A Implementation Details of the Compared Methods

We included detailed descriptions, and implementation details of some selected baselines in this section.

• **Experience Replay (ER)** [33, 34] stores examples in a fix-sized memory for future replay. We use reservoir sampling to decide which examples to store and replace. Following prior works [2, 3, 7], at each time step we draw the same number of examples as the batch size from the memory to replay, which are both set to 10. The algorithm is applied as is to the task-free scenario.

• **Gradient Episodic Memory (GEM)** [26] also stores examples in a memory. Before each model parameter update, GEM projects the gradient of model parameters so that the update does not incur loss increase on any previous task. The approach is however not task-free. We used a memory strength $\gamma = 0.5$ for Split MNIST and Rotated MNIST following the original work and $\gamma = 0.5$ following [4]. We applied the same γ for remaining datasets.

• **Averaged Gradient Episodic Memory (AGEM)** [8] prevents the average loss increase on a randomly drawn subset of examples from the memory. We draw $k = 256$ examples to compute the regularization at each iteration. Note than this setup of k is much larger than the number of examples drawn for replay ($k = 10$) in ER approaches. The approach is task-free.

• **Bayesian Gradient Descent (BGD)** [44] is a regularization-based continual learning algorithm. It adjusts the learning rate for parameters by estimating their certainty, noting their importance to previous data. We tune the initial standard deviation of parameters and set it as 0.011 for Split CIFAR-10 and Split CIFAR-100, 0.05 for permuted MNIST, and 0.017 for Split MNIST and Rotated MNIST. We also tune the "learning rate" hyperparameter η and set it as 8 for Split CIFAR-10 and Split CIFAR-100, and 1 for Permuted MNIST, Split MNIST, and Rotated MNIST. The approach is task-free.

• **Gradient based Sample Selection (GSS)** [4] builds upon Experience Replay (ER) by encouraging the diversity in the stored examples. We use GSS-Greedy, which is the best performing variant. The approach is task-free.

• **Hindsight Anchor Learning (HAL)** [7] learns a pseudo "anchor" example per task per class in addition to the replay memory by maximizing its estimated forgetting, and tries to fix model outputs on the anchors at training. However, unlike GMED, HAL estimates forgetting by comparing the loss before and after the model performs updates with the replay memory examples (and thus forgetting is estimated with "hindsight"). We refer to hyperparameters in the original work and set the mean embedding strength $\gamma = 0.1$, anchor learning rate as 0.001, gradient steps on anchor k as 100, and fine-tune 50 epochs on the memory to estimate forgetting across datasets. The approach is not task-free.

• **Maximally Interfering Retrieval (MIR)** [2] improves ER by selecting top forgettable examples from the memory for replay. Following the official implementation, we evaluate forgetting on a candidate set of 25 examples for mini-ImageNet dataset, and 50 examples for others. While the approach is task-free, the official implementation filter out memory examples that belong to the same task as the current data stream, which assumes knowledge about tasks boundaries. We remove this operation to adapt the method to the task-free setup. Therefore, our results are not directly comparable to the official results.

• **Neural Dirichlet Process Model for Continual Learning (CN-DPM)** [22] is a task-free model-expansion based continual learning algorithm. We report the official results in the paper. In the comparison study between ER/ER+GMED with CN-DPM, for the base model in ER/ER+GMED, we use the full expanded model in CN-DPM (*i.e.*, the model architecture when the training ends in CN-DPM). We use the same optimizer and the learning rate as CN-DPM for compared methods in this set of experiments.

• **Progressive Neural Networks (Prog. NN)** [36] is a task-aware model-expansion base approach. We treat linear layers and ResNet blocks as basic components of expansion in MLP and ResNet-18 models, *i.e.*, the model creates a new linear layer or ResNet block at each layer of the model when it encounters a new task. In addition to added components, it also introduces the connectivity between the current component and the components in previous layers as learnable parameters. The model before expansion has the same architecture and sizes as models used in other approaches such as ER.

• **Compositional Continual Learning (CompCL)** [29] is another task-aware model-expansion based approach. When the model encounters a new task, the approach performs two-stage model updates by first learning the architecture over existing and one new model component, and then performing updates on parameters of components. While the original work adds one shared component for all layers, we find it helpful to add separate components for all layers, possibly because the tasks in our setup have low similarity.

B Algorithmic and Implementation Details for GMED variants

Algorithm 2: Memory Editing with ER$_{aug}$ (ER$_{aug}$+GMED)

Input: learning rate τ, edit stride α, regularization strength β, decay rate γ, model parameters θ
Receives: stream example (x_D, y_D)
Initialize: replay memory M

for $t = 1$ **to** T **do**

`// draw a random mini-batch for edit and augmentation respectively`
$(x_e, y_e) \sim M$
$k \leftarrow$ replayed_time(x_e, y_e)
$\ell_{\text{before}} \leftarrow \text{loss}(x_e, y_e, \theta_t)$
$\ell_{\text{stream}} \leftarrow \text{loss}(x_D, y_D, \theta_t)$
`//update model parameters with stream examples, discarded later`
$\theta'_t \leftarrow \text{SGD}(\ell_{\text{stream}}, \theta_t, \tau)$
`//evaluate forgetting of memory examples`
$\ell_{\text{after}} \leftarrow \text{loss}(x_e, y_e, \theta'_t)$
$d \leftarrow \ell_{\text{after}} - \ell_{\text{before}}$
`//edit memory examples`
$x'_e \leftarrow x_e + \gamma^k \alpha \nabla_x (d - \beta \ell_{\text{before}})$
replace (x_e, y_e) with (x'_e, y_e) in M
$(x^a_e, y_e) \leftarrow$ data_augmentation(x_e, y_e)
`//replay edited and augmented examples`
$\ell = \text{loss}(\{(x^a_e, y_e), (x'_e, y_e), (x_D, y_D)\}, \theta_t)$
$\theta_{t+1} \leftarrow \text{SGD}(\ell, \theta_t, \tau)$
reservoir_update(x_D, y_D, M)

end for

Algorithmic Details. We present the algorithmic details of ER$_{aug}$+GMED, MIR+GMED and GEM+GMED in Algorithms 2, 3 and 4. The main difference in MIR+GMED and GEM+GMED compared to ER+GMED is that we edit a separate mini-batch of memory examples from the mini-batch used for replay or regularization. Similarly, in ER$_{aug}$+GMED, we additionally replay a mini-batch of edited examples after data augmentation. The motivations for such design are discussed in Sec. 3.4.

Implementation Details. We implemented our models with PyTorch 1.0. We train our models with a single GTX 1080Ti or 2080Ti GPU, and we use CUDA toolkit 10.1. We use a mini-batch size of 10 throughout experiments for all approaches.

Training Time Cost. For ER+GMED, training on Split MNIST, Permuted MNIST, and Rotated MNIST takes 7 seconds, 30 seconds, and 31 seconds respectively (excluding data pre-processing). Training on Split CIFAR-10, Split CIFAR-100, and Split mini-ImageNet takes 11 minutes, 14 minutes, and 46 minutes respectively. In comparison, training with ER takes 6 seconds, 23 seconds, 16 seconds, 7 minutes, 10 minutes, and 32 minutes respectively on six datasets.

Experiment Configuration for Comparison to Model-Expansion based Methods. Model expansion based approaches such as Progressive Networks [36], Compositional Lifelong Learning [29], and CN-DPM [22] involves additional overhead by introducing new model parameters over time. We compute such overhead as the number of extra parameters when the model has fully expanded. Let n_e, n_b be the number of parameters in a fully-expanded model and a regular model in non-expansion based approaches (*e.g.*, ER) respectively, and the difference is $\Delta n = n_e - n_b$. We report such

Algorithm 3: Memory Editing with MIR (MIR+GMED)

Input: learning rate τ, edit stride α, regularization strength β, decay rate γ, model parameters θ
Receives: stream example (x_D, y_D)
Initialize: replay memory M

for $t = 1$ **to** T **do**

`// draw a random mini-batch for edit`
$(x_e, y_e) \sim M$
$k \leftarrow$ replayed_time(x_e, y_e)
$\ell_{\text{before}} \leftarrow \text{loss}(x_e, y_e, \theta_t)$
$\ell_{\text{stream}} \leftarrow \text{loss}(x_D, y_D, \theta_t)$

`// retrieve a separate mini-batch of examples for replay with MIR`
$(x_m, y_m) \leftarrow$ MIR-retrieve(M, θ)

`//update model parameters with stream examples, discarded later`
$\theta'_t \leftarrow \text{SGD}(\ell_{\text{stream}}, \theta_t, \tau)$

`//evaluate forgetting of memory examples`
$\ell_{\text{after}} \leftarrow \text{loss}(x_e, y_e, \theta'_t)$
$d \leftarrow \ell_{\text{after}} - \ell_{\text{before}}$

`//edit memory examples`
$x'_e \leftarrow x_e + \gamma^k \alpha \nabla_x (d - \beta \ell_{\text{before}})$
replace (x_e, y_e) with (x'_e, y_e) in M

`//replay examples retrieved by MIR`
$\ell = \text{loss}(\{(x_m, y_m), (x_D, y_D)\}, \theta_t)$
$\theta_{t+1} \leftarrow \text{SGD}(\ell, \theta_t, \tau)$
reservoir_update(x_D, y_D, M)

end for

Algorithm 4: Memory Editing with GEM (GEM+GMED)

Input: learning rate τ, edit stride α, regularization strength β, decay rate γ, model parameters θ
Receives: stream example (x_D, y_D)
Initialize: replay memory M

for $t = 1$ **to** T **do**

`// draw a random mini-batch for edit`
$(x_e, y_e) \sim M$
$k \leftarrow$ replayed_time(x_e, y_e)
$\ell_{\text{before}} \leftarrow \text{loss}(x_e, y_e, \theta_t)$
$\ell_{\text{stream}} \leftarrow \text{loss}(x_D, y_D, \theta_t)$

`//update model parameters with stream examples, discarded later`
$\theta'_t \leftarrow \text{SGD}(\ell_{\text{stream}}, \theta_t, \tau)$

`//evaluate forgetting of memory examples`
$\ell_{\text{after}} \leftarrow \text{loss}(x_e, y_e, \theta'_t)$
$d \leftarrow \ell_{\text{after}} - \ell_{\text{before}}$

`//edit memory examples`
$x'_e \leftarrow x_e + \gamma^k \alpha \nabla_x (d - \beta \ell_{\text{before}})$
replace (x_e, y_e) with (x'_e, y_e) in M

`//Regularized model updates with GEM using the full memory`
GEM_regularized_update(x_D, y_D, M, θ)
`//Update the replay memory following GEM`
memory_update(x_D, y_D, M)

end for

overhead in the equivalent number of training examples that can be stored in a memory: given an input image with dimension (c, h, w) and the overhead Δn, the equivalent number of examples is computed as $4\Delta n/(chw + 1)$. The coefficient 4 is added because storing a model parameters in float32 type takes up 4 times as much memory as a pixel or a label y. For CN-DPM, the short-term memory used to store past examples also counts as overhead.

C Hyper-parameter Setup

Throughout experiments other than results reported in Table 2, we use SGD optimizer with a learning rate of 0.05 for MNIST datasets, 0.1 for Split CIFAR-10 and Split mini-ImageNet datasets, and 0.03 for the Split CIFAR-100 dataset across all approaches. The learning rate is tuned with ER method and fixed for all other approaches. For results reported in Table 2, we use the same optimizer and learning rates as CN-DPM. See Sec. A for setup of method specific hyperparameters.

GMED variants. Besides, GMED introduces two additional hyper-parameters: the stride of the editing α, and the regularization strength β. As we assume no access to the full data stream in the online learning setup, we cannot select hyper-parameters according to validation performance after training on the full stream. Therefore, we tune the hyper-parameters with only the validation set of first three tasks, following the practice in [8]. The tasks used for hyper-parameter search are included for computing the final accuracy, following [11]. We perform a grid search over all combinations of α and β and select the one with the best validation performance on the first three tasks. We select α from $[0.01, 0.03, 0.05, 0.07, 0.1, 0.5, 1.0, 5.0, 10.0]$, and select β from $[0, 10^{-3}, 10^{-2}, 10^{-1}, 1]$. We tune two hyper-parameters on ER+GMED and share it across MIR+GMED and GEM+GMED. We tune a separate set of hyper-parameters for ER$_{aug}$+GMED. Table 6 reports the optimal hyper-parameters selected for each dataset.

Table 6: Hyperparamters of the editing stride and the regularization strength selected for ER+GMED.

Dataset / Hyper-param	**Editing stride α**	**Regularization strength β**
ER+GMED, MIR+GMED, GEM+GMED		
Split MNIST	5.0	0.01
Permuted MNIST	0.05	0.001
Rotated MNIST	1.0	0.01
Split CIFAR-10	0.05	0.001
Split CIFAR-100	0.01	0.001
Split mini-ImageNet	1.0	0.1
ER$_{aug}$ + GMED		
Split MNIST	5.0	0.001
Permuted MNIST	0.05	0.001
Rotated MNIST	1.0	0.01
Split CIFAR-10	0.07	0.01
Split CIFAR-100	0.05	0.001
Split mini-ImageNet	0.5	0.0

Sensitivity to γ. Table 7 shows the performance of GMED under various decay rates of the editing stride γ. We find $\gamma = 1.0$ (*i.e.*, no decay) consistently outperforms performance when γ is less than 1. It implies it is not necessary to explicitly discourage the deviation of edited examples from original examples with the hyper-parameter γ.

Table 7: Sensitivity of the performance of GMED to the decay rate of the editing stride (γ).

Methods / Datasets	**Split MNIST**	**Permuted MNIST**	**Rotated MNIST**	**Split CIFAR-10**	**Split CIFAR-100**	**Split mini-ImageNet**
ER	81.07 ± 2.5	78.65 ± 0.7	76.71 ± 1.6	33.30 ± 3.9	20.11 ± 1.2	25.92 ± 1.2
ER + GMED$_{\gamma=0.9}$	82.28 ± 1.7	79.07 ± 0.6	77.22 ± 1.2	33.85 ± 1.4	20.06 ± 1.8	26.92 ± 1.9
ER + GMED$_{\gamma=0.99}$	82.60 ± 2.2	79.15 ± 0.6	77.35 ± 1.3	34.10 ± 3.4	19.90 ± 1.5	27.69 ± 0.7
ER + GMED$_{\gamma=1.0}$	82.67 ± 1.9	78.86 ± 0.7	77.09 ± 1.3	34.84 ± 2.2	20.93 ± 1.6	27.27 ± 1.8
MIR	85.72 ± 1.2	79.13 ± 0.7	77.50 ± 1.6	34.42 ± 2.4	20.02 ± 1.7	25.21 ± 2.2
MIR + GMED$_{\gamma=0.9}$	85.67 ± 2.2	79.99 ± 0.7	78.45 ± 1.3	34.98 ± 0.5	19.76 ± 1.7	25.96 ± 1.2
MIR + GMED$_{\gamma=0.99}$	86.76 ± 1.2	79.76 ± 0.7	78.61 ± 0.6	35.78 ± 3.2	20.48 ± 1.7	27.70 ± 1.3
MIR + GMED$_{\gamma=1.0}$	86.52 ± 1.4	79.25 ± 0.8	79.08 ± 0.8	36.17 ± 2.5	21.22 ± 1.0	26.50 ± 1.3

D T-SNE Visualization

In Figure 6, we show the t-SNE [27] visualization of the editing vector $\Delta x = x_{\text{after}} - x_{\text{before}}$ for examples from first 2 tasks in Split MNIST. We note that the editing vectors cluster by the labels of the examples. It implies the editing performed is correlated with the labels and is clearly not random.

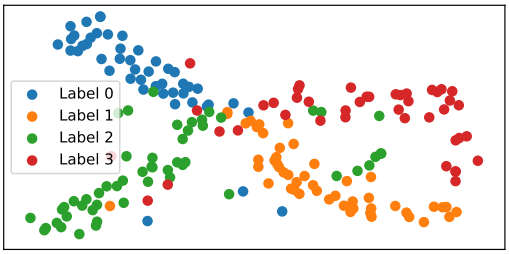

Figure 6: A t-SNE visualization of the editing performed on data examples. We use labels from the first two tasks in Split MNIST.

E Dataset Details

Following [2], we limit the number of training examples per task to 1,000 for all MNIST experiments, including Split MNIST, Permuted MNIST, and Rotated MNIST. The datasets consist of 5, 10, and 20 tasks respectively. In Split CIFAR-10, Split CIFAR-100, and Split mini-ImageNet, each task consists of 10,000, 2,500, and 2,500 training examples. For tuning hyperparameters, we separate out 5% of the training set examples in the first three tasks as the validation set. For Rotated MNIST, we limit the number of testing examples to 1,000 per task; while for other datasets, we use the complete test sets.

License and Links of Datasets. The MNIST dataset can be found in `http://yann.lecun.com/exdb/mnist/`. The dataset is released without a specific license. CIFAR-10 and CIFAR-100 could be found in `https://www.cs.toronto.edu/~kriz/cifar.html`. Similarly, the dataset is released without a license. Mini-ImageNet and its terms of use can be found in `https://mtl.yyliu.net/download/`.

Personally Identifiable Information or Offensive Content in datasets. Among all datasets we applied, MNIST is a hand-written digit classification dataset, while CIFAR and mini-ImageNet are image classification dataset over common objects. To the best of our knowledge, they do not contain any sensitive information.

F Comparison with ER+T

In addition to data augmentation over the replay memory (ER$_{aug}$), [5] also studied a number of other tricks, such as exponential LR decay and balanced reservoir sampling. The integrated method, referred to as ER with Tricks (ER+T), achieved SoTA performance in a task-aware, non-online setup over a number of datasets. We adapt ER+T to an online task-free continual learning setup by discarding the tricks not compatible with the online task-free scenario (namely the Bias Control (BiC) trick) and build GMED upon it. Table 8 summarizes the results. We find ER+T does not outperform ER$_{aug}$, *i.e.*, the tricks other than the data augmentation is not effective in an single-epoch online training setup, except Split MNIST. We further find ER+T+GMED outperforms or performs comparably with ER+T. The improvement is significant on Split MNIST and Split mini-ImageNet datasets.

Table 8: Building GMED over ER+T. * indicates significant improvement with p-value less than 0.05.

Methods / Datasets	Split MNIST	Permuted MNIST	Rotated MNIST	Split CIFAR-10	Split CIFAR-100	Split mini-ImageNet
ER + T	78.35 ± 4.5	77.71 ± 0.7	**80.05 ± 1.3**	**47.55 ± 2.6**	19.40 ± 1.5	31.25 ± 1.5
ER + T + GMED	**83.02*** **± 0.4**	**77.92 ± 0.3**	79.96 ± 0.2	47.39 ± 5.0	**19.75 ± 1.2**	**31.84*** **± 1.3**

Table 9: Performance of methods over data streams with fuzzy task boundaries over all six datasets. * indicates significant improvement with p-value less than 0.05.

Methods / Datasets	Split MNIST	Permuted MNIST	Rotated MNIST	Split CIFAR-10	Split CIFAR-100	Split mini-ImageNet
ER	79.74 ± 4.0	**78.98 ± 0.5**	76.45 ± 1.2	37.15 ± 1.6	**21.99 ± 1.1**	26.47 ± 2.3
ER + GMED	**82.73*** **± 2.6**	78.91 ± 0.5	**76.55 ± 1.0**	**40.57*** **± 1.7**	21.79 ± 1.9	**28.20*** **± 0.6**
MIR	85.80 ± 1.9	**79.31 ± 0.7**	77.04 ± 1.2	38.70 ± 1.7	21.57 ± 1.4	25.83 ± 1.5
MIR + GMED	**86.17 ± 1.7**	79.26 ± 0.8	**77.56*** **± 1.1**	**41.22*** **± 1.1**	**22.16 ± 1.1**	**26.86*** **± 0.7**
ER$_{aug}$	81.30 ± 2.0	**77.71 ± 0.8**	**80.31 ± 0.9**	47.97 ± 3.5	18.47 ± 2.0	31.75 ± 1.0
ER$_{aug}$ **+ GMED**	**82.39*** **± 3.7**	77.68 ± 0.8	80.30 ± 1.0	**51.38*** **± 2.2**	**18.63 ± 1.3**	**31.83 ± 0.8**

G Full Results of Experiments with Fuzzy Task Boundaries

Table 9 summarizes full results of experiments where we apply fuzzy boundaries between tasks. Experiments show that GMED generally improves performance on Split MNIST, Split CIFAR-10 and Split mini-ImageNet significantly when built upon ER, MIR, or ER$_{aug}$.

H Comparison to Task-Aware CL Methods

We additionally present the results of task-aware memory-based approaches in Table 10. We notice HAL was not as competitive as ER on Split MNIST, Split CIFAR-10, and Split mini-ImageNet that employs a class incremental learning setup — while in the original work, the approach was mainly test in a domain-incremental learning setup [41]. GEM performs competitively on two of the datasets (Split MNIST and Rotated MNIST), where GEM+GMED could further slightly improve the performance.

Table 10: **Performance of Task-Aware Approaches.**

Methods / Datasets	Split MNIST	Rotated MNIST	Split CIFAR-10	Split mini-ImageNet
HAL	77.92 ± 4.2	78.48 ± 1.5	32.06 ± 1.5	21.18 ± 2.1
GEM	87.21 ± 1.3	78.40 ± 0.5	14.81 ± 0.4	5.92 ± 0.6
GEM + GMED	87.69 ± 1.4	78.62 ± 0.4	14.13 ± 0.3	5.81 ± 0.5

I GMED without Writing Edits Back

Table 11 summarizes the result of discarding the editing performed after each time step: in these experiments, we replay the edited examples, but do not replace original examples in the memory with edited ones. The approach is noted as ER+GMED w/o writeback. We tune a separate set of editing stride and regularization strength for the method. The results indicate that ER+GMED w/o writeback achieves slightly lower performance on 5 out of 6 datasets (except Split CIFAR-10) compared to ER+GMED.

Table 11: GMED without storing back edited examples, *i.e.*, the algorithm replays edited examples, but does not update the original examples in the memory as edited ones. The results are shown after ER + GMED w/o writeback.

Methods / Datasets	Split MNIST	Permuted MNIST	Rotated MNIST	Split CIFAR-10	Split CIFAR-100	Split mini-ImageNet
ER	81.07 ± 2.5	78.85 ± 0.7	76.71 ± 1.6	33.30 ± 3.9	20.41 ± 1.2	25.92 ± 1.2
ER + GMED	82.67 ± 1.9	78.86 ± 0.7	77.09 ± 1.3	34.84 ± 2.2	20.93 ± 1.6	27.27 ± 1.8
ER + GMED w/o writeback	81.18 ± 2.6	78.49 ± 0.7	76.88 ± 1.1	34.86 ± 2.7	20.86 ± 1.6	27.20 ± 1.8

J Applying the Full Dataset in Split MNIST

In our main experiments, we sampled 1,000 training examples for Split MNIST following [3]. We further include the results of using the entire dataset for training in Table 12, and still see significant improvements.

Table 12: **Performance on Split MNIST constructed from the entire MNIST training set**. Note than in our main experiments, we sampled 1,000 examples per task. * and ** indicates significant improvement with $p < 0.1$ and $p < 0.01$ respectively.

Methods	w/o GMED	w/ GMED
ER	86.61 ± 1.3	**88.48**** $\pm$ **1.0**
MIR	89.18 ± 1.5	**89.88**** $\pm$ **1.1**
ER$_{aug}$	91.52 ± 1.5	**92.30*** $\pm$ **0.9**

K Tabular Results of Model Performance under Various Memory Sizes

Figure 2 in the main paper present model performance under various memory size setups with bar charts. In Table 13, we further summarize results in tables, showing exact numbers of mean and standard deviation of performance. We find the improvements on Split-MNIST and Split mini-ImageNet are significant-with <0.05 over all memory size setups. Improvements on Rotated MNIST and Split CIFAR-10 are also mostly significant.

Table 13: **Performance of ER, GMED+ER, MIR, and GMED+MIR with various memory sizes**, shown in tables. * indicates significant improvement with $p < 0.05$.

Dataset	**Split MNIST**			**Rotated MNIST**			**Split CIFAR-10**		
Mem size	100	200	500	100	200	500	100	200	500
ER	71.13 ± 1.5	77.85 ± 1.4	81.07 ± 2.5	63.35 ± 1.7	70.07 ± 1.2	76.71 ± 1.6	21.94 ± 1.5	26.03 ± 2.1	33.30 ± 3.9
ER+GMED	$73.90^* \pm 3.7$	$79.19^* \pm 1.6$	$82.67^* \pm 1.9$	$64.23^* \pm 1.5$	$70.71^* \pm 1.2$	77.09 ± 1.3	$22.89^* \pm 1.8$	$27.18^* \pm 2.0$	$34.84^* \pm 2.2$
MIR	73.53 ± 1.6	80.79 ± 2.5	85.72 ± 1.2	62.91 ± 1.1	69.95 ± 1.2	77.50 ± 1.6	23.09 ± 1.2	28.78 ± 2.0	34.42 ± 2.4
MIR+GMED	$77.09^* \pm 2.5$	$83.52^* \pm 1.1$	$86.52^* \pm 1.4$	$64.39^* \pm 0.4$	$70.62^* \pm 1.2$	$79.08^* \pm 0.8$	$25.87^* \pm 1.9$	28.89 ± 1.8	36.17 ± 2.5
Dataset	**Split CIFAR-100**			**Split mini-ImageNet**					
Mem size	2000	5000	10000	5000	10000	20000			
ER	12.31 ± 1.0	16.86 ± 0.4	20.41 ± 1.2	18.92 ± 0.9	25.92 ± 1.2	29.93 ± 1.9			
ER+GMED	12.65 ± 0.8	$17.40^* \pm 0.7$	20.93 ± 1.6	$21.36^* \pm 0.4$	$27.27^* \pm 1.8$	$30.60^* \pm 1.8$			
MIR	11.85 ± 0.8	18.36 ± 1.5	20.02 ± 1.7	17.47 ± 1.0	25.21 ± 2.2	30.08 ± 1.2			
MIR+GMED	12.32 ± 0.8	18.60 ± 1.1	$21.22^* \pm 1.0$	$20.31^* \pm 0.8$	$26.50^* \pm 1.3$	$31.33^* \pm 1.6$			

L Performance of Optimal Editing

Table 14 summarizes the performance of optimal editing (ER+Optimal) discussed in Sec. 4.4 compared to ER and ER+GMED. We notice that ER+Optimal outperforms ER+GMED on Split MNIST and Rotated MNIST, slightly improves on Split CIFAR-10, but does not improve on Split mini-ImageNet. The hyperparameters are not tuned extensively on mini-ImageNet because optimal editing is very expensive to compute, requiring to compute forgetting over a large sample of early examples every training step, even on moderately-sized datasets.

We hereby note that our ER+Optimal does not necessarily achieve an upper-bound performance of memory editing, as it computes optimal edit for only the coming *one* training step over the data stream. Because the edited examples may in turn affect training dynamics in future training steps, it is hard to derive an exact upper bound performance of memory editing.

M Editing Extra Examples

Table 15 summarizes the results when we edit an additional number of examples per task. We randomly sample additional examples from the memory, perform edits, and writeback, without replaying them. We notice that, over all datasets, replaying a small number of extra examples improve the performance. However, the performance drops when too many examples are edited. We hypothesize replaying additional examples per step has a similar effect to increasing the stride of

Table 14: Performance of optimal edits in Sec. 4.4 and their comparison to ER and GMED.

Methods/Datasets	**Split MNIST**	**Rotated MNIST**	**Split CIFAR-10**	**Split mini-ImageNet**
ER	80.14±3.2	76.71±1.6	33.30±3.9	25.92±1.2
ER+GMED	82.67±1.9	77.09±1.3	34.84±2.2	27.27±1.8
ER+Optimal Editing	83.40±2.6	77.73±1.3	35.04±2.6	27.01±1.6

Table 15: Results when editing an additional number of examples per step in ER+GMED.

Methods/Dataset	Split MNIST	Rotated MNIST	Split CIFAR-10	Split mini-ImageNet
ER+GMED (10 examples)	82.67±1.9	77.09±1.3	34.84±2.2	27.27±1.8
+ 3 examples	83.00±2.3	77.01±1.6	34.87±2.9	27.36±1.9
+ 5 examples	82.78±2.3	76.98±1.6	35.31±2.2	28.00±2.0
+ 10 examples	82.21±2.2	76.51±1.4	34.86±2.8	27.17±1.8
+ 50 examples	81.42±2.6	76.50±1.3	31.79±2.6	27.10±1.9

Table 16: Prediction change rate of edited examples compared to the corresponding original examples.

Dataset	Split MNIST	Rotated MNIST	Split CIFAR-10	Split mini-ImageNet
Prediction Change Rate	10.2%	2.4%	5.1%	5.5%

edits performed per example. As such, when required, the number of additional examples to edit can be an additional tunable hyperparameter in our approach.

N Prediction Change Rate of Edited Examples

We show the percentage of changed predictions after example edits in Table 16. Specifically, when the training completes, we classify examples stored in the memory (which have experienced editing) and the corresponding original examples with the model. We then compute "prediction change rate" as the portion of changed predictions over all examples stored in the memory. The prediction change implies the edited examples are more adversarial, or they can be simply artifacts to the model. However, we notice that such prediction change rate is positively correlated with performance improvement of GMED over four datasets. It implies such adversarial or artifact examples are still beneficial to reducing forgetting.

O Building Upon Other Memory Population Strategies

In all our main experiments, we used reservoir sampling to populate the replay memory. We further experiment with the greedy variant of Gradient-based Sample Selection strategy (GSS-Greedy) [4] on feasibly-sized datasets, which tries to populate the memory with more diverse examples. We report the results in Table 17. The results show GMED could still bring modest improvements when built upon GSS-Greedy.

P Ethics Statement: Societal Impact

In this work, we extend existing memory-based continual learning methods to allow gradient-based editing. We briefly discuss some of the societal impacts in this section.

Societal Impact (Positive): Improvement in Continual learning, and in particular, the online version of continual learning, can lead to potential benefits in wide variety of machine-learning tasks dealing with streaming data. With novel information being introduced by the day, it is imperative that models are not re-trained on the entire data, instead adapt and take into account the streaming data without forgoing previously learned knowledge. As an example, BERT trained on 2018 language data would be less useful for current events, but still be useful for commonsense knowledge information and as such it would be extremely beneficial if one could train over streaming current events while retaining other knowledge.

Towards this goal, a primary advantage of GMED is its integration with other memory-based continual learning frameworks. As a result, we expect advances in memory-based methods to be complemented

Table 17: Prediction change rate of edited examples compared to the corresponding original examples.

Method/Dataset	Split MNIST	Rotated MNIST
ER+GSS-Greedy	83.70±1.0	73.80±1.6
ER+GSS-Greedy+GMED	84.62±1.2	74.47±1.3
p-value	0.016	0.18

by GMED, incurring only a minor computational overhead – a key requirement for deploying any online continual learning algorithm in the wild.

Societal Impact (Negative): Caution must be taken in deploying our continual learning algorithms in the wild. This is because, CL algorithms at present, are validated solely on small datasets with synthetically curated streams. In the wild, the examples in the continual stream can have unexpected correlations which are not apparent in image-classification only streams.

Another key issue with continual learning is that once a knowledge is learned it is difficult to know whether it has been completely unlearned or still retained in the neural network which can be probed later. This could happen for say hospital records where patient confidentiality is needed, but were used by the continual learning model. Deleting such records is non-trivial for usual machine learning models but have dire consequences in the continual learning domain where the original stream can no longer be accessed.