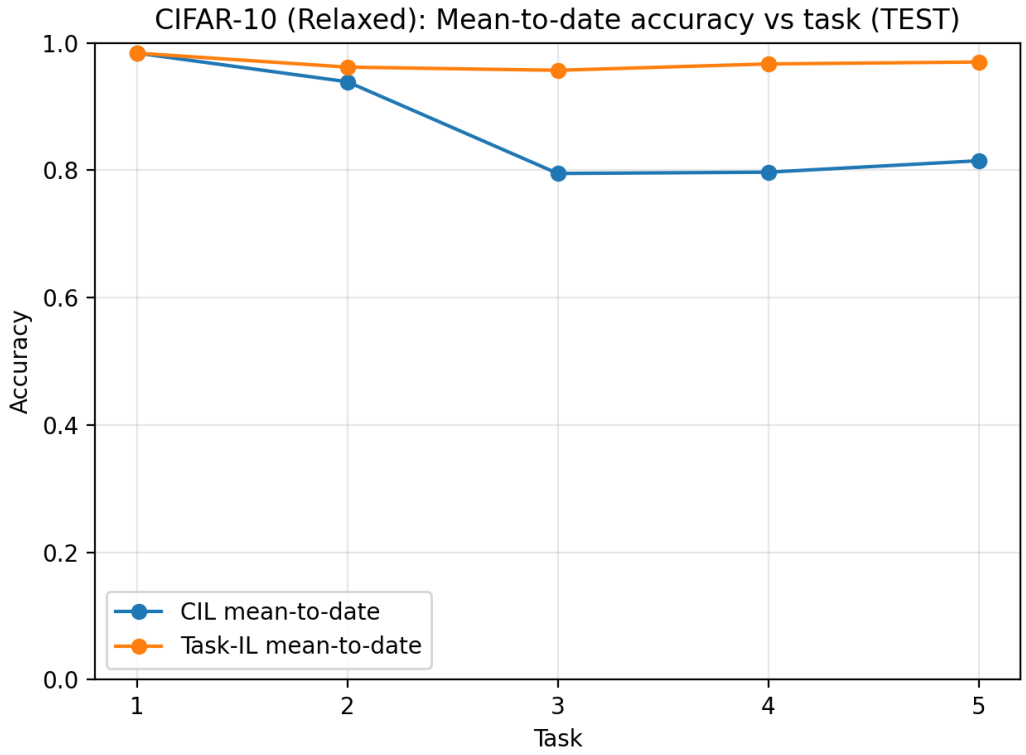


Figure 4.5: Split CIFAR-10 (Relaxed): mean-to-date accuracy vs. task

Forgetting. This figure 4.6 shows the task-wise forgetting in the CIL setting. The forgetting score is overall low and averaged at 0.0341, and focused on the early tasks. In case of later tasks, forgetting is almost negligible which would indicate that the method is stable through time and therefore captures prior learned information well as the training advances.

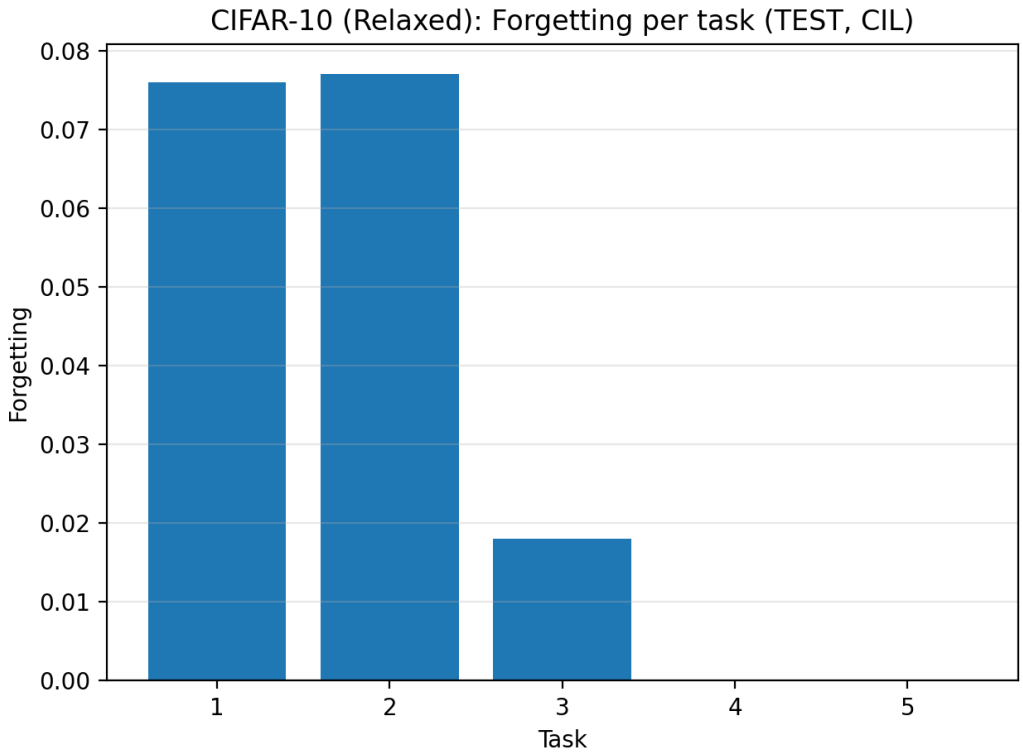


Figure 4.6: Split CIFAR-10 (Relaxed): forgetting per task (CIL)

Tight latency regime

Mean-to-date accuracy (CIL vs Task-IL). Figure 4.7 reports the tight-regime trend. The last CIL mean-to-date value is 0.826 (Task 5), and the last Task-IL is 0.9673. Task-IL is also consistently high on tasks as compared to CIL which is significantly more difficult because of the effects of class-incremental interference and total accumulation of prior classes across the stream.

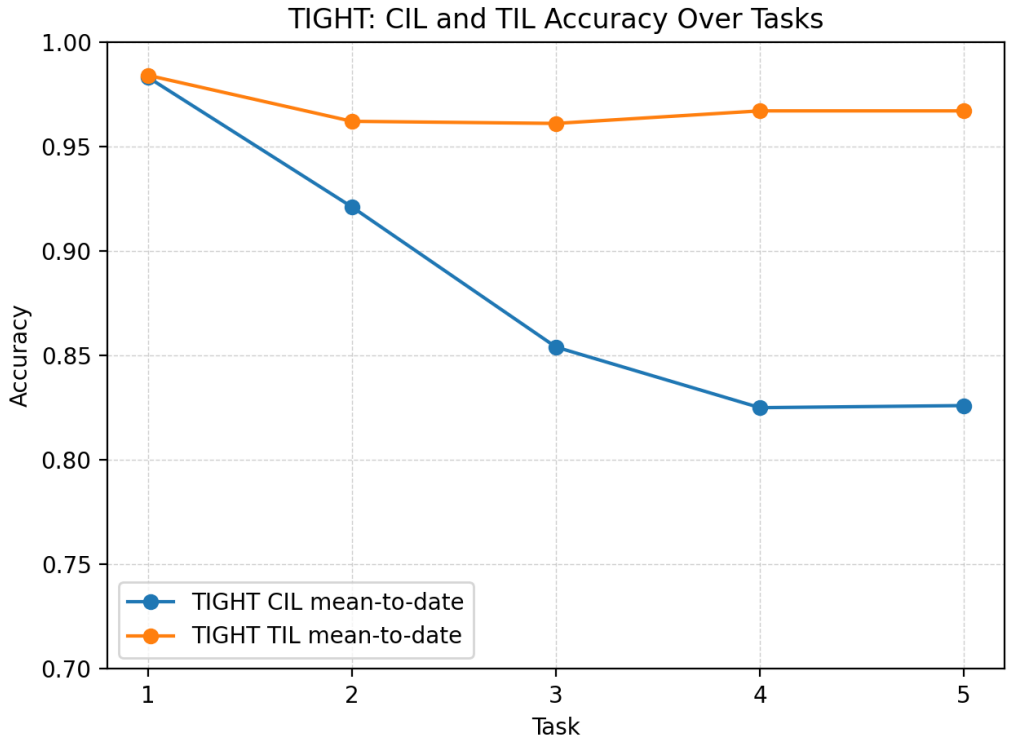


Figure 4.7: Tight CIL and TIL Accuracy Over Tasks

Forgetting. Figure 4.8 is the summary of the forgetting when under tight latency regime. The forgetting increases in earlier tasks but decreases towards later tasks, suggesting that when the exemplar memory is stable with the controller adapted the system will remember later tasks better. The mean forgetting in this tight run is 0.0527 and the per task forgetting:

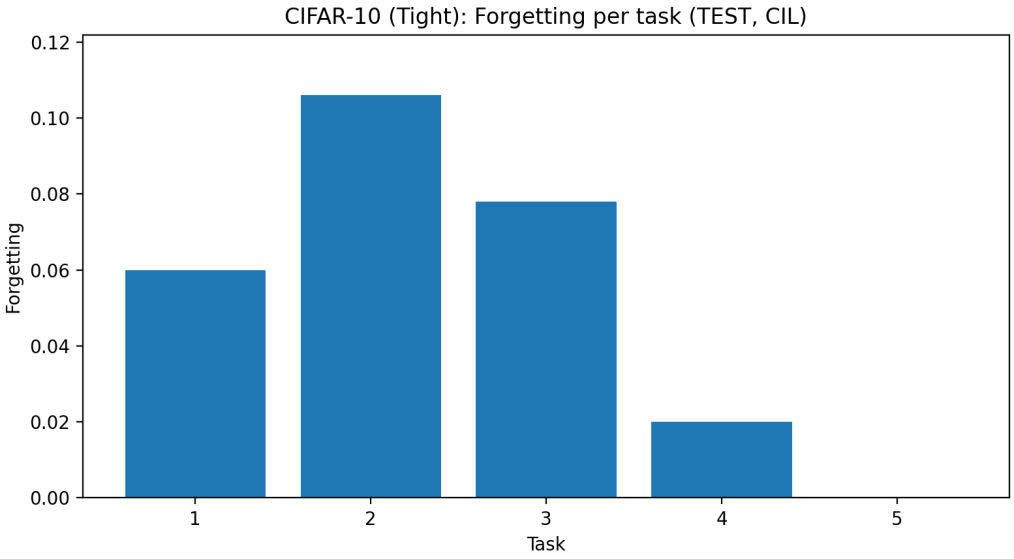


Figure 4.8: CIFAR-10 (Tight): Forgetting per task (TEST, CIL)

4.2 Adaptive Head Behaviour Under Latency Constraints (Split CIFAR-10)

In this section the impact of change in adaptive classifier head with change in latency. How regimes are analyzed and the change of regimes with accuracy and run time is brought out.

4.2.1 Head selection across tasks

Figure 4.9 illustrates the preferred classifier head of each of the tasks in the lax and tight latency regimes. The decisions vary across the stream and, consequently, TRACER does not depend on a single head design, but instead it utilizes the topology of the classifier head tuning in order to trade off between the performance of continual learning and run-time.
Relaxed: H3 → H1 → E4 → H8 → H2
Tight: H4 → H1 → H1 → E4 → H7

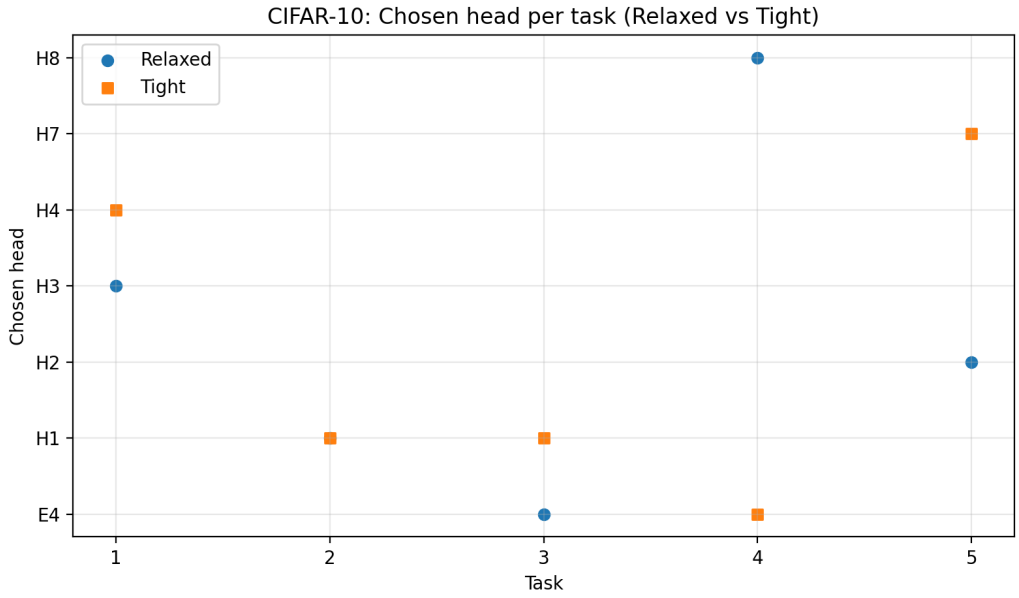


Figure 4.9: CIFAR-10: Chosen head per task (Relaxed vs Tight)

4.2.2 Latency of the selected heads (p95)

Latency of the selected heads (p95). The p95 latency of the selected head per task in each regime gets reported in Figure 4.10. In line with expectations, the tight regime forces the controller to the low latency choices and the optimization is a binding constraint in the tight case (dual variable active across the run), whereas the relaxed regime gives higher latency where that is rewarding.

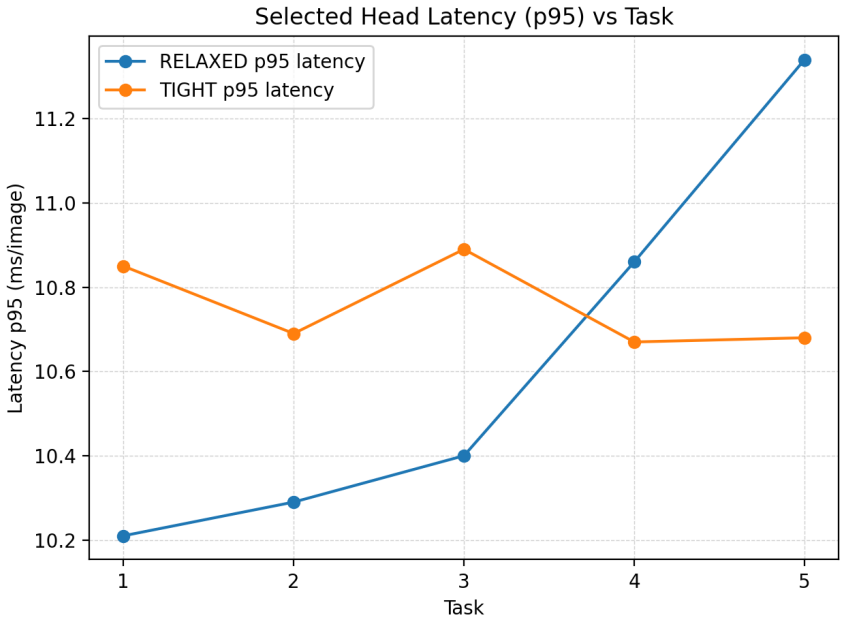


Figure 4.10: Split CIFAR-10: selected-head latency (p95) vs. task.

One such observation is that tight latency does not require that different tasks use the same head, but rather than that, different trade-offs are made by the controller, who picks different heads as the difficulty of a task and its stability vary across the stream.

4.2.3 Mutation impact (Δaccuracy vs Δlatency)

Mutation impact (Δaccuracy vs Δlatency). Figure 4.11 is a visualization of the trade-off provided by mutations in the form of a scatter plot of Δaccuracy vs Δp95 latency between regimes (or between candidate heads whatever you made the delta plot with). The high accuracy (delta accuracy of 0 or more) but shorter runtime (delta latency of 0 or less) is a favorable mutation. Even the points with a minor lower accuracy but observable latency improvement could be helpful in a strict constraint where feasibility is more important than reward.

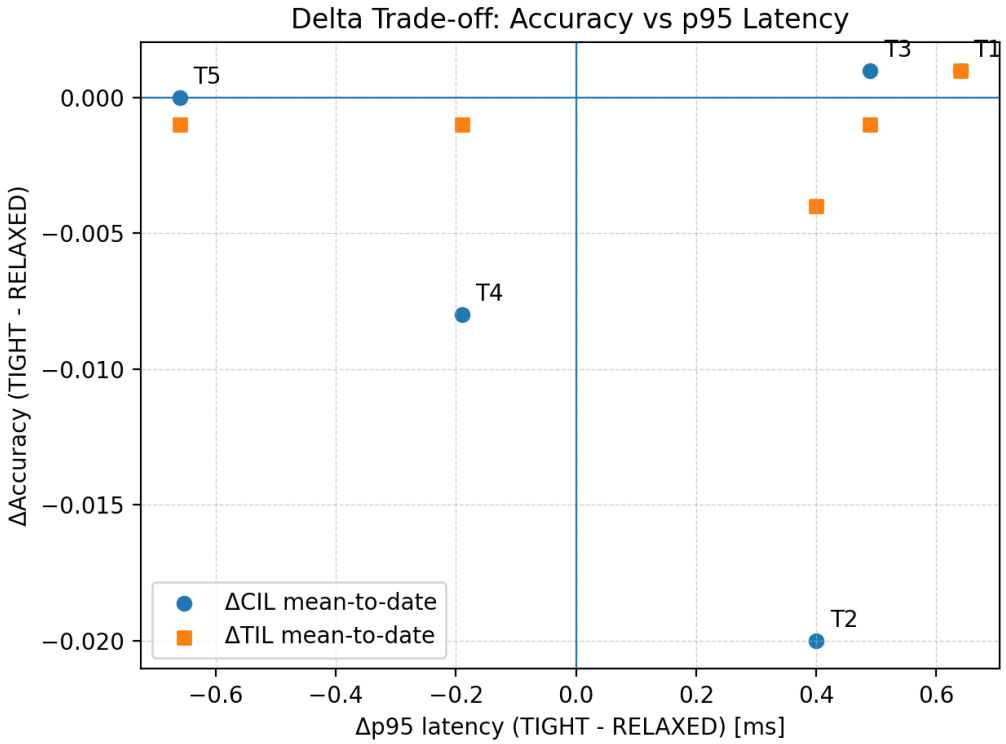


Figure 4.11: Mutation delta plot (Δacc vs. Δp95).

This delta-view helps corroborate the fact that the evolutionary component is productive: it provides candidate heads that search accuracy latency trade-offs in a biased manner as opposed to random alterations in architecture.

4.3 Comparison

4.3.1 Comparison with Baseline Methods For CIFAR-100

Table 4.1: Final performance comparison after Task 10

Method	Final CIL Mean Acc (%)	Final Task-IL Mean Acc (%)	Avg. Forgetting (%)
No-Search	58.01	89.43	4.70
iCaRL (NME)	53.76	84.02	11.35
DER++	49.93	88.76	38.61
TRACER (Search-ON)	61.36	90.55	4.51

We see that TRACER (Search-ON) compares to No-Search, iCaRL, and DER++ using the same Split CIFAR-100 protocol (in terms of backbone, memory and training). No-Search has worse performance as depicted in Table 5.4, which implies that a fixed head is not as adaptable between the variable tasks. iCaRL and DER++ achieve greater stability through replay, yet, nevertheless, continue to deteriorate on successive tasks. All in all, it is indicated that TRACER provides a higher final CIL

accuracy at the lowest average forgetting, implying that task-adaptive selection of head and search are beneficial compared to using replay alone.

4.3.2 Accuracy–Forgetting Trade-off Discussion

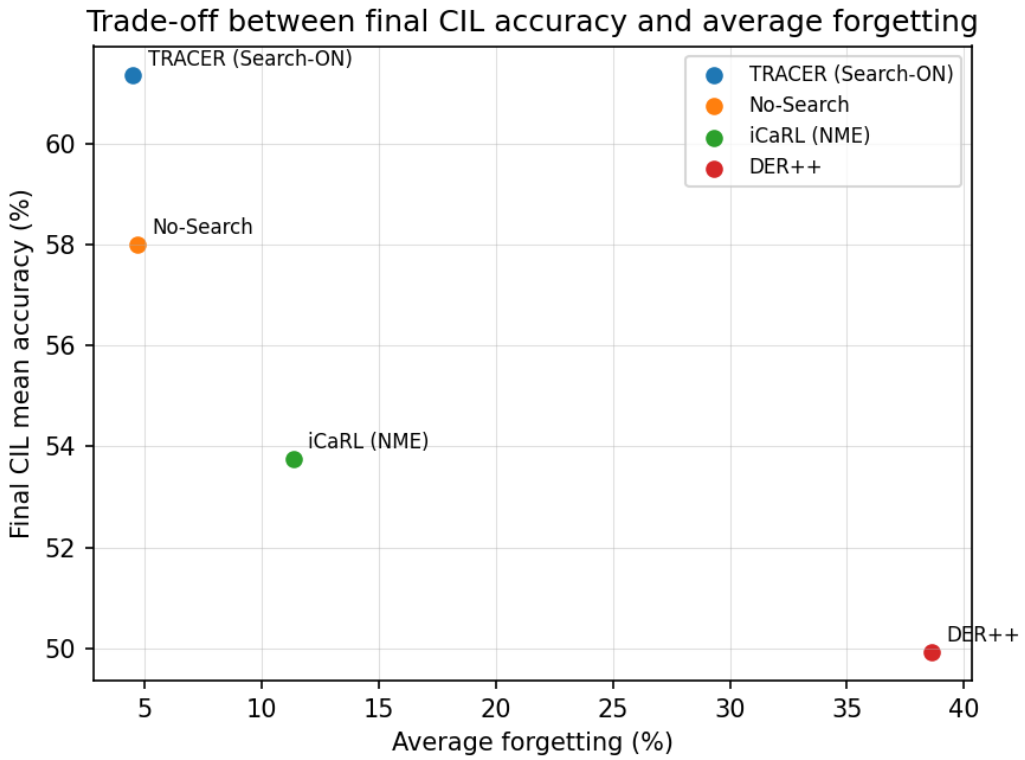


Figure 4.12: The trade-off between final CIL accuracy and average forgetting.

Figure 4.12 shows a summary of the trade off between final CIL accuracy and mean forgetting. Even though replay-based techniques produce lower levels of forgetting than No-Search, their accuracy improvements are minimal. TRACER provides greater accuracy and reduced forgetting, indicating that head selection with controller leads to preservation of task discrimination without high levels of interference that are achieved by replay-based methods.

4.3.3 Split CIFAR-10 (after Task 5)

We draw a comparison between TRACER (Search-ON) and the oracle continual learning strategies iCaRL and DER++ using the identical Split CIFAR-10 protocol (5 tasks, 2 classes per task) with an identical backbone network (ResNet-18). TRACER is the type of task-adjusted head selection based on the controller, and iCaRL and DER++ are dependent mostly on replay to shrink forgetting.

Table 4.2: Final performance comparison after Task 5 (Split CIFAR-10).

Method	**Final CIL Mean Acc (%)**	**Final Task-IL Mean Acc (%)**	**Avg. Forgetting (%)**
iCaRL (NME)	77.70	95.50	—
DER++	77.30	87.50	—
TRACER (Search-ON)	82.06	97.00	3.41

According to Table 4.2, TRACER shows the best meaning Final Class-Incremental Learning (CIL) performance of all the methods evaluated with a high Task-Incremental Learning (Task-IL) accuracy. Specifically, TRACER achieves the ultimate CIL mean accuracy of 82.06% and low average forgetting (3.41%), which means that the previously learned classes are actually preserved. In contrast, despite iCaRL and DER++ being more stable with replay, the ultimate instability of CIL in this experiment with their methods is worse. On the whole, these findings confirm that task-adaptive and controller-guided head selection offers more than replay-based methods as evidenced by higher performance under class-incremental assessment.

4.4 Additional Diagnostics

4.4.1 Confusion Matrix (Normalized)

Figure 4.13 presents the normalized confusion matrix over all CIFAR-100 classes. The matrix shows a strong diagonal structure, indicating that most samples are correctly classified into their true class. The off-diagonal entries are comparatively sparse and low-intensity, meaning that misclassifications are distributed across relatively few confusing alternatives per class rather than being uniformly random. Overall, the confusion matrix confirms that the classifier has learned class-discriminative representations, while remaining errors mainly arise from visually similar categories.

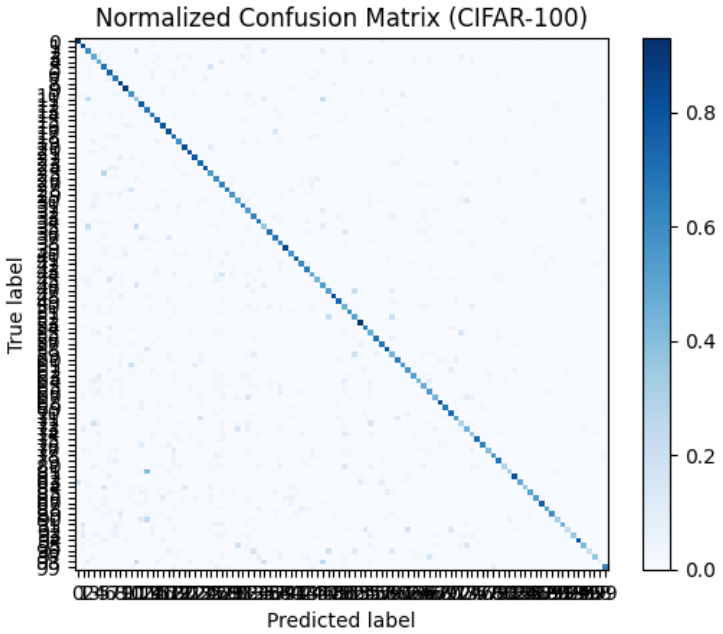


Figure 4.13: Confusion matrix (CIFAR-100) is normalized (over gray box). The darker diagonal values represent the correct prediction and the off-diagonal values represent the misclassifications. The overpowering diagonal tendency demonstrates much class separability, and the left over confusion is moved across visually similar classes.

4.4.2 ROC Curve (example: class 0) + ROC-AUC (macro/micro)

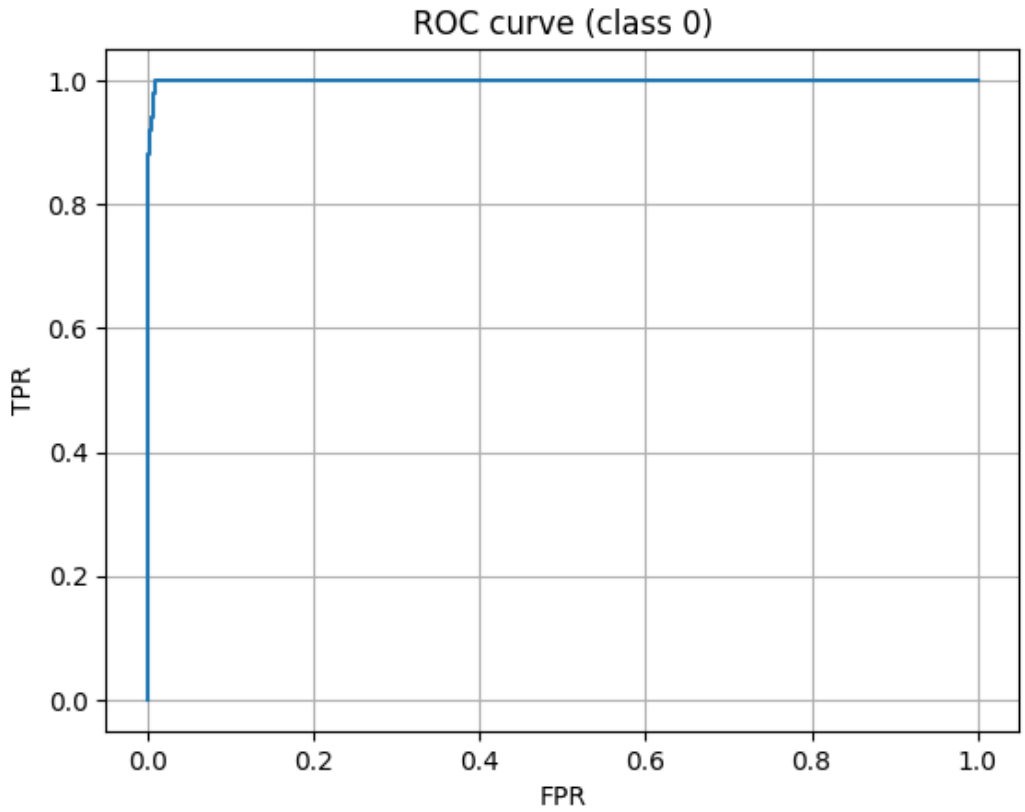


Figure 4.14: ROC curve for class 0. It means that the curve is close to the top-left corner meaning that it is very sensitive at low false-positive rates. High ROC-AUC (macro: 0.9860; micro: 0.9849) confirms overall discrimination.

The ROC curve of a typical class (class 0) is presented in Figure 4.14. This curve is kept near the upper-left corner which proves that the model has high true-positive rate (TPR) but has extremely low false-positive rate (FPR) at all levels. The high score of the total ROC-AUC scores are consistent (macro: 0.9860, micro: 0.9849) which shows a good separability between positive vs. negative samples of all classes.

4.4.3 Precision–Recall Curve (example: class 0) + PR-AUC (macro/micro)

Figure 4.15 reports results of precision at different decision thresholds and the recall of class 0 by curves known as precision-recall (PR) curve, that focuses more on precision than on recall. As indicated in the PR curve, there is a high level of precision at a large range of recall, after which it decreases slowly as the recall approaches 1.0. The global PR-AUC values (macro: 0.6740, micro: 0.6587) are not as high as ROC-AUC is to be expected in multi-class problems where the class prior is very small for each one-vs-rest assessment (e.g. 1 out of 100 classes). Notably, PR-AUC is significantly higher than the random baseline, which provides evidence of the significant ranking quality.

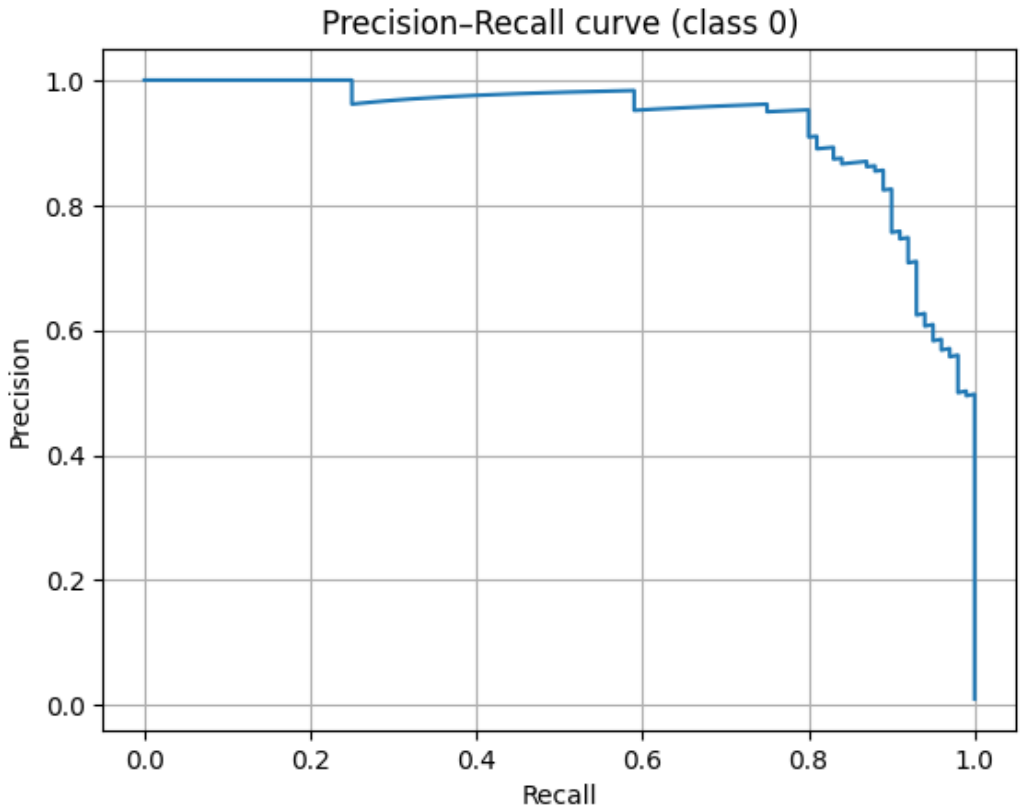


Figure 4.15: Precision–recall curve for class 0. PR curves emphasize precision under threshold variation. The PR-AUC (macro: 0.6740; micro: 0.6587) reflects ranking quality under low per-class prevalence and complements ROC-AUC analysis.

4.5 Discussion

TRACER evaluation in this chapter was based on two protocols; class-incremental (CIL) and task-incremental (Task-IL) but latency limitations related to the edges were explicitly taken into consideration. TRACER achieved an end final CIL mean accuracy of 0.6136 on Split CIFAR-100 and a great deal more Task-IL of 0.9055. The gap is predicted to happen in the continuous learning process and it signifies that a primary portion of the challenge is the class-ambiguity aroused by the joint evaluation of all the classes and not the inability to learn each task when it is jointly evaluated together with the task identity. As with the regular stream behaviour, forgetting was concentrated in early tasks and little forgetting was observed in later tasks.

TRACER exhibited a steady high level of Task-IL performance on Split CIFAR-10 in both loose and strict latency environments, but CIL accuracy was observed to reduce as tasks were added. Significantly, the design-solution outcomes demonstrate that TRACER does not depend on a given head: the selected head varies with tasks and varies in relaxed and tight who-done regime. Plots of latency also verify that the p95 latency of the selected-head is following the desired level of constraint, which proves that the controller is actively using latency in the selection process.

Lastly, the mutation analysis indicates that the search mechanism does investigate quality of architectural variations: candidate heads can be used to provide

greater accuracy with minimal rise in latency or less latency with minimal accuracy cost, which is of particular concern in difficult circumstances. Comprehensively, the findings validate the central argument within TRACER, namely, that based on a latency-conscious optimization criterion, task-adaptive head selection can offer a viable accuracy-latency stream when continual learning occurs on edge devices.

4.6 Future Work

The realism as well as the generalization of the proposed framework can be improved in the future. To begin with, the latency lower bound may either be deployment-faithful (i.e. the current scaling-based proxy has been replaced with execution of hardware-in-the-loop profiling (i.e. ONNX/TensorRT) on the target edge device) or a trained latency predictor based on actual measurements. Second, backbone freezing can be used to stabilize and make a system more efficient, but it has been suggested that instead, lightweight adaptation (e.g. adapters/LoRA or even partial unfreezing schedules) can enable that system to be more flexible in complex (open-ended) learning tasks without creating a conflict with the latency constraint. Third, they could improve risk-conscious aspects through more benchmark risk estimators, a combination of the CVaR accuracy with calibration/ uncertainty indicators and smooth distribution of the exemplars over time to decrease variability. Finally, this approach must be experimented in larger continuous learning settings (domain-incremental, long tailed, higher resolution datasets) and with task orders held varied in order to learn more robustness and controller sensitivity.

Chapter 5

Conclusion

This paper has presented TRACER, a progressive image classification method, which is formulated to be used with realistic edge constraints, meaning there should be no catastrophic forgetting and models should also satisfy inference-latency constraints. The algorithm incorporates the use of a frozen ResNet18 feature extractor and trains a population of small heads of classifiers, which are chosen and updated task-switched.

One of the contributions is that continual learning is viewed as a risk-sensitive, limited optimization problem as opposed to accuracy-based training. The reward has been used to make the model selection towards a trade-off between validation, stability in previously learned classes, and the latency feasibility ability of the solution applications, thus selecting solutions can be made in the form of solutions that can be deployed. Additional risk sensitivity is added to the form of CVaR-style signals, a risk-sensitive exemplar allocation policy that puts more weight in memory on harder or more prone to error classes on a fixed budget.

In general, the findings reinforce the thesis statement to the effect that hybrid search (RL + evolution) with constraint-aware rewards and risk-aware rehearsal form a viable way to attain continuous learning systems that will be both stable and deployment-ready. It should be further applied in future work on partial backbone adaptation (e.g. adapters), on hardware-in-the-loop validation of latency, and to the wider range of continual learning benchmarks to more effectively test generality and robustness tothe real world.

Bibliography

[1] R. T. Rockafellar and S. Uryasev, "Conditional value-at-risk for general loss distributions," *Journal of Banking & Finance*, vol. 26, no. 7, pp. 1443–1471, 2002. DOI: 10.1016/S0378-4266(02)00271-6. Accessed: Dec. 28, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378426602002716.

[2] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Technical Report, Apr. 2009, Version dated April 8, 2009. Accessed: Nov. 30, 2025. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[3] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013. DOI: 10.1145/2408776.2408794. Accessed: Sep. 1, 2025. [Online]. Available: https://cacm.acm.org/research/the-tail-at-scale/.

[4] A. Tamar, Y. Chow, M. Ghavamzadeh, and S. Mannor, "Policy gradient for coherent risk measures," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. Accessed: Sep. 26, 2025. [Online]. Available: https://arxiv.org/abs/1502.03919.

[5] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2016. Accessed: Jun. 19, 2025. [Online]. Available: https://arxiv.org/abs/1511.05641.

[6] Z. Li and D. Hoiem, "Learning without forgetting," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. Accessed: Feb. 22, 2025. [Online]. Available: https://arxiv.org/abs/1606.09282.

[7] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "Icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. Accessed: Jul. 27, 2025. [Online]. Available: https://arxiv.org/abs/1611.07725.

[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 4095–4104. Accessed: Mar. 6, 2025. [Online]. Available: https://arxiv.org/abs/1802.03268.

[9] T.-J. Yang et al., "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 289–304. DOI: 10.1007/978-3-030-01249-6_18. Accessed: Sep. 3, 2025. [Online]. Available: https://arxiv.org/abs/1804.03230.

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. Accessed: May 29, 2025. [Online]. Available: https://arxiv.org/abs/1707.07012.

[11] M. De Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *arXiv preprint arXiv:1909.08383*, 2019. Accessed: Feb. 5, 2025. [Online]. Available: https://arxiv.org/abs/1909.08383.

[12] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019. DOI: 10.1016/j.neunet.2019.01.012. Accessed: Jan. 28, 2025. [Online]. Available: https://arxiv.org/abs/1802.07569.

[13] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, "Dark experience for general continual learning: A strong, simple baseline," *arXiv preprint arXiv:2004.07211*, 2020. arXiv: 2004.07211. Accessed: Nov. 7, 2025. [Online]. Available: https://arxiv.org/pdf/2004.07211v1.

[14] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations (ICLR)*, 2020. Accessed: Jan. 8, 2026. [Online]. Available: https://arxiv.org/abs/1908.09791.

[15] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce activations, not trainable parameters for efficient on-device learning," *arXiv preprint arXiv:2007.11622*, 2020. arXiv: 2007.11622. Accessed: Sep. 14, 2025. [Online]. Available: https://arxiv.org/pdf/2007.11622.

[16] A. Douillard, M. Cord, C. Ollion, T. Robert, and E. Valle. "Podnet: Pooled outputs distillation for small-tasks incremental learning." arXiv: 2004.13513, Accessed: Oct. 11, 2025. [Online]. Available: https://arxiv.org/abs/2004.13513.

[17] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "Remind your neural network to prevent catastrophic forgetting," in *European Conference on Computer Vision (ECCV)*, 2020. Accessed: Oct. 5, 2025. [Online]. Available: https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123530460.pdf.

[18] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020. DOI: 10.1016/j.inffus.2019.12.004. Accessed: Apr. 9, 2025. [Online]. Available: https://arxiv.org/abs/1907.00182.

[19] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. Accessed: Aug. 18, 2025. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/86c51678350f656dcc7f490a43946ee5-Paper.pdf.

[20] X. Liu et al., "Generative feature replay for class-incremental learning," *arXiv preprint arXiv:2004.09199*, 2020. Accessed: Aug. 14, 2025. [Online]. Available: https://arxiv.org/abs/2004.09199.

[21] A. Prabhu, P. H. S. Torr, and P. K. Dokania, "Gdumb: A simple approach that questions our progress in continual learning," in *European Conference on Computer Vision (ECCV)*, 2020. Accessed: Oct. 29, 2025. [Online]. Available: https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123470511.pdf.

[22] M. De Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *arXiv preprint arXiv:1909.08383*, 2021. arXiv: 1909.08383. Accessed: Jan. 19, 2026. [Online]. Available: https://arxiv.org/pdf/1909.08383.

[23] D. Shim, Z. Mai, J. Jeong, S. Sanner, H. Kim, and J. Jang, "Online class-incremental continual learning with adversarial shapley value," *arXiv preprint arXiv:2009.00093*, 2021. arXiv: 2009.00093. Accessed: Jun. 8, 2025. [Online]. Available: https://arxiv.org/pdf/2009.00093.

[24] K. Wang, L. Herranz, and J. van de Weijer, "Acae-remind for online continual learning with compressed feature replay," *arXiv preprint arXiv:2105.08595*, 2021. arXiv: 2105.08595. Accessed: Mar. 17, 2025. [Online]. Available: https://arxiv.org/pdf/2105.08595.

[25] G. Merlin, V. Lomonaco, A. Cossu, A. Carta, and D. Bacciu, "Practical recommendations for replay-based continual learning methods," *arXiv preprint arXiv:2203.10317*, 2022. arXiv: 2203.10317. Accessed: Feb. 11, 2025. [Online]. Available: https://arxiv.org/pdf/2203.10317.

[26] Z. Wang et al., "Dualprompt: Complementary prompting for rehearsal-free continual learning," in *European Conference on Computer Vision (ECCV)*, 2022. Accessed: Nov. 16, 2025. [Online]. Available: https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136860617.pdf.

[27] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *arXiv preprint arXiv:2302.00487*, 2023. arXiv: 2302.00487. Accessed: Dec. 3, 2025. [Online]. Available: https://arxiv.org/pdf/2302.00487.

[28] A. Dequino et al., "Compressed latent replays for lightweight continual learning on spiking neural networks," *arXiv preprint arXiv:2407.03111*, 2024. arXiv: 2407.03111. Accessed: Aug. 26, 2025. [Online]. Available: https://arxiv.org/pdf/2407.03111.

[29] Z. Gao, J. Cen, and X. Chang, "Consistent prompting for rehearsal-free continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. Accessed: May 6, 2025. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2024/papers/Gao_Consistent_Prompting_for_Rehearsal-Free_Continual_Learning_CVPR_2024_paper.pdf.

[30] H. Shi et al., "Continual learning of large language models: A comprehensive survey," *arXiv preprint arXiv:2404.16789*, 2024. arXiv: 2404.16789. Accessed: Sep. 12, 2025. [Online]. Available: https://arxiv.org/pdf/2404.16789.

[31] J. S. Smith et al., "Adaptive memory replay for continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2024. Accessed: May 21, 2025. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2024W/ELVM/papers/Smith_Adaptive_Memory_Replay_for_Continual_Learning_CVPRW_2024_paper.pdf.

[32] D. Yu et al., "Recent advances of multimodal continual learning: A comprehensive survey," *arXiv preprint arXiv:2410.05352*, 2024. arXiv: 2410.05352. Accessed: Jul. 18, 2025. [Online]. Available: https://arxiv.org/pdf/2410.05352.

[33] D.-W. Zhou, H.-L. Sun, J. Ning, H.-J. Ye, and D.-C. Zhan, "Continual learning with pre-trained models: A survey," *arXiv preprint arXiv:2401.16386*, 2024. arXiv: 2401.16386. Accessed: Jan. 22, 2026. [Online]. Available: https://arxiv.org/pdf/2401.16386.

[34] S. Li, Y. D. Kwon, L.-H. Lee, and P. Hui, "Metaclbench: Meta continual learning benchmark on resource-constrained edge devices," *arXiv preprint arXiv:2504.00174*, 2025. arXiv: 2504.00174. Accessed: Apr. 23, 2025. [Online]. Available: https://arxiv.org/pdf/2504.00174.

[35] Y. Liu et al., "Continual learning for vlms: A survey and taxonomy beyond forgetting," *arXiv preprint arXiv:2508.04227*, 2025. arXiv: 2508.04227. Accessed: Aug. 7, 2025. [Online]. Available: https://arxiv.org/pdf/2508.04227.

[36] M. F. Minhas, R. V. W. Putra, F. Awwad, O. Hasan, and M. Shafique, "Replay4ncl: An efficient memory replay-based methodology for neuromorphic continual learning in embedded ai systems," *arXiv preprint arXiv:2503.17061*, 2025. arXiv: 2503.17061. Accessed: Mar. 29, 2025. [Online]. Available: https://arxiv.org/pdf/2503.17061.

[37] C. Tian et al., "Clone: Customizing llms for efficient latency-aware inference at the edge," *arXiv preprint arXiv:2506.02847*, 2025. arXiv: 2506.02847. Accessed: Jun. 30, 2025. [Online]. Available: https://arxiv.org/pdf/2506.02847.