

<https://doi.org/10.1038/s44387-025-00028-4>

Building adaptive knowledge bases for evolving continual learning models

Jack Julian¹ ✉, Yun Sing Koh¹ & Albert Bifet^{2,3}

Continual learning addresses catastrophic forgetting and knowledge transfer when learning from task streams. Dynamic architectures have introduced task-specific components like adapters layered over fixed pre-trained backbones. However, identifying the task of a new input remains a core challenge, leading to task-agnostic and dynamic detection methods. Existing approaches often overlook the reuse of previously learned adapters, missing opportunities for efficient forward and backwards transfer. We propose Continual Adapter-Based Learning (CABLE), a reinforcement learning framework that computes gradient similarity between new examples and past tasks. This similarity score drives a policy that assigns existing adapters when beneficial, rewarding improved performance and reducing reliance on newly initialised parameters. CABLE adopts a dynamic adapter routing strategy without assuming prior task labels. Evaluations on image classification and time series forecasting show that CABLE mitigates catastrophic forgetting and promotes efficient knowledge transfer across tasks.

Continual learning^{1–5} is an approach that aims to incrementally learn a sequence of tasks to achieve two principal objectives: mitigating catastrophic forgetting and fostering knowledge transfer between tasks within an evolving setting.

One critical issue in continual learning is the inability of conventional models to maintain a stable performance across multiple tasks when exposed to new ones. This phenomenon of catastrophic forgetting arises due to the static nature of the model's parameters, which are typically optimized for a fixed dataset. As new data is introduced, these parameters are modified to fit the new information, eroding previously learned tasks. Conversely, humans can use previously learned tasks to aid in understanding new tasks, and new information can sometimes assist in further understanding previous concepts. Therefore, the problem of continual learning is not only preventing catastrophic forgetting but also allowing knowledge to be transferred.

Dynamic architectures reduce the possibility of catastrophic forgetting by expanding the number of parameters available for training by new tasks. Current methods train additional task-specific layers known as adapters or experts on top of a frozen, pre-trained backbone^{6,7} to enable continual learning while achieving positive transfer from an immutable knowledge base. Moreover, recent continual learning approaches have begun to assume that the task information of incoming training examples is unavailable and must be inferred, resulting in task-agnostic methods^{7,8} and task-detection techniques^{9,10} which allow adapters to be assigned tasks dynamically. However, gaps persist in addressing nuanced settings where new tasks may

be learned effectively by existing adapters without forgetting, enabling forward and backwards transfer through the learning process. Existing methods may ignore the possibility that not all tasks require new adapters to be learned.

We address the open problem of recognizing tasks that may be learned by the same parameters without forgetting, specifically to encourage positive and negative transfer and decrease the number of additional parameters that need to be initialized in a dynamic architecture setting. To address these gaps, we introduce Continual Adapter-Based Learning (CABLE), a novel approach leveraging reinforcement learning to recognize tasks that may be learned within the same adapter, thereby addressing key limitations in existing methods. CABLE scores the similarity between examples learned by an adapter and a newly encountered task and uses that score to train a policy for adapter task assignment. The policy is rewarded by positive model performance and explores a gamified approach to task recognition in continual learning.

Our contributions are as follows: First, we developed a novel reinforcement learning framework for task assignments in continual learning that aims to minimize forgetting between classes and incentivize positive transfer. Next, we developed a method of forecasting whether a task will likely induce forgetting of a learned classification, allowing a method to adapt and manage how that task is learned. We also introduce an implementation of our method for forecasting maritime trajectories and air quality monitoring. Lastly, we demonstrate how catastrophic forgetting may be mitigated through an adapter-based

¹School of Computer Science, University of Auckland, Auckland, New Zealand. ²AI Institute, University of Waikato, Waikato, New Zealand. ³LTCl, Télécom Paris, IP Paris, France. ✉e-mail: jack.julian@auckland.ac.nz

method and how effective task recognition may restrict the expansion of dynamic architectures.

Results

We conduct experiments to investigate the efficacy of our adapter-based CL method. We design these experiments to answer the following research questions: RQ1: How does a task-selection module using a reinforcement learning technique compare to the continual learning performance of existing adapter-based approaches? RQ2: Can task similarity measures be used to assess the likelihood of a new task introducing forgetting on an adapter during continual learning? RQ3: Adapter-based continual learning models often use knowledge bases pre-trained on large datasets. How does the continual learning performance compare when knowledge bases are pre-trained on fewer data points? RQ4: What are the contributions of CABLE's distinct modules, and how do they contribute to computational cost?

To answer RQ1, we introduce five benchmark datasets and present our experiment hyperparameters before comparing the classification accuracy of CABLE against state-of-the-art continual learning methods. We then investigate the sensitivity of our method by varying key hyperparameters and comparing classification accuracy. We also compare the task-selection agent to a baseline with random task assignment. To address RQ2, we show the task similarity values corresponding to each forgetting event observed over our adapters and demonstrate a relationship. To answer RQ3, we compare adapter-based baselines after limiting the knowledge base to a subset of classes on the learned dataset. Finally, to answer RQ4, we conduct an ablation study on CABLE's components and assess the computational cost of the CABLE framework compared to continual learning baselines.

Benchmark image datasets

We use 6 benchmark image datasets for all experiments: Fashion MNIST¹¹, CIFAR-100¹², Mini ImageNet (M-IN)¹³, COIL-100¹⁴, CUB¹⁵, and CORe50¹⁶. We order each dataset as a sequence of classes and introduce examples sequentially to all methods, ensuring that the distribution of incoming examples changes over time. For all datasets, the testing set used to determine the classification accuracy is a random 20% split of examples. Table 1 presents an overview of each dataset.

Time series forecasting use-cases

For our maritime trajectory prediction use case, we use the publicly available National Oceanic and Atmospheric Administration (NOAA) AIS dataset¹⁷ for training. This dataset is a 24-hour recording of AIS transmissions filtered to one-minute intervals for all United States coastal waters. A single AIS contains six features we use for training: a unique Maritime Mobile Service Identity (MMSI), timestamp, latitude, longitude, speed over ground (SOG) and course over ground (COG). The NOAA dataset was chosen because it exhibits various behaviours over a large ocean area. We observe sparse and highly trafficked areas and popular fishing zones and port regions with complex trajectory patterns.

For preprocessing, we first index all AIS transmissions by MMSI and order these transmissions by timestamp. A series of five consecutive transmissions represents a trajectory. Latitude and longitude values are

converted to Universal Transverse Mercator (UTM) coordinates to preserve consistent spacing between points. Vessels with frequent missing values are discarded.

For our air quality forecasting use case, we use four $PM_{2.5}$ air quality datasets from around the world. First, the Beijing, China air quality dataset contains readings from 12 sensors in different districts across Beijing from 2013 to 2017¹⁸. Second, the Delhi, India air quality dataset contains readings from a single sensor in Delhi from 2020 to 2023¹⁹. The Sofia, Bulgaria air quality dataset contains readings from 50 sensors between 2017 and 2019²⁰. Finally, the Vinnytsia, Ukraine air quality dataset contains readings from 24 sensors between 2019 and 2021²¹. Each dataset has a temporal granularity of 1 hour.

Baseline methods

We evaluate various existing continual learning methods, including memory replay-based approaches, regularization approaches, and dynamic architectures with adapters. The following baseline methods are compared to CABLE: the replay-based approaches ER²², ER+GMED²³, PCR²⁴, CoPE²⁵ and OCM²⁶, and the adapter-based approaches SEDEM²⁷, MoE-Adapters⁷, DyTox⁶, CMoA²⁸ and EASE²⁹. Unless otherwise stated, we do not adjust these baselines' internal machine learning architecture.

To evaluate CABLE on AIS data, we compare the trajectory prediction metrics of six state-of-the-art deep learning trajectory prediction models: Linear prediction, Bidirectional Gated Recurrence Unit (Bi-GRU)³⁰, Bidirectional Long-Short Term Memory (Bi-LSTM)³¹, TrAISformer³², Masked Autoencoders for Trajectory Prediction (Traj-MAE)³³ and PromptCast³⁴.

Reproducibility

The experiments were implemented and evaluated in the Python programming language. They were conducted on an NVIDIA A100 GPU with CUDA compatibility. Except for sensitivity tests, we use the following hyperparameters obtained via grid search for our experiments: $B = 0.1$, $\epsilon = 0.2$, $b = 50$. Adapters are composed of two convolutional layers and are appended to a knowledge base consisting of a pre-trained CLIP unless otherwise stated. When an adapter $\mathcal{A}_i$ is created, its parameters are randomly sampled from a Gaussian distribution. During continual learning, the stochastic gradient descent optimizer was used for adapter learning, with batch size 32, weight decay 0.0005, and momentum 0.9. The Adam optimizer was adopted for fine-tuning adapters, with an initial learning rate of 0.001, which decayed by a factor of 10 at the 55th and 80th, respectively.

Performance metrics

We use three performance metrics for experiments on image classification: classification accuracy, transfer, and severity. Classification accuracy refers to the average classification accuracy over all classes once a model has finished training. Transfer is the average change in classification accuracy after introducing a new task. Severity is a newly defined measure described in Equation (15).

Time series forecasting problems require different metrics. For maritime trajectories, we use Average Displacement Error (ADE), Final Displacement Error (FDE), and Interval Accuracy (IAcc). ADE represents the average L^2 distance between the actual position y_v^t for vessel v at time t , and the average predicted position $\hat{y}_v^t$ across all time steps, defined by

$$\text{ADE} = \frac{\sum_{v=1}^V \sum_{t=0}^{T_{len}} \|\hat{y}_v^t - y_v^t\|_2}{VT_{len}}, \quad (1)$$

where V is the number of vessels and T_{len} is the total number of timesteps predicted. FDE is the average L^2 distance between $y_v^{T_{len}}$ and $\hat{y}_v^{T_{len}}$, in other words, the average distance between the final actual and predicted position for each trajectory:

$$\text{FDE} = \frac{\sum_{v=1}^V \|\hat{y}_v^{T_{len}} - y_v^{T_{len}}\|_2}{V}. \quad (2)$$

Table 1 | Overview of the image datasets

Dataset	# Examples	# Classes	Test Set	Training Set
Fashion MNIST (FMNIST)	70,000	10	14,000	56,000
CIFAR100	70,000	100	14,000	56,000
Mini ImageNet (M-IN)	60,000	100	12,000	48,000
COIL-100	7200	100	1440	5760
CUB	11,788	200	2358	9430
CORe50	164,866	50	32,973	131,893

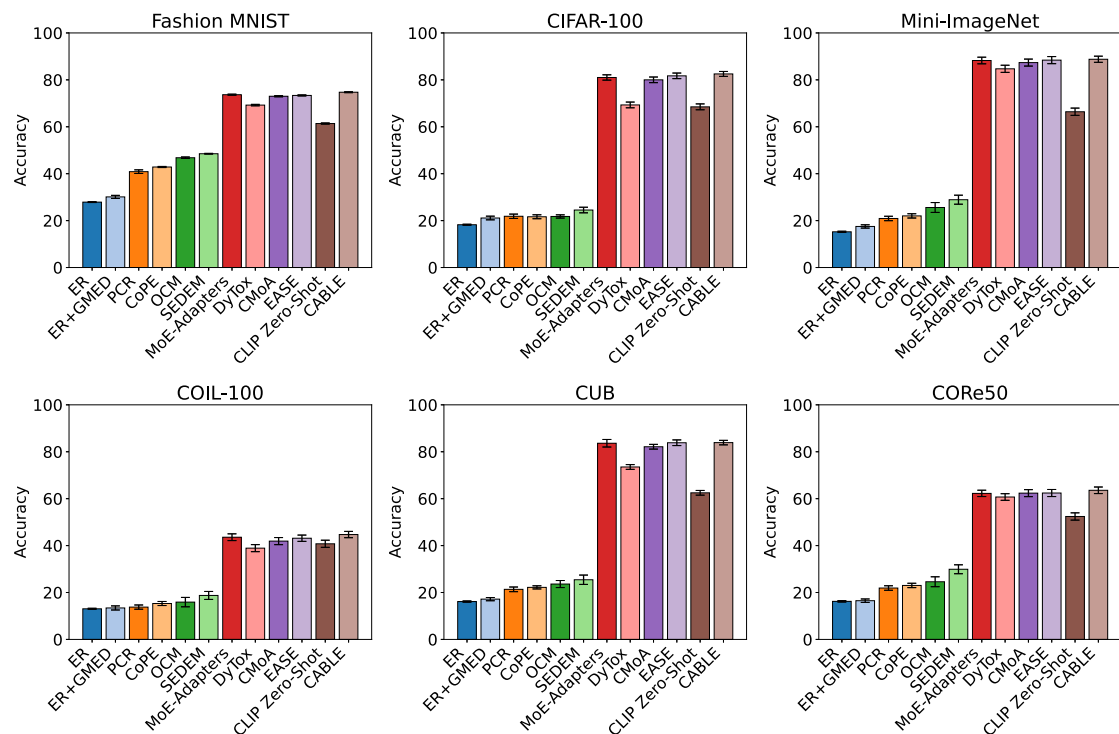


Fig. 1 | Baseline classification accuracy. The classification accuracy of baseline algorithms with CABLE on benchmark datasets.

Finally, IAcc is the percentage of total predicted positions within two standard deviations of the actual position:

$$\text{IAcc} = \frac{\sum_{v=1}^V \sum_{t=0}^{t=T_{len}} acc_v^t}{VT_{len}}, \quad (3)$$

where acc_v^t defines whether a predicted point lies within range:

$$acc_v^t = \begin{cases} 1 & \text{if } \hat{y}_v^t - 2\sigma_v^t < y_v^t < \hat{y}_v^t + 2\sigma_v^t \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

For air quality forecasting, we use Mean Absolute Error (MAE), which measures the average absolute difference between the predicted and actual result.

Classification accuracy (RQ1)

We analyze the classification accuracy of CABLE in image recognition tasks and in AIS trajectory forecasting. Figure 1 compares the accuracy of baseline models against CABLE. We find that CABLE achieves state-of-the-art results on image classification compared to deep learning models and MoE approaches, and note the performance gain compared to CLIP Zero-Shot, demonstrating the utility of the adapter component.

We further analyze how baseline models perform over individual tasks during one epoch of model training. As new tasks are introduced, we expect the performance of previous tasks to decrease due to forgetting. Figure 2 compares the accuracy of baseline methods as tasks from benchmark datasets are learned. In this figure, results are provided for each model without a pre-trained CLIP backbone to illustrate better the process of forgetting during training.

We also compare different RL policies to Clipped PPO. Table 2 shows the results of CABLE with Clipped PPO and CABLE with a random task selection policy that assigns a new class to a random adapter, Trust Region Policy Optimization (TRPO)³⁵, PPO³⁶, Asynchronous Advantage Actor-Critic (A3C)³⁷ and Advantage Actor-Critic (A2C)³⁸. Here, to better demonstrate the performance impact of each RL method, we omit a pre-trained backbone. We find that random task assignment yields less accurate

results compared to RL methods, and observe that Clipped-PPO gives the highest accuracy out of the baseline RL approaches. To investigate this, we also show the normalised reward curves of various RL approaches on three key datasets in Fig. 3. We observe that while approaches such as A2C and A3C experience a steeper gain in reward earlier in training, Clipped-PPO exhibits a superior long-term reward, possibly due to its bounded policy updates.

Table 3 shows the performance of CABLE on maritime trajectory forecasting with a pre-trained time series prediction model, PromptCast, along with a series of baseline models on a variety of metrics. Figure 4 shows examples of some AIS trajectories, each showing different behaviours. We observe that PromptCast does not outperform existing domain-specific approaches, though its performance is elevated by the CABLE framework, demonstrating the importance of adaptation and indicating that superior results may be achieved by employing CABLE on top of pre-trained models on selected domains. Similarly, Table 4 shows results of CABLE with PromptCast on air quality forecasting. While CABLE consistently improves the performance of the knowledge base, classic deep-learning approaches maintain the benefit of multiple passes over the training data.

Ablation study (RQ4)

We conduct a series of ablation tests to assess the contributions of the individual components of CABLE. First, we substitute the RL component for an agent that randomly selects an adapter for each new class encountered in the data stream, representing an ablation of the task selection policy. We further remove the task similarity measure from the reward function, instead equating reward to adapter loss on the current batch. Finally, we remove the buffer mechanism, instead only saving the last encountered batch of each class in memory.

Table 5 presents the performance of CABLE under ablations of its RL component, the task similarity term within the reward function, and the replay buffer. Randomised task selection results in reduced accuracy relative to RL-assisted task assignment, indicating the advantage of structured task allocation. Eliminating task similarity from the reward function yields performance comparable to random task assignment, suggesting that loss alone is inadequate for estimating transferability between adapters and

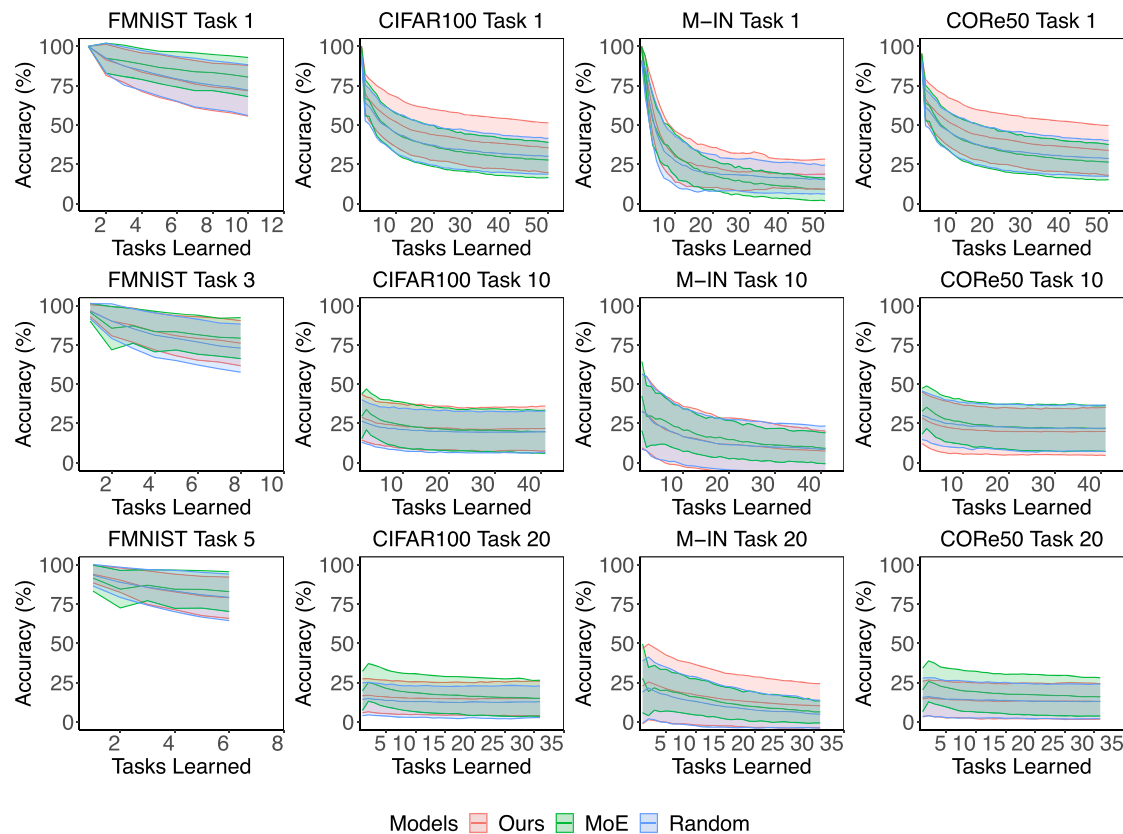


Fig. 2 | Classification accuracy over predefined tasks for benchmark datasets. Models compared are MoE Adapters, CABLE and random task selection. The CLIP backbone is not pre-trained to show the continual learning process across tasks better.

Table 2 | Accuracy of CABLE with different RL methods for task assignment

Methods	FMNIST	CIFAR100	M-IN	CUB	CRe50
CABLE+Random	17.36%	18.55%	15.23%	18.11%	17.22%
CABLE+TRPO	19.15%	19.52%	16.98%	17.79%	17.07%
CABLE+PPO	20.01%	20.13%	17.09%	17.85%	17.59%
CABLE+A2C	20.01%	20.13%	17.09%	18.86%	18.15%
CABLE+A3C	20.12%	20.18%	17.41%	18.84%	18.11%
CABLE +Clipped PPO	21.24%	21.17%	18.22%	19.05%	18.95%

novel tasks. When restricting the buffer size to $b = 1$, accuracy mirrors that of the loss-only reward setting, implying that effective estimation of task similarity requires a sufficiently large sample from the source task.

Sensitivity (RQ4)

We investigate the sensitivity of CABLE to the following hyperparameters: reward bias B , Clipped-PPO boundary ϵ , adapter buffer size b , and number of initial adapters N . Figure 5 provides the accuracy of CABLE on our benchmark datasets over a range of values for each hyperparameter. Buffer size b ranges from 25 and 125, clipping boundary ϵ ranges from 0.2 and 1.0, and reward bias B ranges from 0.1 and 0.5. While varying a hyperparameter, other hyperparameters are set to the following default values: $B = 0.1$, $\epsilon = 0.2$, $b = 50$. We notice diminishing returns as the buffer size b increases beyond 50, indicating that this limit is sufficient to assess task similarity. A higher clipping boundary limits the magnitude of policy changes, and thus performance decreases at high ϵ . With high bias, CABLE is prone to

misrepresenting relationships between tasks, and thus performance decreases. When the initial number of adapters is higher, the RL module degrades as the action space is larger. Therefore, we observe a slight decrease in performance. On the other hand, when the number of adapters is too low, the number of trainable parameters is too small to effectively learn the tasks, and so accuracy is lower.

Computational cost

We compare the computational cost of CABLE with existing baselines to assess its memory and time efficiency during training. Table 6 shows a comparison between CABLE and existing approaches. Due to the need to train CABLE's RL policy, it carries additional computational demands; however, due to a frozen backbone, there are fewer trainable parameters in the model, and thus the computational expense is comparable to approaches such as experience replay.

Transfer and task similarity (RQ2)

Many task similarity measures have been proposed for multi-task and continual learning. We select three measures to compare with ours and observe their performance over our adapters trained on tasks with positive and negative transfer. Positive transfer occurs when the overall performance accuracy is improved after training on a new task. Negative transfer occurs when the overall performance accuracy is reduced after training on a new task.

Table 7 shows the average value of existing task similarity measures for tasks exhibiting positive and negative transfer on a set of adapters pre-trained on 50% of tasks in each benchmark dataset. Although different task measures cannot be directly compared, a lower value of each measure ideally indicates that a task will introduce negative transfer.

Fig. 3 | Reward curves. Reward of different RL components with the CABLE framework.

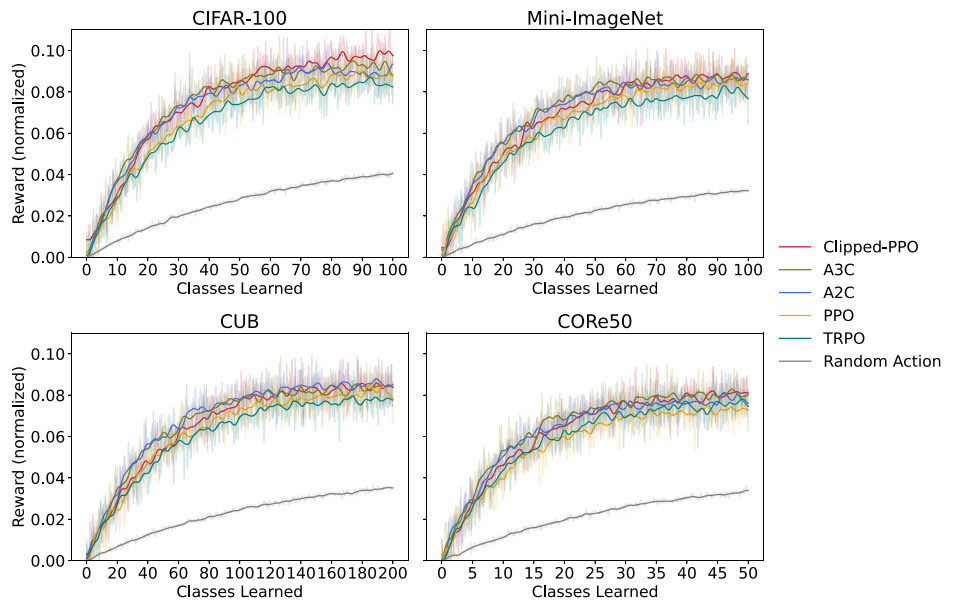


Table 3 | Performance statistics for AIS trajectory prediction

Methods	ADE (↓)	FDE (↓)	IAcc (↑)
Linear	0.41 ± 0.04	1.04 ± 0.14	4.59% ± 1.01
Bi-GRU ³⁰	0.39 ± 0.12	0.92 ± 0.22	49.52% ± 5.10
Bi-LSTM ³¹	0.34 ± 0.09	0.74 ± 0.21	68.16% ± 6.91
Traj-MAE ³³	0.33 ± 0.14	0.71 ± 0.25	70.26% ± 6.51
TrAISformer ³²	0.33 ± 0.12	0.72 ± 0.21	71.41% ± 5.94
PromptCast ³⁴	0.38 ± 0.11	0.79 ± 0.31	62.82% ± 6.11
PromptCast+CABLE	0.31 ± 0.09	0.70 ± 0.30	72.22% ± 6.09

As indicated by arrows, it is desired to have a lower ADE and FDE, and a higher IAcc.

Task similarity of forgotten examples (RQ2)

We analyze the effectiveness of the task similarity measure by evaluating its ability to forecast forgetting between two tasks trained on an adapter. Using Equation (15), we compare the severity of incoming examples with the task similarity of the same examples.

Figure 6 plots the severity of newly encountered examples and the task similarity of those new examples compared with learned examples in the adapter's memory buffer side by side for comparison.

Adapter-based setting with smaller knowledge base (RQ3)

Existing adapter-based approaches employ a frozen knowledge base, often pre-trained with a high volume of data. To determine the effectiveness of the continual learning components in each approach, we decrease the number of pre-trained examples in the knowledge-base and restrict that data to a subset of the training dataset. DyTox and MoE-Adapters, like CABLE, both employ pre-trained knowledge bases with a continual learning component.

Table 8 shows the classification accuracy of continually learned components of the CORe50, Mini ImageNet, CUB and CORe50 datasets when the knowledge bases for all baselines are trained on 20% and 40% of classes.

Discussion

Adapting large pre-trained models to learn continuously elevates evolving classification performance and reduces the need for large-scale retraining to acquire new knowledge^{39–42}. We propose a framework for building adaptive knowledge bases for evolving continual learning models. By integrating task-specific adapters into pre-trained models such as CLIP and ResNet18,

we demonstrate a new approach to the problem of task interference and transfer in continual learning systems. Our approach leverages a task similarity measure to allocate tasks to appropriate adapters, ensuring that each adapter is specialized for the most relevant subset of knowledge while mitigating catastrophic forgetting. Incorporating a reinforcement learning algorithm further enhances the system's adaptability, allowing for continual updates as new tasks emerge.

Our methodology, CABLE, provides a means to maintain and expand the knowledge base of models without requiring retraining from scratch. The task similarity measure is a mechanism to provide further information about the interaction between two tasks through a look-ahead of parameters. At the same time, the reinforcement learning-based task identification strategy allows for an online, self-optimizing process that continuously refines CABLE's task assignment capabilities based on the current state of the model. By combining these elements, we show that adaptive knowledge bases enable models to remain flexible as they encounter new and diverse tasks over time. Furthermore, we demonstrate that CABLE can be used for a variety of real-world purposes by including a use case for trajectory forecasting and air quality forecasting. In future work, CABLE may be extended to natural language processing by integrating adapters into transformer-based architectures, allowing continual adaptation across evolving corpora, domains, or linguistic tasks without requiring explicit task identifiers. In robotics, future work may be completed to assess CABLE's ability to identify movement tasks in a three-dimensional environment.

Our results suggest that this approach can improve the stability and effectiveness of continual learning for environments where data distributions change dynamically. Moreover, leveraging pre-trained models like CLIP, ResNet18 and PromptCast offers a pathway for transferring knowledge across domains and tasks, removing the computational burden of training large-scale models from scratch. Future work will explore the application of this framework in more complex settings, such as multi-modal learning and real-time adaptation to highly variable environments, where task allocation and continual learning strategies will be key to long-term success. Further work could investigate how a task allocation system responds to concept drift within tasks by re-evaluating task assignments.

Methods

This section details a new method for adapter-based, task-free continual learning utilizing a policy optimization approach to decide tasks. First, we provide preliminaries for the method. After, we describe detecting negative

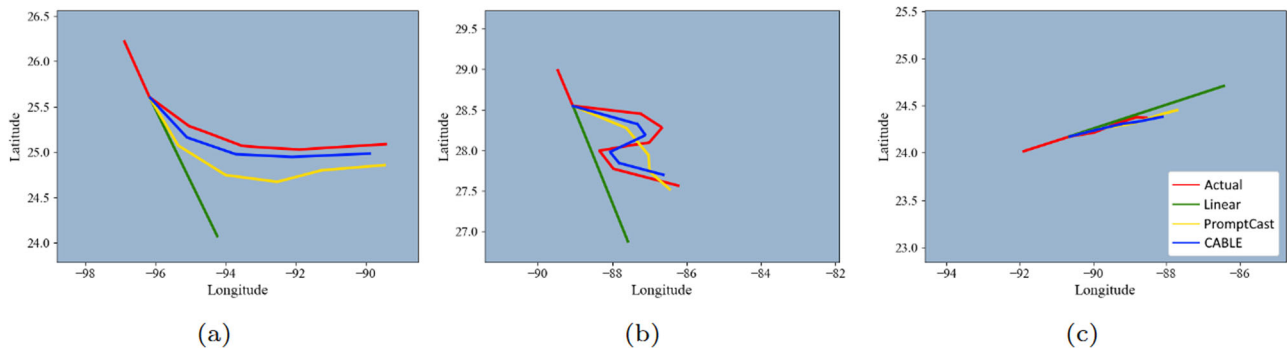


Fig. 4 | Three examples of trajectory forecasting for AIS coordinates with PromptCast+CABLE compared to existing baselines. The examples show three different behaviours. **a** A course change. **b** A multiple turns in a short space of time. **c** A decrease in speed.

Table 4 | MAE of baseline methods on $PM_{2.5}$ for a range of air quality datasets

Methods	Delhi	Beijing	Sofia	Vinnytsia
LSTM	52.26 ± 1.42	50.16 ± 1.61	48.52 ± 1.15	61.34 ± 1.74
Bi-GRU ³⁰	50.72 ± 1.32	47.74 ± 1.51	46.26 ± 1.46	46.12 ± 1.31
Bi-LSTM ³¹	42.15 ± 1.28	41.25 ± 1.64	41.47 ± 1.32	39.67 ± 1.73
Transformer	41.72 ± 1.42	40.61 ± 1.61	41.62 ± 1.15	39.32 ± 1.29
PromptCast ³⁴	42.11 ± 1.51	41.42 ± 1.40	41.68 ± 1.75	39.14 ± 1.57
PromptCast+CABLE	41.98 ± 1.51	41.27 ± 1.40	41.51 ± 1.75	39.00 ± 1.57

inputs by calculating the task similarity between learned tasks and a newly introduced task for each adapter. We then describe how task-free adapters can be applied to a pre-trained learning model. We detail our reinforcement learning module that uses the task similarities to predict which adapter would best benefit from training on a new task or if a new adapter should be created for the new task. Finally, we discuss how task interference is mitigated within our model.

Continual learning with adapters

Here, we define the problem setting and notations used in the article. First, we define a data stream $S = \{(x_i, y_i)\}_{i=1}^{\infty}$, containing a potentially infinite number of examples $x_i \in \mathcal{X}$ each mapped to a class $y_i \in \mathcal{Y}$, where $\mathcal{Y}$ is the space of class labels. Each example is a vector comprised of features denoted as $x_i = (x_i^1, \dots, x_i^n)$, where n is a constant determined by the dataset denoting the number of features in an example.

A continual learning system is a model $\mathcal{M}$ which trains on a data stream S to predict a class $\hat{y}_i \in Y$ for each example x_i such that $\hat{y}_i = y_i$.

Many existing continual learning approaches learn in a *Task Incremental Learning* (TIL) setting, where the model $\mathcal{M}$ trains on a sequence of T tasks $\{T^t\}_{t=1}^T$. Each task $T^t = \{\mathcal{D}^t, \mathcal{C}^t\}$, where $\mathcal{D}^t = \{x_i^t, y_i^t\}_{i=1}^{N^t}$ represents a stream of N^t examples for which $y_i^t \in \mathcal{C}^t$ and $\mathcal{C}^t \subset \mathcal{Y}$. In TIL, the model $\mathcal{M}$ receives a subset of $\mathcal{C}_N^t$ classes $\mathcal{C}^t = \{y_i^t\}_{i=1}^{C_N^t}$ which limit the candidates for each y_i^t prediction. We use a *Class Incremental Learning* (CIL) environment, where the data stream S is used and task information t is not provided with the data, extending the set of candidate class labels to all of $\mathcal{Y}$.

Adapters are modules applied between the layers of a pre-trained model. We leverage the adapter structure proposed by Houlsby et al.⁴³, where each adapter layer A_i transforms the input h as follows:

$$A_i(h) = h + f(h\theta_{\text{down}})\theta_{\text{up}}. \quad (5)$$

First, an input h is projected to a lower-dimensional subspace using weights $\theta_{\text{down}} \in \mathbb{R}^{d \times r}$ and passes to a nonlinear activation function $f(\cdot)$ such as ReLU. The result is projected back to the original dimension through weights $\theta_{\text{up}} \in \mathbb{R}^{r \times d}$. For ease of notation, we denote the parameters of an

adapter A_i as $\theta_{A_i} = \{\theta_{\text{down}}^{A_i}, \theta_{\text{up}}^{A_i}\}$. In a transformer model⁴⁴, adapters are placed after the multi-head attention layer and the feed-forward sub-layer. In deep residual neural networks⁴⁵, adapters are applied between convolutional layers.

Policy optimization

A policy optimization method trains an estimator of the policy gradient through a stochastic gradient ascent algorithm with loss

$$\mathcal{L}^{\text{PG}}(\theta) = \mathbb{E}_t[\log \pi_{\theta}(a_t | s_t) \hat{A}_t] = \frac{1}{t} \sum_{i=0}^{t-1} \log \pi_{\theta}(a_i | s_i) \hat{A}_i,$$

where π_{θ} is a stochastic policy that produces an action a_t with a given state s_t and $\hat{A}_t$ is the estimated advantage at time t . The expected value of $\pi_{\theta}(a_t | s_t)$ is calculated as the empirical average over a batch of samples.

Optimizing $\mathcal{L}^{\text{PG}}$ can often lead to large policy updates that negatively impact model performance³⁵. Therefore, we choose a policy optimization method that constrains the magnitude of policy updates to a defined trust region.

There are two main methods of establishing trust regions for policy optimization: Trust Region Policy Optimization (TRPO) and Clipped Proximal Policy Optimization (PPO). TRPO³⁵ uses the KL divergence of the policy gradient to set a constraint on policy updates:

$$\begin{aligned} & \underset{\theta_t}{\text{maximize}} \quad \mathbb{E}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{t-1}}(a_t | s_t)} \hat{A}_t \right], \\ & \text{s.t.} \quad \mathbb{E}_t \left[\text{KL} \left[\pi_{\theta_{t-1}}(a_t | s_t) \parallel \pi_{\theta}(a_t | s_t) \right] \right] \leq \delta, \end{aligned}$$

where $\pi_{\theta_{t-1}}$ is the previous policy before the update, and δ is the trust region threshold. However, continually calculating $\text{KL}[\pi_{\theta_{t-1}}(a_t | s_t) \parallel \pi_{\theta}(a_t | s_t)]$ at each time-step t is a potentially expensive operation computationally, we will use Clipped PPO.

Clipped PPO sets a range in which the policy can be updated rather than changing penalties over time. This is computationally cheaper than TRPO and improves policy performance⁴⁶. Let $r(\theta_t) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{t-1}}(a_t | s_t)}$. The loss function for Clipped PPO is changed to:

$$\mathcal{L}^{\text{CLIP}}(\theta_t) = \mathbb{E}_t[\min(r(\theta_t)\hat{A}_t, \text{clip}(r(\theta_t), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)], \quad (6)$$

where ϵ is a hyperparameter defining the trust region for the policy. The term $\text{clip}(r(\theta_t), 1 - \epsilon, 1 + \epsilon)\hat{A}_t$ removes the incentive for updating $r(\theta_t)$ outside of the interval $[1 - \epsilon, 1 + \epsilon]$ and prevents large updates that may degrade model performance.

Overview

This article presents a modular, adapter-based framework called CABLE designed to enable pre-trained models to learn additional tasks in a CIL setting without requiring task information to be provided through the data

stream. Instead, task information is inferred by a reinforcement learning (RL) component that is rewarded by instances of positive transfer between tasks. While previous approaches have considered backwards transfer for task identification, we prioritise both forward and backwards transfer and consider underlying class properties through the RL module.

An overview of the CABLE framework is provided in Fig. 7. CABLE contains a set of adapters $\{A_i\}_{i=1}^N$ where N is the number of adapters initialized before training. Each adapter A_i is assigned a task $\mathcal{T}^i$ for which the class set $\mathcal{C}^i$ is unknown at the start of training. As instances of new classes are encountered during training, that is, an example (x_i, y_i) is encountered such that $y_i \notin \bigcup_{i=0}^N \mathcal{C}^i$, the RL module assigns class y_i to an existing adapter based on a policy learned by the RL module. Task information decided by the RL module is used to inform a router $\mathbf{W}^R$, which computes a gated average of adapter outputs to produce a prediction. Unlike existing approaches, the RL module aims to boost both forward and backwards transfer between identified tasks and considers underlying inter-task and intra-task interference.

Motivation and negative inputs

We aim to mitigate forgetting on adapters by avoiding the introduction of *negative inputs*. A negative input is some sequence of examples that, when trained on by an adapter, causes misclassification of previously learned examples. While existing works on continual learning have considered the impact of negative inputs on an entire neural network⁴⁷, in an adapter-based setting, we have separate parameters for each adapter and must consider different behaviours. We adapt *task similarity* from multi-task learning to address this, proposing a novel task similarity measure, and further demonstrate that monitoring a single task similarity measure is not sufficient for task selection, motivating the CABLE framework.

Multi-task learning is a learning paradigm that attempts to learn multiple tasks simultaneously, prioritizing positive transfer between tasks. To achieve this, previous works have proposed measures to determine whether tasks may share parameters to transfer information during gradient

updates^{48–50}. It has been shown that some tasks learn better on shared parameters, and it is beneficial to determine which tasks should and should not be trained on the same parameters⁵¹. Inspired by this, we seek to select tasks for training on separate adapter layers in an evolving, class incremental setting.

Consider an adapter-based learning setting where the loss function is parameterized by $\{\theta_*^t\} \cup \{\theta_{A_i}^t\}_{i=1}^N$ where θ_*^t represents the frozen parameters of the internal deep learning architecture at time t and $\theta_{A_i}^t$ represents the trainable parameters of adapter layer A_i at time t . The loss over an incoming batch of examples $\mathcal{X}^t$ is then given by:

$$\mathcal{L}(\mathcal{X}^t, \theta_*, \{\theta_{A_i}^t\}_{i=1}^N) = \sum_{i=1}^N \mathcal{L}_i(\mathcal{X}^t, \theta_*, \theta_{A_i}^t), \quad (7)$$

where $\mathcal{L}_i$ is the loss for adapter A_i with assigned task $\mathcal{T}^i$. Because θ_* remains frozen for the entire duration of model training, we simplify this notation:

$$\mathcal{L}(\mathcal{X}^t, \{\theta_{A_i}^t\}_{i=1}^N) = \mathcal{L}(\mathcal{X}^t, \theta_*, \{\theta_{A_i}^t\}_{i=1}^N) = \sum_{i=1}^N \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t). \quad (8)$$

When an input is received that matches an adapter's task $\mathcal{T}^i$, that adapter updates its parameters based on the stochastic gradient descent policy with learning rate η :

$$\theta_{A_i}^{t+1} = \theta_{A_i}^t - \eta \nabla_{\theta_{A_i}^t} \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t). \quad (9)$$

We propose that a comparison between the updated parameters $\theta_{A_i}^{t+1}$ and the initial parameters $\theta_{A_i}^t$ may indicate the impact of new examples $\mathcal{X}^t$ on the previously learned task $\mathcal{T}^i$, and whether $\mathcal{X}^t$ may positively or negatively impact performance on $\mathcal{T}^i$. Previous approaches, such as task affinity⁴⁹, perform a lookahead loss; however, they only compare the loss functions with respect to the current examples $\mathcal{X}^t$, which does not consider backwards transfer. We, therefore, propose a new task similarity measure to promote both forward and backwards transfer:

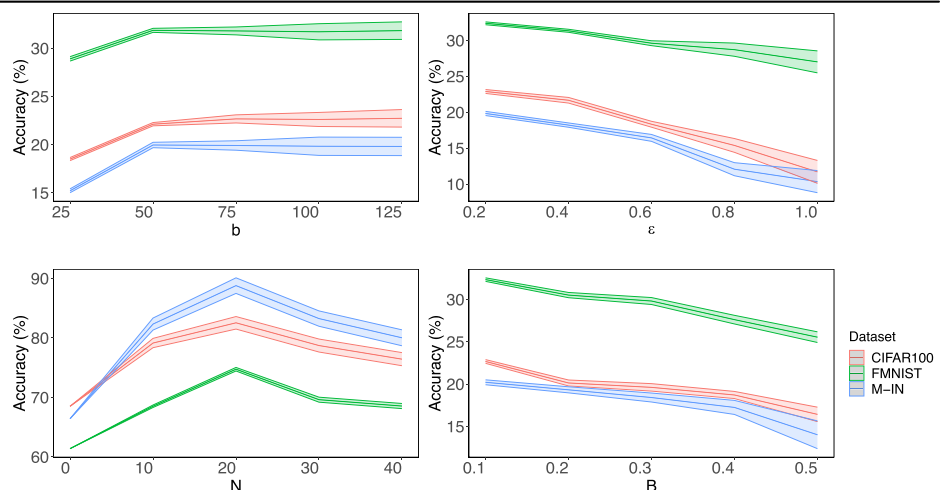
$$\Lambda(\mathcal{X}^t, A_i) = 1 - \frac{\mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^{t+1}) \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^{t+1})}{\mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t) \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t)}, \quad (10)$$

where $\mathcal{X}^i$ represents a set of examples stored in memory previously used to train adapter A_i . A positive value of $\Lambda(\mathcal{X}^t, A_i)$ indicates an overall lower loss over adapter A_i after parameter update $\theta_{A_i}^{t+1}$, and a negative value of $\Lambda(\mathcal{X}^t, A_i)$ indicates a higher loss, and thus poorer performance, after the parameter update. Equation (10) captures both forward and backward transfer by directly quantifying the change in loss on both the new task

Table 5 | Accuracy of CABLE compared against the ablation of the RL module, task similarity and the buffer mechanism

Methods	FMNIST	CIFAR100	M-IN	CUB	CORE50
CABLE+Random	17.36%	18.55%	15.23%	18.11%	17.22%
CABLE+Loss Reward	17.58%	18.61%	15.29%	18.19%	17.71%
CABLE+Limited Buffer	17.56%	18.58%	15.17%	18.26%	17.81%
CABLE	21.24%	21.17%	18.22%	19.05%	18.95%

Fig. 5 | Sensitivity tests of key CABLE hyper-parameters. Classification accuracy of varying buffer size b , clipping boundary ϵ , adapters N and bias B for CABLE over Fashion MNIST, CIFAR-100 and Mini ImageNet datasets.



examples $\mathcal{X}^t$ and previously seen data $\mathcal{X}^i$, before and after the parameter update. Unlike gradient-based similarity measures, which rely on local approximations (e.g., gradient dot products or cosine similarity), $\Lambda(\mathcal{X}^t, A_i)$ evaluates the impact of training via actual loss changes, making it more robust to nonlinear interactions and curvature effects in parameter space. The multiplicative structure provides a normalised comparison between

pre- and post-update performance, avoiding the need for tuning hyperparameters that weigh forward and backwards components.

We now provide a theoretical lower bound for $\Lambda(\mathcal{X}^t, A_i)$. Assume a supervised learning setting with bounded, smooth loss functions. Also assume that $\mathcal{L}_i(\cdot, \theta)$ is L -Lipschitz in θ , $\mathcal{L}_i(\cdot, \theta)$ is β -smooth and $\|\nabla_{\theta} \mathcal{L}_i(\mathcal{X}, \theta)\| \leq G$ for all θ and all inputs. We define relative loss change factors:

$$\Delta_t = \frac{\mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^{t+1})}{\mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t)}, \quad \Delta_i = \frac{\mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^{t+1})}{\mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t)},$$

so that $\Lambda(\mathcal{X}^t, A_i) = 1 - \Delta_t \Delta_i$. Using smoothness and standard gradient descent, we obtain:

$$\mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^{t+1}) \leq \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t) - \eta \|\nabla_{\theta} \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t)\|^2 + \frac{\beta \eta^2}{2} \|\nabla_{\theta} \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t)\|^2.$$

Using the bounded gradient norm, this yields:

$$\Delta_t \leq \bar{\Delta}_t := 1 - \eta \frac{\|\nabla_{\theta} \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t)\|^2}{\mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t)} + \frac{\beta \eta^2 G^2}{2 \mathcal{L}_i(\mathcal{X}^t, \theta_{A_i}^t)}.$$

For backwards transfer, we expand the loss change on $\mathcal{X}^i$:

$$\mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^{t+1}) \leq \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t) - \eta \langle \nabla_{\theta} \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t), \nabla_{\theta} \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t) \rangle + \frac{\beta \eta^2}{2} G^2,$$

leading to:

$$\Delta_i \leq \bar{\Delta}_i := 1 - \eta \frac{\langle \nabla_{\theta} \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t), \nabla_{\theta} \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t) \rangle}{\mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t)} + \frac{\beta \eta^2 G^2}{2 \mathcal{L}_i(\mathcal{X}^i, \theta_{A_i}^t)}.$$

Combining these:

$$\Lambda(\mathcal{X}^t, A_i) \geq 1 - \bar{\Delta}_t \bar{\Delta}_i. \quad (11)$$

This lower bound shows that $\Lambda(\mathcal{X}^t, A_i)$ increases when the gradients from $\mathcal{X}^t$ and $\mathcal{X}^i$ are aligned, and both losses are reduced.

Calculating task similarity requires that a selection of previously trained examples be stored in memory. We keep a buffer $\mathcal{B}_i^y = \bigcup_{j=t-b}^t \{\mathbf{x}_j^i, \mathbf{y}\}$ for each class $\mathbf{y}$, where $\mathbf{x}_j^i$ is the most recent example with class label $\mathbf{y}$ trained on adapter A_i and b is the maximum number of examples in

Table 6 | Comparison of computational cost during training between CABLE and continual learning baselines in terms of GPU burden and training times of each iteration on CIFAR-100 and CORE50

Methods	CIFAR-100		CORE50	
	GPU (I)	Times (I)	GPU (I)	Times (I)
ER	16514MiB	3.51s/it	28125MiB	7.19s/it
OCM	17129MiB	3.15s/it	29419MiB	7.27s/it
EASE	21420MiB	1.49s/it	34103MiB	6.01s/it
MoE-Adapters	22358MiB	1.58s/it	36341MiB	6.20s/it
CABLE	26236MiB	3.96s/it	41362MiB	8.91s/it

Table 7 | Comparison of various state-of-the-art task similarity measures used to indicate the likelihood of positive transfer between tasks

Measures	Positive Transfer (I)			
	CIFAR-100	M-IN	CUB	CORE50
Task Affinity	0.025	0.011	0.004	0.006
Task Distance	0.043	0.013	0.021	0.032
Gradient Similarity	0.026	0.021	0.012	0.041
Task Similarity (Ours)	0.051	0.066	0.052	0.043
Measures	Negative Transfer (I)			
	CIFAR-100	M-IN	CUB	CORE50
Task Affinity	0.006	-0.004	-0.009	0.002
Task Distance	0.026	0.019	0.025	0.006
Gradient Similarity	0.041	0.050	0.054	0.005
Task Similarity (Ours)	-0.004	-0.025	-0.013	0.002

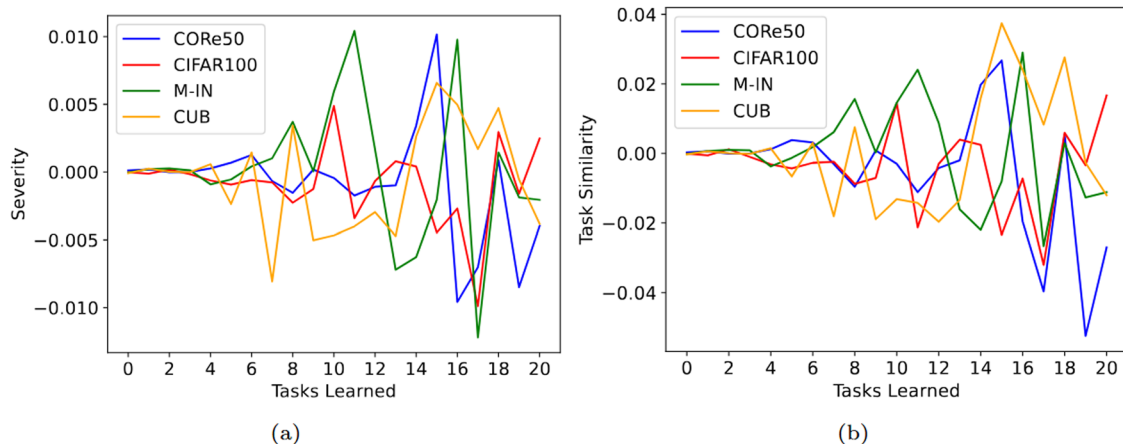


Fig. 6 | Severity and task similarity of CABLE on baseline datasets. a The average severity of newly encountered examples across all adapters as tasks are learned during training. **b** The average task similarity of new examples compared to examples in the buffer of each adapter.

Table 8 | Classification accuracy of baseline algorithms with pre-trained knowledge bases limited to variously sized subsets of the training data from CRe50, Mini ImageNet and CUB

CLIP	20% Pre-Trained			40% Pre-Trained		
	CRe50	M-IN	CUB	CRe50	M-IN	CUB
MoE-Adapters ⁷	19.64%	28.01%	25.91%	22.25%	37.43%	38.38%
DyTox ⁶	19.25%	25.36%	27.36%	21.47%	33.96%	35.71%
Ours	20.14%	29.47%	28.41%	23.26%	39.74%	39.77%
Gain	↑2.55%	↑4.82%	↑3.84%	↑4.54%	↑6.17%	↑3.62%
ResNet-18	20% Pre-Trained			40% Pre-Trained		
	CRe50	M-IN	CUB	CRe50	M-IN	CUB
MoE-Adapters ⁷	18.61%	26.14%	24.61%	28.74%	35.40%	37.41%
DyTox ⁶	19.36%	23.26%	22.60%	26.25%	31.35%	33.39%
Ours	20.01%	27.52%	26.01%	28.75%	38.13%	38.05%
Gain	↑3.36%	↑5.28%	↑5.69%	↑0.03%	↑7.71%	↑1.71%

Classification accuracy is acquired through testing of classes that are not trained by the knowledge base.

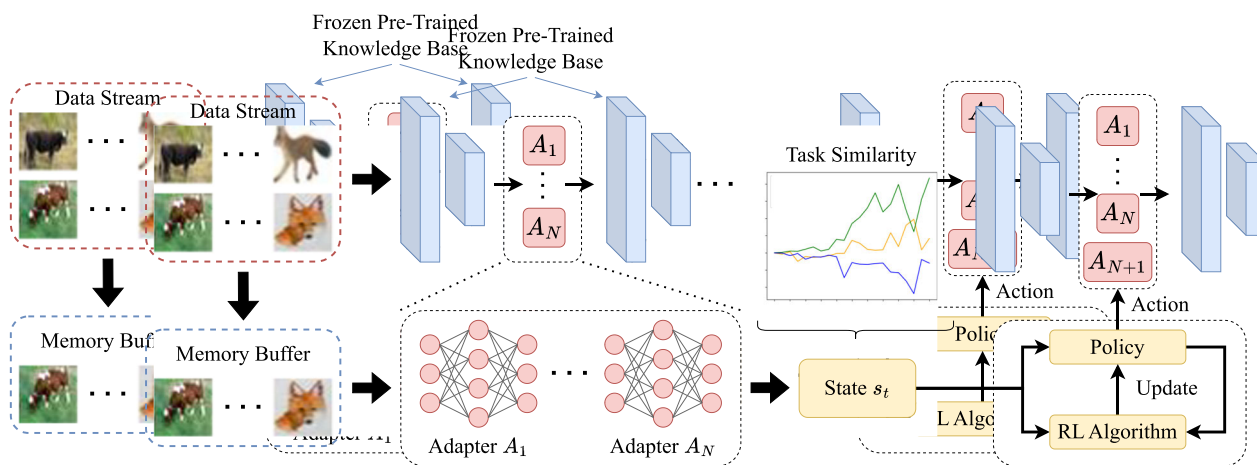


Fig. 7 | Overview of the CABLE framework. Blue components represent the frozen pre-trained backbone, red components represent dynamic continual learning adapters, and yellow components represent the RL components.

buffer $\mathcal{B}_i^y$. To extend the buffer to represent all classes learned by an adapter equally, we set an adapter buffer $\mathcal{X}^i = \bigcup_{y \in \mathcal{C}_i} \mathcal{B}_i^y$ with maximum size $b|\mathcal{Y}|$. This buffer will be used when calculating $\Lambda(\mathcal{X}^t, A_i)$.

To assess whether new examples are negative inputs on adapters, we calculate the task similarity of an incoming batch of examples $\mathcal{X}^t = \{\mathbf{x}_{t-|\mathcal{B}_i|}, \dots, \mathbf{x}_t\}$ from data stream $\mathcal{S}$.

Figure 8 shows an example of task similarity between four classes from the CIFAR-100 dataset. In this example, we find that visually similar classes to humans score highest on average in the task similarity measure. For instance, the Cattle and Fox classes feature quadruped animals, which exhibit transferrable knowledge to a learning system. When the Fox class is compared to an adapter trained on the Cattle class, the mean task similarity of batches from the Fox class is higher than that of the Truck or Television classes. Likewise, when the Cattle class is compared to an adapter trained on the Fox class, it scores the highest task similarity. A task similarity can also be observed between the Television and Truck classes, which often visually contain rectangular geometric shapes. In this case, the classes should be grouped together in a task, as in the process shown in Fig. 9.

However, task similarity can be misleading. In the Fig. 8, the Truck class produces a high similarity spike on batch 12 onto the Cattle class, despite generally being unrelated. This example highlights that while task similarity offers valuable insight, it is not sufficient for reliable adapter assignment. Spurious spikes or short-term artifacts may cause misassignment and lead to

performance degradation, and we further observe that thresholds alone are not well equipped to manage the variation in task similarity scores. This motivates the need for a more robust policy beyond task similarity alone, which we addressed by proposing the CABLE framework.

Adapters with knowledge bases

We utilize adapters to create a scalable architecture that addresses the issue of catastrophic forgetting in the continual learning of large pre-trained models. CABLE consists of multiple adapters $\mathbf{A} = \{A_i\}_{i=1}^N$, where N represents the number of predefined adapters. A set of routers decides on the outputs of our adapters via a gated average. Due to the modular nature of adapters, we propose that CABLE may be implemented on many existing pre-trained deep learning models. To demonstrate this, we implement CABLE on CLIP and ResNet18.

CLIP contains parallel encoders (F_i, F_T) , where F_i represents the image embedding function and F_T represents the text embedding function. Features of input images and corresponding texts are extracted, and the final embeddings' highest cosine similarity determines the input image's class. We implement adapters in all transformer blocks of encoders (F_i, F_T) so that the outputs of a multi-head attention layer are passed to the adapters. ResNet18 is a convolutional model and does not use text encoders, so we implement adapters after each convolutional layer.

A router $\mathbf{W}^R$ is used at each adapter layer to determine which adapters are activated for a new input. When an input $\mathbf{x}_t$ is received, the router

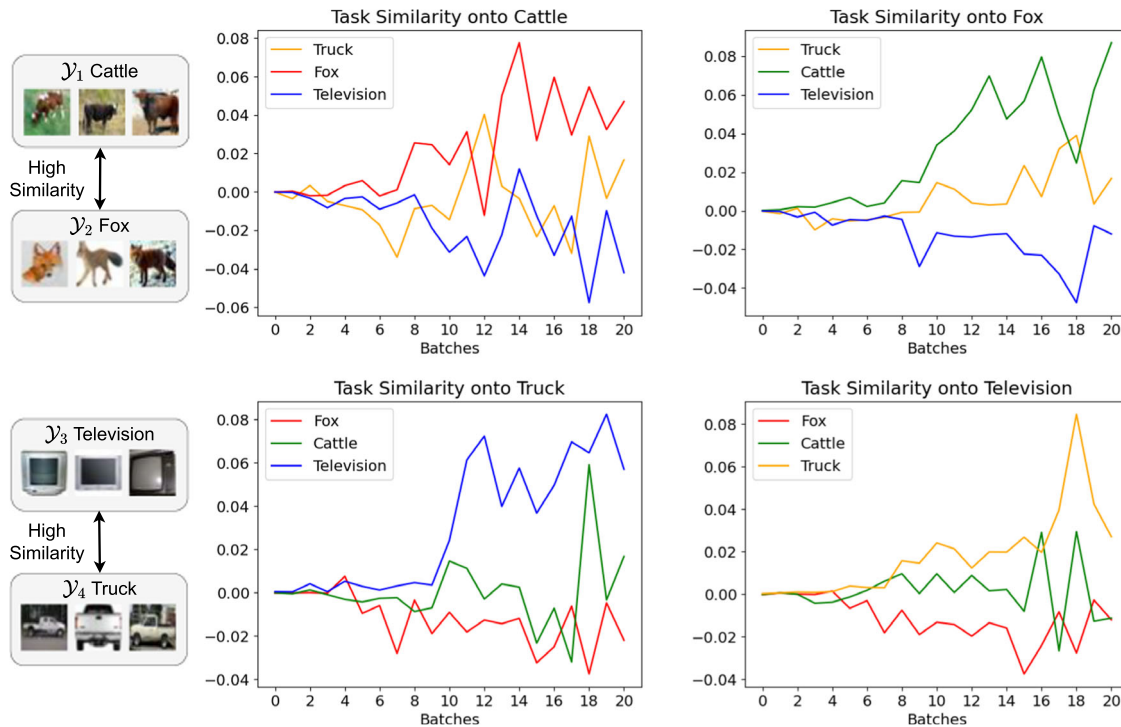


Fig. 8 | Examples belonging to four classes from CIFAR-100 are compared with the task similarity measure. The classes are Cattle, Fox, Television and Truck. Each graph of results is acquired over one adapter training on the first 20 batches of the

target class. After training on each batch, a distinct batch from each of the remaining three classes is chosen to calculate task similarity.

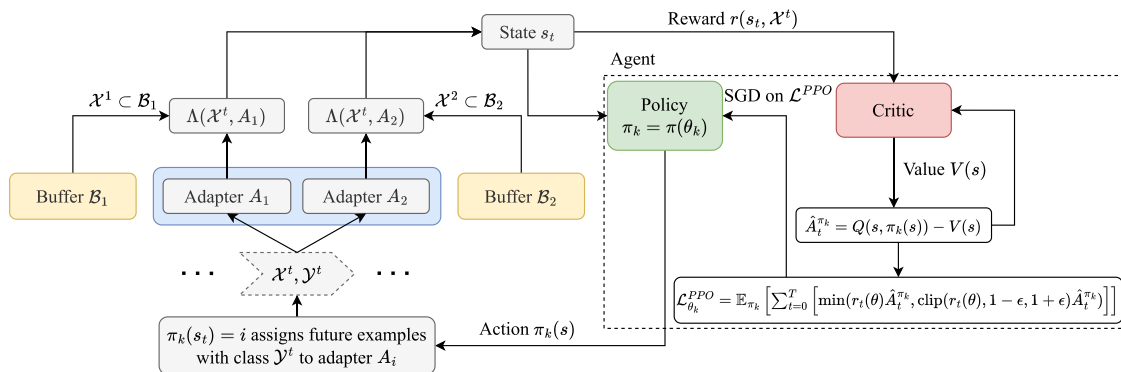


Fig. 9 | An example of agent-based task selection for a model with two adapters, A_1 and A_2 . A new batch with class $\mathcal{Y}^t$, is encountered. The task similarity for each adapter is calculated using buffers of previously learned examples $\mathcal{B}_1$ and $\mathcal{B}_2$. A critic is trained based on model performance and provides feedback to the policy, which

provides an action for the given state of task similarities. The action identifies the adapter best suited to training on examples with class $\mathcal{Y}^t$ based on the current state of the model.

predicts a set of N gating weights: $R(\mathbf{x}_t) = \text{softmax}(\mathbf{W}^R \mathbf{x}_t) \in \mathbb{R}^N$, each corresponding to one of the N adapters. Each router $\mathbf{W}^R$ is learned during training. The combined output $\mathbf{y}_t$ for $\mathbf{x}_t$ is determined by:

$$\mathbf{y}_t = \sum_{i=1}^N W_i^R A_i(\mathbf{x}_t), \quad (12)$$

where $\mathbf{W}^R = \{W_i^R\}_{i=1}^N$. Routers ensure that the output of each adapter is weighted for an input based on that adapter's expertise.

Training a task selection policy using reinforcement learning

We train a policy optimization model to decide whether to add a newly encountered class to the task of an existing adapter or to create a new adapter for that class. The policy should learn not only to

respect existing task similarity calculations but also to predict the advantage of future gradient similarities and the future performance of each adapter. We train our policy by using a variation on Clipped PPO with clipping³⁶.

The state s_t of an adapter-based model with adapters $\mathbf{A}$ is defined as $s_t = \{\mathcal{X}_y^t, \{\mathcal{C}_{A_i}\}_{i=1}^N, \{\Lambda(\mathcal{X}_y^t, A_i)\}_{i=1}^N\}$, where $\mathcal{X}_y^t$ is a batch of new examples with class y that is yet to be assigned to an adapter, $\{\mathcal{C}_{A_i}\}_{i=1}^N$ is the set of classes learned by each adapter, and $\{\Lambda(\mathcal{X}_y^t, A_i)\}_{i=1}^N$ is the task similarity of $\mathcal{X}_y^t$ with each adapter. The state transitions $s_t \rightarrow s_{t+1}$, where $s_{t+1} = \{\mathcal{X}_y^{t+1}, \{\mathcal{C}_{A_i}\}_{i=1}^N, \{\Lambda(\mathcal{X}_y^{t+1}, A_i)\}_{i=1}^N\}$.

An action a_t assigns y to an existing adapter based on this state. The action is decided by a policy $\pi_\theta(s_t)$, which is trained using the loss function presented in Equation (6). The advantage $\hat{A}_t$ is a key factor in training π_θ , representing the expected improvement of action a_t on the model. We

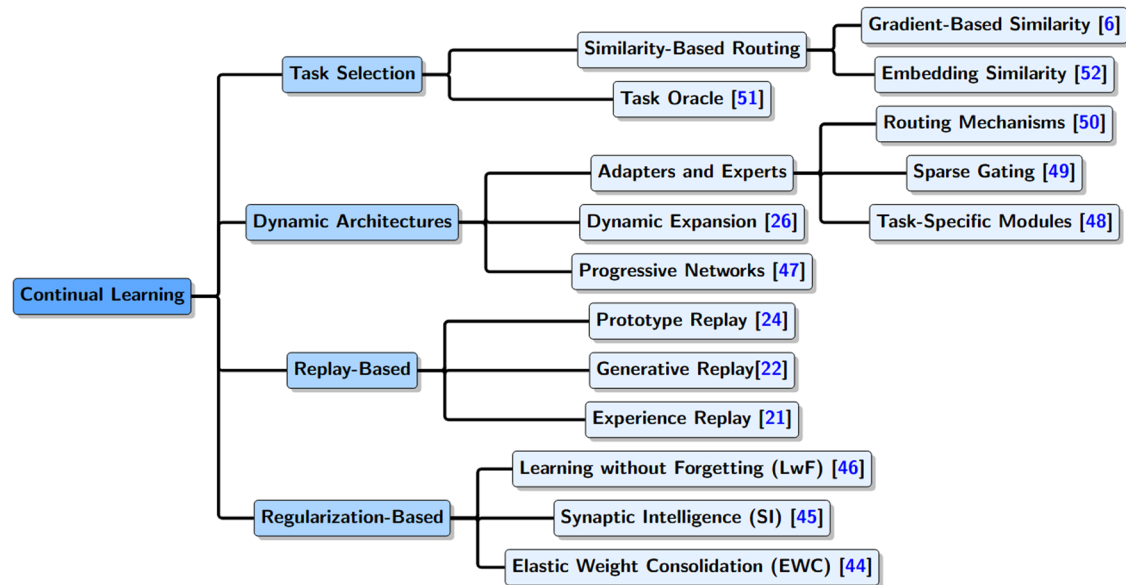


Fig. 10 | A taxonomy of continual learning approaches. We show approaches to four areas of continual learning: task selection, dynamic architectures, replay-based approaches and regularization-based approaches.

define the advantages as follows:

$$Q(s_t, a_t) = \mathbb{E}_t \left[\sum_{i=0}^{\infty} b^i r(s_{t+i}) | s_t, a_t \right], \quad V(s_t) = \mathbb{E}_t \left[\sum_{i=0}^{\infty} b^i r(s_{t+i}) | s_t \right],$$

$$\hat{A}_t = \hat{A}_t(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

where $V(s_t)$ is the value function, the expected reward of a state, and $Q(s_t, a_t)$ is the expected reward given an action in a state. The bias B is set between 0 and 1 and represents the penalty for rewards r further in the future. We define the reward as follows:

$$r(s_t, \mathcal{X}^t) = \frac{1}{B|\mathbf{A}|} \sum_{A_i \in \mathbf{A}} \Lambda(\mathcal{X}^t, A_i) \cdot (y - y')^2. \quad (13)$$

The reward incentivizes a high task similarity between an adapter's assigned tasks and a low loss.

Inter-task and intra-task interference

The continual learning aspect of CABLE uses adapters to train on tasks identified by the RL module. Therefore, there can be no inter-task interference between two identified tasks, as they share no trainable parameters. However, the RL module may incorrectly identify a task if an incoming batch $\mathcal{X}^t$ returns an anomalous $\Lambda(\mathcal{X}^t, A_i)$ for an adapter A_i that does not represent the distribution of the class associated with $\mathcal{X}^t$. This may lead to interference and forgetting within adapter A_i , which we call intra-task interference.

To mitigate intra-task interference, we must reduce the probability that an action is made by policy π_θ on a state that incorrectly represents the incoming class. Intuitively, the more examples from a class that are represented in a state, the better the distribution of that class is captured. We, therefore, take two steps to mitigate intra-task interference:

First, the state space is represented as $S = (X'_y, \{A_i\}_{i=1}^N, \Lambda)$, where $X'_y = \{\mathcal{X}_y^i\}_{i=1}^{N_x}$. This ensures that more examples from class y are interpreted by π_θ before an action is taken, reducing the probability that an anomalous example contributes to the action. Second, we update the reward function:

$$r(s_t, X') = \frac{1}{B|\mathbf{A}||X'|} \sum_{A_i \in \mathbf{A}} \sum_{\mathcal{X}' \in X'} \Lambda(\mathcal{X}', A_i) \cdot (y - y')^2, \quad (14)$$

where a single batch does not determine that policy updates and the mean task similarity of a series of examples is instead considered. This reduces the probability of a policy update that reduces the performance of the RL module.

To further examine interference, we define a *forgetting example* as follows: For any adapter A with parameters θ_A^t trained on a set of examples $(\mathbf{X}, \mathbf{Y})$ up to time t , a *forgetting example* is an example $(\mathbf{X}^F, \mathbf{Y}^F)$, where $\mathbf{Y}^F \cap \mathbf{Y} = \emptyset$, which updates parameters θ_A^t to θ_A^{t+1} at time $t + 1$ such that loss $\mathcal{L}(\theta_A^{t+1}, \mathbf{X}, \mathbf{Y}) < \mathcal{L}(\theta_A^t, \mathbf{X}, \mathbf{Y})$.

Using this definition for a forgetting example, we define the severity of a forgetting example as follows:

$$\text{Sev}(\mathbf{X}^F, \mathbf{Y}^F) = \mathcal{L}(\theta_A^{t+1}, \mathbf{X}, \mathbf{Y}) - \mathcal{L}(\theta_A^t, \mathbf{X}, \mathbf{Y}). \quad (15)$$

A larger severity indicates a more considerable increase in loss for previously learned examples before and after introducing a forgetting example. These definitions help to capture how interference from input examples affects a continual learning model.

Computational complexity

The integration of an RL component in the CABLE framework introduces additional computational overhead. Specifically, we employ Clipped-PPO to train a policy that assigns new class instances to adapters based on estimated transfer dynamics. This section analyses the resulting computational cost.

Let N denote the number of adapters, T the number of time steps per episode, and K the number of rollouts per policy update. Denote by $|\pi|$ the number of parameters in the policy network, C_f the cost of a forward pass through the model and adapter stack, and B the batch size. The RL pipeline consists of three primary stages:

First, the policy rollout stage. At each time step, the policy network selects an adapter based on the current state s_t , and a rollout is executed over a batch of examples X' . The per-step cost is $\mathcal{O}(|\pi|)$ for policy inference and $\mathcal{O}(C_f)$ for evaluating adapter predictions. Across T steps and K rollouts, this contributes $\mathcal{O}(KT(|\pi| + C_f))$.

Second, the reward computation stage. The reward signal at each step is given by Equation (14). For each A_i and each sample in X' , this entails forward passes to compute updated losses and predictions. The worst-case cost of reward computation is $\mathcal{O}(BNC_f)$ per step, giving $\mathcal{O}(KTBN C_f)$ total.

Finally, the policy update stage. Clipped-PPO performs multiple epochs of stochastic gradient descent over the rollout buffer. Let B_{RL} be the batch size used for PPO updates. The policy update phase costs $\mathcal{O}(KB_{\text{RL}}|\pi|)$.

Combining all components, the total additional computational cost introduced by the RL module per training cycle is:

$$\mathcal{O}(KT(|\pi| + BNC_f) + KB_{\text{RL}}|\pi|).$$

This overhead scales linearly with the number of adapters N , batch size B , and PPO hyperparameters. Although the cost is non-trivial, it is offset by parallelisation and amortisation over long training sequences, and is justified by the RL module's role in promoting effective task decomposition and transfer-aware routing.

Time series forecasting

We demonstrate the versatility of CABLE by providing real-world applications in both maritime trajectory forecasting and air quality forecasting. These tasks highlight the advantage of adapting existing pre-trained models to a continual learning environment, showing how CABLE can be applied to diverse time series data domains under noisy and evolving real-world conditions.

For maritime trajectory forecasting, CABLE receives an input trajectory $j_{t-1} = P_1, \dots, P_N$ comprised of T_{len} points $P_i = \text{MMSI}_i, x_i, y_i, \text{sog}_i, \text{cog}_i$, where x_i and y_i are UTM coordinates representing the vessel's spatial location. The goal is to predict a future trajectory $\hat{j}_t = P_1, \dots, P_N, \dots, P_{N+n}$ over the next T_{len} points.

For air quality forecasting, CABLE is provided with a historical input sequence $q_{t-1} = q_1, \dots, q_M$, where each point $q_j = \text{PM}_{2.5,j}, t_j$ represents the PM2.5 concentration and its associated timestamp. The objective is to forecast the future sequence $\hat{q}_t = q_1, \dots, q_M, \dots, q_{M+m}$ of PM2.5 values for the next T_{len} time steps.

In both settings, we use the pre-trained time series knowledge base PromptCast³⁴ as the backbone for CABLE. PromptCast is a generative time series forecasting model that operates by conditioning on a textual prompt to predict future numerical sequences. To enable continual learning, we integrate adapter layers after each attention block in the PromptCast architecture. These adapters allow CABLE to retain previously learned representations while incorporating new domain-specific knowledge. For both maritime and PM2.5 forecasting, where prediction tasks do not involve discrete class labels, Root Mean Squared Error (RMSE) is used as the loss function and as the criterion for assessing task similarity across adaptation steps.

Related work

Machine learning systems today are typically constrained by two major limitations: the lack of adaptability and the need for extensive training data. Pre-trained knowledge bases, such as large language models⁵² and deep residual neural networks⁴⁵, offer a promising solution by providing foundational knowledge that can reduce the need for retraining from scratch. However, incorporating new information is a continuous challenge, particularly when mitigating catastrophic forgetting and ensuring that new knowledge integrates seamlessly with prior learning. Continual learning addresses these limitations by enabling models to adapt to new information over time, avoiding catastrophic forgetting, and preserving previously learned knowledge. We summarise existing literature on continual learning, specifically focussing on continual learning approaches with pre-trained knowledge bases and task-free adapters. Figure 10 provides an overview of continual learning approaches with a focus on dynamic architectures and task selection.

Continual learning

Continual learning has garnered increasing attention in recent literature, with a primary focus on addressing the challenge of catastrophic forgetting. Existing methodologies for mitigating catastrophic forgetting are

commonly classified into three principal categories: experience replay strategies, regularization-based techniques, and dynamic architectural modifications⁵³.

Replay-based methods address catastrophic forgetting by reintroducing data from prior tasks during training on new inputs. This process reinforces previously learned representations, counteracting the tendency of model parameters to drift when adapting to new data in the absence of prior task information. Experience Replay (ER)²² is a replay method that maintains a fixed-size memory buffer containing examples from previously encountered tasks. During training, the model is updated using a mixture of current task data and randomly sampled memory data, enabling it to retain performance on earlier tasks by continuously revisiting past experiences. Many replay techniques, including Gradient-based Memory Editing (GMED)²³ and Proxy-based Contrastive Replay (PCR)²⁴, represent samples in the memory buffer in such a way that they optimize their utility for mitigating forgetting by either enhancing gradient alignment with past tasks or preserving task-specific features through proxy representations. Alternatively, Continual Prototype Evolution (CoPE)²⁵ integrates prototype-based representation learning with replay by maintaining evolving class prototypes instead of raw exemplars, demonstrating that it is unnecessary to memorise raw examples. Furthermore, Online Cooperative Memorisation (OCM)²⁶ samples from short-term and long-term memory buffers, specifically selecting examples that should be memorised long-term. While replay has demonstrated success in reducing forgetting, a major limitation is the need to store and constantly re-train on previously observed concepts.

Regularisation-based approaches to continual learning mitigate catastrophic forgetting by constraining parameter updates to preserve knowledge from previous tasks. These methods introduce additional loss terms that penalise deviations from important weights identified during prior training. Techniques such as Elastic Weight Consolidation (EWC)⁵⁴ and Synaptic Intelligence (SI)⁵⁵ estimate parameter importance and restrict updates accordingly, thereby reducing interference with previously acquired knowledge while allowing adaptation to new tasks. However, in settings with high task dissimilarity, the imposed parameter restraints can restrict model adaptation to new tasks.

Dynamic architectures mitigate the problem of catastrophic forgetting by expanding the number of parameters that may be trained to avoid overwriting parameters important to a previously learned task. The Self-Evolved Dynamic Expansion Model (SEDEM)²⁷ framework evaluates diversity between a set of experts based on examples stored in memory, and creates a new expert to learn novel information. Recent dynamic approaches have applied experts, or adapters, to pre-trained deep learning architectures known as knowledge bases.

Continual learning with knowledge bases

This article investigates the integration of continual learning with pre-trained knowledge bases, considering how the combination can lead to more efficient learning systems capable of lifelong learning without sacrificing the quality or utility of pre-existing knowledge. One of the main challenges in continual learning is ensuring that the new knowledge does not overwrite or diminish the value of pre-existing knowledge in the knowledge base. Knowledge retention strategies can be integrated with continual learning frameworks to prevent the model from forgetting previous knowledge. Techniques like episodic memory⁵⁶ and knowledge distillation⁵⁷ are valuable for ensuring that important information from a model remains intact as the system learns over time; however, even with these methods, catastrophic forgetting has not been eliminated, making frozen pre-trained parameters ideal for applying continual learning to foundational knowledge.

In the context of pre-trained transformer-based models such as CLIP⁵⁸, adapting these models to new tasks or domains without retraining them entirely is a critical challenge. One efficient solution to this problem is the integration of adapters⁵⁹. Adapters are lightweight task-specific modules inserted between the layers of a pre-trained model. Unlike full fine-tuning, which requires adjusting the weights of all model parameters, adapters typically consist of small bottleneck layers that introduce only a small

number of parameters. The key idea is that the large pre-trained model retains its original weights, and only the adapter layers are trained for specific tasks. This reduces the computational cost and memory usage while enabling the model to generalize across various tasks, and multiple adapters prevent inter-task interference. Transformers⁴⁴ are composed of multiple layers, each with attention and feed-forward components. When adapters are added, they are typically inserted after the attention or feed-forward layers in each transformer block. The adapter modules consist of a small multi-layer perceptron with a bottleneck structure—usually a smaller hidden layer followed by an expansion layer. The input to the adapter is the output of the transformer layer, and the output of the adapter is then added to the original layer's output through a residual connection. This ensures that the pre-trained knowledge is not overwritten but complemented by the task-specific modifications introduced by the adapter. The task-specific nature of adapters often requires task information to be accessible in a dataset, which is often not the case in real-world applications. Therefore, we also propose that adapters train in a task-free environment.

With the growing popularity of large-scale foundation models, there has been an increasing interest in adapting pre-trained models to a continual learning setting. Using transformer architectures, DualPrompt⁶⁰ and Learning to Prompt⁶¹ train a set of prompts that update through the continual learning process and report improvements compared to traditional continual learning methods. Fine-tuning knowledge bases has also been a popular approach. The Slow Learner with Classifier Alignment method⁴² fine-tunes a vision transformer knowledge base with a lower learning rate than classifier heads. However, using softmax requires introducing a classifier alignment method, which imposes a significant memory cost as a feature covariance matrix for each class. Fine-tuning has also been applied to larger-scale knowledge bases such as the CLIP model by leveraging the Learning without Forgetting regularization algorithm⁶². Lastly, First Session Adaptation⁶³ avoids updating the knowledge base entirely and instead trains classifier heads known as adapters for each class. Despite promising performance and low memory cost compared to other continual learning methods, this approach struggles to introduce positive transfer between tasks.

Task-free adapters

Class-incremental learning (CIL) refers to the scenario where a continual learning model must learn a sequence of new classes without access to task-specific information. This differs from task-incremental learning (TIL), where examples are received and a task ID specifying that example's relation to a pre-established task. In an adapter-based setting, task IDs can be used to train adapters on specific tasks, further diminishing the possibility of inter-task interference.

The reliance on task IDs during inference presents a key limitation when adapting pre-trained models with task-specific adapters⁶⁴. In scenarios where the model is expected to perform various tasks, the task ID is typically used as a routing mechanism to select the correct adapter for each task. This works well when the task is predefined or known in advance, such as in batch processing, where task information is readily available.

However, the task ID may be unknown during inference in many real-world applications, especially in online or dynamic environments. For instance, when a system is tasked with answering a wide array of queries or handling multiple user requests pertaining to different tasks (e.g., natural language processing, image recognition, or recommendation systems), it is impractical to always provide a task ID. Furthermore, the system may not know what task is required until the input is processed, especially in cases where the input could be related to any number of tasks. This dependency on task IDs makes the task-specific adapter approach unsuitable for environments requiring task-free inference, where the model must dynamically choose the correct adapter without a predefined identifier.

Applying continual learning in a task-free environment has recently been a popular area of research. One approach is task-free memory replay^{23,65,66}, which stores previously experienced examples in memory and interleaves those examples in future training to mitigate catastrophic forgetting.

Maximally Interfered Retrieval⁶⁷ is a task-free replay process that trains variational autoencoders and classifiers by storing the most perturbed examples in replayed memory. Task-free regularization^{68,69} is another approach where penalties are applied to changes made to parameters that contribute to the classification of a previous task.⁶⁴ estimates the importance of each weight to previously experienced classes using memory-aware synapses. The most popular approach in recent research is dynamic architectures, where catastrophic forgetting is avoided by incrementally adding new parameters to avoid overwriting old parameters⁷. expands on the Mixture-of-Experts architecture by applying a Distribution Discriminative Auto-Selector to infer the tasks of examples in the data stream. DYnamic feature space Self-Organization (DYSON)⁷⁰ updates its architecture when new classes are encountered and then aligns the feature space to fit the new geometry. However, existing dynamic architecture approaches often reduce transfer between tasks as each task contains unique classes, eliminating opportunities for positive transfer between different tasks. We aim to encourage both forward and backwards transfer while deciding on task boundaries.

Task similarity is a widely used method for task selection in continual learning, particularly within dynamic architectures such as adapters. By projecting inputs into a shared embedding space, models compute similarity scores against stored representations or centroids to determine the most relevant module or expert. This approach enables task inference without requiring explicit task identifiers. For example⁶¹, utilises embedding similarity for task-conditioned expert selection, while⁷¹ incorporates similarity-based routing in incremental learning frameworks, demonstrating improved adaptability and reduced forgetting. SEDEM²⁷ identifies tasks for existing and new experts by comparing new examples to previously experienced examples stored in a memory buffer, prioritising forward transfer of examples.

Data availability

All datasets used are publicly available: Fashion MNIST (<https://github.com/zalando-research/fashion-mnist>), CIFAR-100 (<https://www.cs.toronto.edu/~kriz/cifar.html>), Mini ImageNet (<https://huggingface.co/datasets/timm/mini-imagenet>), COIL-100 (<https://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php>), CUB (https://www.vision.caltech.edu/datasets/cub_200_2011/), CORE50 (<https://vlomonaco.github.io/core50/>), NOAA maritime data (<https://coast.noaa.gov/htdata/CMSP/AISDataHandler/2020/index.html>), and air quality datasets from Delhi (<https://www.kaggle.com/datasets/deepaksirohiwal/delhi-air-quality>), Beijing (<https://archive.ics.uci.edu/dataset/501/beijing+multi+site+air+quality+data>), Sofia (<https://www.kaggle.com/datasets/hmavrodiev/sofia-air-quality-dataset>) and Vinnytsia (<https://www.kaggle.com/datasets/vbmokin/air-quality-monitoring>).

Code availability

Code for the CABLE implementation in Python is available at <https://github.com/jjul482/CABLE>.

Received: 14 February 2025; Accepted: 29 July 2025;

Published online: 01 October 2025

References

1. Ho, S., Liu, M., Du, L., Gao, L. & Xiang, Y. Prototype-guided memory replay for continual learning. *IEEE Transactions on Neural Networks and Learning Systems* (2023).
2. Smith, J. S., Tian, J., Halbe, S., Hsu, Y.-C. & Kira, Z. A closer look at rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2409–2419 (2023).
3. Madaan, D., Yin, H., Byeon, W., Kautz, J. & Molchanov, P. Heterogeneous continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 15985–15995 (2023).

4. Pham, Q., Liu, C., Sahoo, D. & Steven, H. Contextual transformation networks for online continual learning. In *International Conference on Learning Representations* (2020).
5. Buzzega, P., Boschini, M., Porrello, A., Abati, D. & Calderara, S. Dark experience for general continual learning: a strong, simple baseline. *Adv. neural Inf. Process. Syst.* **33**, 15920–15930 (2020).
6. Douillard, A., Ramé, A., Couairon, G. & Cord, M. Dytox: Transformers for continual learning with dynamic token expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9285–9295 (2022).
7. Yu, J. et al. Boosting continual learning of vision-language models via mixture-of-experts adapters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 23219–23230 (2024).
8. Hihn, H. & Braun, D. A. Hierarchically structured task-agnostic continual learning. *Mach. Learn.* **112**, 655–686 (2023).
9. Kim, G., Esmaeilpour, S., Xiao, C. & Liu, B. Continual learning based on ood detection and task masking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 3856–3866 (2022).
10. Kim, G., Xiao, C., Konishi, T., Ke, Z. & Liu, B. A theoretical study on solving continual learning. *Adv. Neural Inf. Process. Syst.* **35**, 5065–5079 (2022).
11. Xiao, H., Rasul, K. & Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).
12. Krizhevsky, A. et al. Learning multiple layers of features from tiny images (2009).
13. Ravi, S. & Larochelle, H. Optimization as a model for few-shot learning. In *International Conference on Learning Representations* (2016).
14. Nayar. Columbia object image library (coil100) (1996). <https://cir.nii.ac.jp/crid/1572824501191844864>.
15. Dataset, E. Novel datasets for fine-grained image categorization. In *First Workshop on Fine Grained Visual Categorization, CVPR. Citeseer. Citeseer*, vol. 5, 2 (Citeseer, 2011).
16. Lomonaco, V. & Maltoni, D. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on robot learning*, 17–26 (PMLR, 2017).
17. Oceanic, N. & Administration, A. Vessel traffic ais data (2023). <https://marinecadastre.gov/ais/>.
18. Zhang, S. et al. Cautionary tales on air-quality improvement in beijing. *Proc. R. Soc. A: Math., Phys. Eng. Sci.* **473**, 20170457 (2017).
19. Paappanen, M. Weather api (2017).
20. Kristo, A. Sofia weather dataset (2021). <https://doi.org/10.7910/DVN/WIOOVZ>.
21. Chugai, A. & Lavrov, T. Public monitoring as a tool for assessing the state of the air basin in the ukrainian regions. *Ukrainian Hydrometeorol. J.* 81–86 (2022).
22. Riemer, M. et al. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations* (2019).
23. Jin, X., Sadhu, A., Du, J. & Ren, X. Gradient-based editing of memory examples for online task-free continual learning. *Adv. Neural Inf. Process. Syst.* **34**, 29193–29205 (2021).
24. Lin, H., Zhang, B., Feng, S., Li, X. & Ye, Y. Pcr: Proxy-based contrastive replay for online class-incremental continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 24246–24255 (2023).
25. Lange, D., Matthias, Tuytelaars & Tinne. Continual prototype evolution: Learning online from non-stationary data streams. In *Proceedings of the IEEE/CVF international conference on computer vision*, 8250–8259 (2021).
26. Ye, F. & Bors, A. G. Continual variational autoencoder learning via online cooperative memorization. In *European Conference on Computer Vision*, 531–549 (Springer, 2022).
27. Ye, F. & Bors, A. G. Self-evolved dynamic expansion model for task-free continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 22102–22112 (2023).
28. Cui, Y. et al. Cmoa: Contrastive mixture of adapters for generalized few-shot continual learning. *IEEE Transactions on Multimedia* (2025).
29. Zhou, D.-W., Sun, H.-L., Ye, H.-J. & Zhan, D.-C. Expandable subspace ensemble for pre-trained model-based class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 23554–23564 (2024).
30. Wang, C., Ren, H. & Li, H. Vessel trajectory prediction based on ais data and bidirectional gru. In *2020 International conference on computer vision, image and deep learning (CVIDL)* (IEEE, 2020).
31. Yang, C.-H., Wu, C.-H., Shao, J.-C., Wang, Y.-C. & Hsieh, C.-M. Ais-based intelligent vessel trajectory prediction using bi-lstm. *IEEE Access* (2022).
32. Li, Y., Wang, J., Li, T. & Fu, Z. Traformer: Spatio-temporal ship trajectory prediction based on transformer. In *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 1099–1104 (IEEE, 2024).
33. Chen, H. et al. Traj-mae: Masked autoencoders for trajectory prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 8351–8362 (2023).
34. Xue, H. & Salim, F. D. Promptcast: A new prompt-based learning paradigm for time series forecasting. *IEEE Transactions on Knowledge and Data Engineering* (2023).
35. Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. Trust region policy optimization. In *International conference on machine learning*, 1889–1897 (PMLR, 2015).
36. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
37. Mnih, V. et al. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937 (PMLR, 2016).
38. Schulman, J., Moritz, P., Levine, S., Jordan, M. & Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations* (2016).
39. Lee, K.-Y., Zhong, Y. & Wang, Y.-X. Do pre-trained models benefit equally in continual learning? In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 6485–6493 (2023).
40. McDonnell, M. D., Gong, D., Parvaneh, A., Abbasnejad, E. & van den Hengel, A. Ranpac: Random projections and pre-trained models for continual learning. *Advances in Neural Information Processing Systems* **36** (2024).
41. Zhou, D.-W., Sun, H.-L., Ning, J., Ye, H.-J. & Zhan, D.-C. Continual learning with pre-trained models: A survey. In *Proceedings of the International Joint Conference on Artificial Intelligence* (2024).
42. Zhang, G., Wang, L., Kang, G., Chen, L. & Wei, Y. Slca: Slow learner with classifier alignment for continual learning on a pre-trained model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 19148–19158 (2023).
43. Houluby, N. et al. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, 2790–2799 (PMLR, 2019).
44. Vaswani, A. Attention is all you need. *Adv. Neural Inform. Process. Syst.* (2017).
45. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 (2016).
46. Chen, X. et al. The sufficiency of off-policyyness and soft clipping: Ppo is still insufficient according to an off-policy measure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 7078–7086 (2023).

47. Toneva, M. et al. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations* (2019).
48. Adel, T. Similarity-based adaptation for task-aware and task-free continual learning. *J. Artif. Intell. Res.* **80**, 377–417 (2024).
49. Fifty, C. et al. Efficiently identifying task groupings for multi-task learning. *Adv. Neural Inf. Process. Syst.* **34**, 27503–27516 (2021).
50. Lin, X., Baweja, H., Kantor, G. & Held, D. Adaptive auxiliary task weighting for reinforcement learning. *Adv. Neural Inform. Process. Syst.* **32** (2019).
51. Standley, T. et al. Which tasks should be learned together in multi-task learning? In *International conference on machine learning*, 9120–9132 (PMLR, 2020).
52. Chang, Y. et al. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.* **15**, 1–45 (2024).
53. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C. & Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Netw.* **113**, 54–71 (2019).
54. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl Acad. Sci.* **114**, 3521–3526 (2017).
55. Zenke, F., Poole, B. & Ganguli, S. Continual learning through synaptic intelligence. In *International conference on machine learning*, 3987–3995 (PMLR, 2017).
56. Lopez-Paz, D. & Ranzato, M. Gradient episodic memory for continual learning. *Adv. Neural Inform. Process. Syst.* **30** (2017).
57. Li, S., Su, T., Zhang, X. & Wang, Z. Continual learning with knowledge distillation: A survey. *IEEE Transactions on Neural Networks and Learning Systems* (2024).
58. Radford, A. et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, 8748–8763 (PMLR, 2021).
59. Ermis, B., Zappella, G., Wistuba, M., Rawal, A. & Archambeau, C. Memory efficient continual learning with transformers. *Adv. Neural Inf. Process. Syst.* **35**, 10629–10642 (2022).
60. Wang, Z. et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, 631–648 (Springer, 2022).
61. Wang, Z. et al. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 139–149 (2022).
62. Ding, Y., Liu, L., Tian, C., Yang, J. & Ding, H. Don't stop learning: Towards continual learning for the clip model. *arXiv preprint arXiv:2207.09248* (2022).
63. Panos, A., Kobe, Y., Reino, D. O., Aljundi, R. & Turner, R. E. First session adaptation: A strong replay-free baseline for class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 18820–18830 (2023).
64. Aljundi, R., Kelchtermans, K. & Tuytelaars, T. Task-free continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11254–11263 (2019).
65. Roy, S., Verma, V. & Gupta, D. Efficient expansion and gradient based task inference for replay free incremental learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 1165–1175 (2024).
66. Wang, Z. et al. Improving task-free continual learning by distributionally robust memory evolution. In *International conference on machine learning*, 22985–22998 (PMLR, 2022).
67. Aljundi, R. et al. Online continual learning with maximal interfered retrieval. *Adv. Neural Inform. Process. Syst.* **32** (2019).
68. Lässig, F., Aceituno, P. V., Sorbaro, M. & Grewe, B. F. Bio-inspired, task-free continual learning through activity regularization. *Biol. Cybern.* **117**, 345–361 (2023).
69. Pelosin, F., Jha, S., Torsello, A., Raducanu, B. & van de Weijer, J. Towards exemplar-free continual learning in vision transformers: an account of attention, functional and weight regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3820–3829 (2022).
70. He, Y. et al. Dyson: Dynamic feature space self-organization for online task-free class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 23741–23751 (2024).
71. Rosenfeld, A. & Tsotsos, J. K. Incremental learning through deep adaptation. *IEEE Trans. Pattern Anal. Mach. Intell.* **42**, 651–663 (2018).

Acknowledgements

This research is supported by MBIE Strategic Science Investment Fund (SSIF) Data Science platform - Time-Evolving Data Science / Artificial Intelligence for Advanced Open Environmental Science (UOWX1910).

Author contributions

J.J. wrote the main manuscript text and prepared figures. All authors reviewed the manuscript. Y.S.K. and A.B. supervised the work.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to Jack Julian.

Reprints and permissions information is available at <http://www.nature.com/reprints>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025, corrected publication 2025