Appendix

A. Limitations and Future Work

In the main paper, we proposed a novel evaluation paradigm for real-time online continual learning, aimed at simulating real-world scenarios as closely as possible. However, we recognize that there are few limitations that can be addressed in future work. Although *ER* and *ER++* achieved the highest Average Online Accuracy among all other considered methods, their performance only reached up to 20% and 35%, respectively. This low performance suggests that there is room for improvement with better OCL methods. Furthermore, ER may have had an advantage in this setup due to potential temporal correlations in the CLOC dataset. One possible solution is to design new OCL datasets that have less temporal dependency. Additionally, while ER demonstrated the best online adaptation among other methods, it performed slightly worse in information retention (see backward transfer in Appendix D). Future work could explore OCL methods that can optimize both online adaptation and information retention.

B. Pre-training Free Online Continual Learning

In real world-settings, continual learning proceeds a pre-trained model on locally annotated available data; as reported in all previous experiments in the main paper. However, we consider in this section a setting where models are continuously trained without any pre-training. We repeat our experiments on the CLOC dataset while training the ResNet50 backbone from scratch. We find that in this setting, a learning rate of 5×10^{-2} works best for all methods except PoLRS, which works better with a learning rate of 10^{-2}.

Following the evaluation grouping we use in the main paper, we present the results in three figures: Fast Stream (Figure 1), Fast Stream with Data Normalization (Figure 2), and Slow Stream (Figure 3). We note that the results without pre-training reveal a similar conclusion to the main paper, where the simple baseline (*ER*) outperforms more expensive OCL specific methods. Notably, ACE and *ER* are as efficient as each other, and they both consistently outperform the more expensive methods. Interestingly, ACE only very marginally outperforms *ER* despite that *ER* does not use any specific OCL technique.

C. Small-Scale Experiments

In this section, we test whether previous methods work well under real-time evaluation of continual learning if the dataset used is unrealistically small. To do this, we repeat the fast stream and slow stream experiments on CIFAR10 and CIFAR100 datasets, which many of the considered methods were originally tested on. We randomly shuffle these datasets without task boundaries constructing an online stream and sequentially train models over the stream in a single pass.

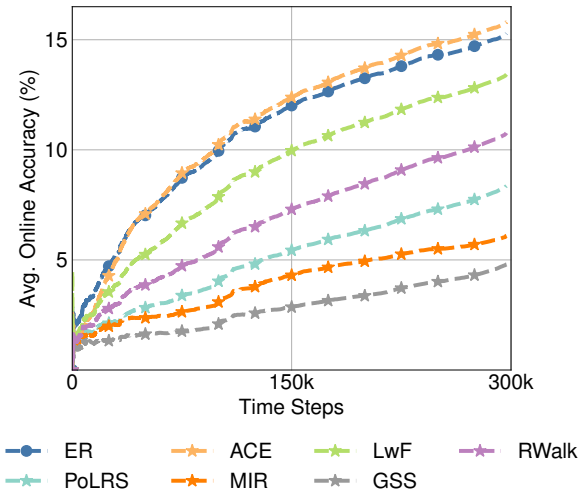

Figure 1. **Fast Stream Evaluation (Pre-training Free).** Consistent with the main paper experiments, the most efficient methods, *ER* and ACE, significantly outperform the other considered methods, even when the backbone is trained from scratch.

Implementation Details. For all experiments on CIFAR10 and CIFAR100, the stream reveals 10 images per time step which are augmented with another 10 images sampled from a memory. This procedure creates a training minibatch of 20 samples per time step. Moreover, we set the memory size to 100. All models use a reduced ResNet18 as a backbone [6] where we use SGD for optimizing models. For learning rate selection, we cross-validate on a held-out 5% of each dataset. In CIFAR10, we find the following learning rate values to work best:

- 10^{-3} for ER, ACE, LwF and RWalk
- 5×10^{-3} for PoLRS, MIR, and GSS

As for CIFAR100, we use these learning rate values:

- 10^{-3} for ER, ACE, PoLRS
- 5×10^{-3} for LwF, RWalk, GSS
- 10^{-2} for MIR

C.1. Fast Stream

Recall that, for ease of comparison, we assume that *ER* has a computational complexity that matches the stream speed and, thus, is able to train on all incoming samples in the fast stream. As a result, more computationally expensive methods than *ER* lag behind the stream. This delay

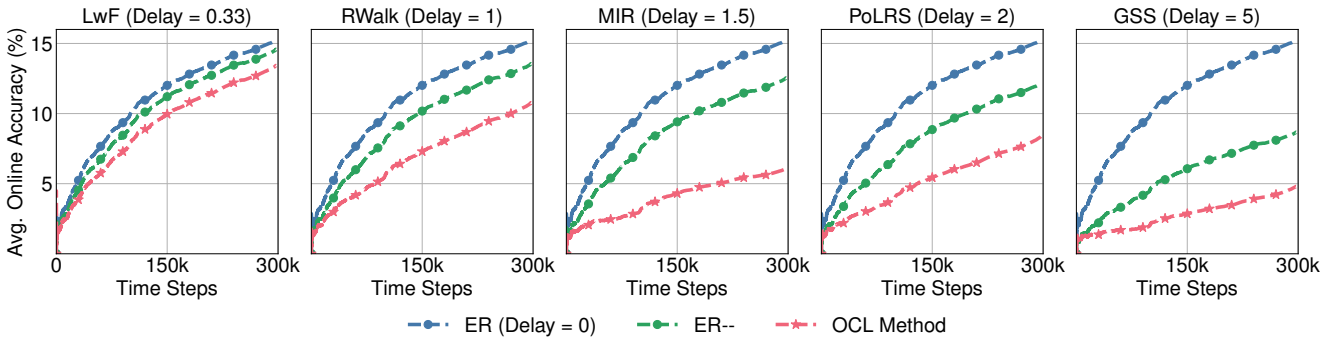

Figure 2. **Fast Stream - Training Data Normalization (Pre-training Free)**. We compare each method against *ER* and its delayed version *ER*--. *ER*-- performs extra gradient steps per time step to match the delay of the compared-against method, so it trains on fewer samples than *ER*. We highlight that even when no pre-training is deployed, *ER*-- outperforms all considered methods.

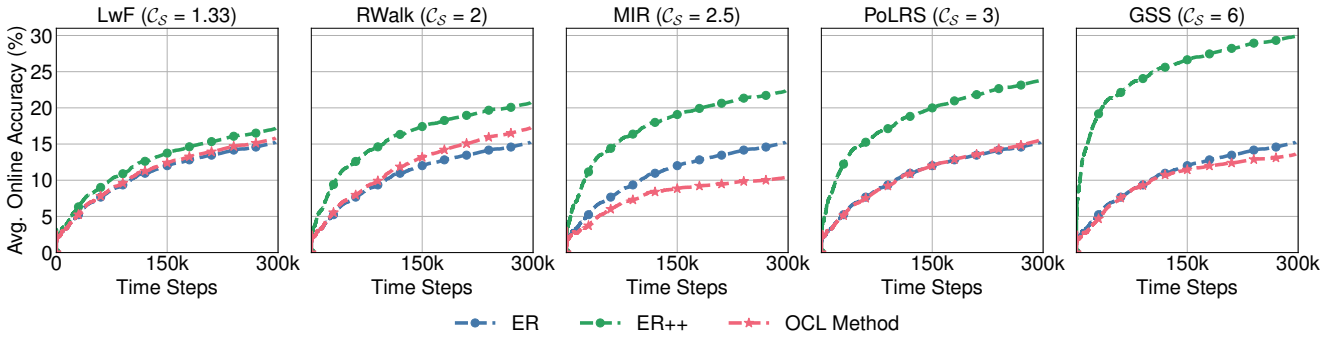

Figure 3. **Slow Stream Evaluation (Pre-training Free)**. We compare each method against *ER* and *ER*++, which performs extra gradient steps per time step to match the complexity $\mathcal{C}_S$ of the compared-against method. *ER*++ outperforms all the considered methods, even when the backbone is not pre-trained.

forces these methods to skip some incoming samples, where they effectively train on only a subset of the stream samples. In Table 1, we report the performance of considered OCL methods when evaluated on CIFAR10 and CIFAR100 datasets given a fast stream. We note that even when using small-scale datasets, ACE and *ER* outperform all other considered methods in the practical, fast stream setting. Remarkably, ACE performance is only marginally better than *ER* by less than 1.5% on both datasets even though *ER* does not do anything specific for continual learning.

Method	Average Online Accuracy (%)	
	CIFAR10	CIFAR100
ER	51.60	13.62
ACE	**53.05**	**14.42**
LwF	49.65	12.41
RWalk	43.90	9.36
PoLRS	41.25	6.92
MIR	43.55	10.57
GSS	33.67	6.56

Table 1. **Small-Scale Experiments - Fast Stream.** We repeat the fast stream experiments on the small-scale datasets CIFAR10 and CIFAR100. We notice that regardless of the size of the dataset, ACE and *ER* outperforms all the considered methods.

C.2. Slow Stream

In this setting, similar to the main paper, the slow stream is as slow as the more expensive OCL method. That is to say, methods in this stream are able to train on all stream samples. Also recall that since methods have different training complexities in this stream, we use the modified baseline *ER*++ for a fair comparison. *ER*++ matches the complexity of each compared-against method by performing additional gradient updates.

We show the results of the slow stream evaluation in Table 2 when considering small-scale datasets such as CIFAR10 and CIFAR100. Similar to the slow stream results on the CLOC dataset, *ER*++ outperforms all considered OCL methods when evaluated on CIFAR10. As for CIFAR100, *ER*++ also outperforms all considered methods except LwF and MIR, which are only slightly better than *ER*++ by no more than 0.36%.

C_S	Method	Average Online Accuracy (%)	
		CIFAR10	CIFAR100
4/3	LwF	52.48	15.14
	ER++	54.09	14.79
2	RWalk	51.17	14.56
	ER++	56.71	16.35
5/2	MIR	53.15	17.73
	ER++	58.03	17.37
3	PoLRS	49.51	12.41
	ER++	59.52	18.45
6	GSS	53.90	16.45
	ER++	59.76	17.09

Table 2. **Small-Scale Experiments - Slow Stream.** We repeat the slow stream experiments on the small-scale datasets CIFAR10 and CIFAR100. We note that in almost all cases, *ER++* outperforms the compared-against method. The only exceptions are LwF and MIR, and only when evaluated on CIFAR100. Even then, they have marginal gains over their corresponding simple baseline *ER++*, which indicates the need to develop better and more efficient methods for online continual learning.

D. Performance on Held-out Samples

In the main paper, we focused on the Average Online Accuracy as the main evaluation metric since it measures the adaptation to the next training batch (*i.e*. near future distribution), which is a highly desirable property in real-time streams with quick distribution changes. In this section, we consider two other evaluation metrics for continual learning, namely backward transfer and forward transfer. The backward transfer metric measures forgetting on past distributions, while the forward transfer evaluates models' adaptation to future distributions on held-out samples. It is important to note that forward transfer differs from Average Online Accuracy in that it evaluates the performance on held-out samples instead of future unseen training samples. Additionally, it monitors the performance on distributions that are far into the future, rather than the distribution of the next batch only.

To measure backward transfer and forward transfer, we uniformly sample and hold-out 1% of the CLOC dataset from different time steps. The backward transfer of the final model obtained at the end of the stream steps is measured by computing the average accuracy on the unseen samples from past timesteps at every time step. Similarly, the forward transfer is evaluated by measuring the average accuracy on the unseen samples starting from the timestamp at which the model was obtained to the end of the stream. Following CLOC, we measure the forward transfer for the model saved at 1/3 and 2/3 of the stream.

Figures 4 and 5 show the backward and forward transfer results in the slow and fast streams, respectively. We observe that all OCL methods perform similarly to the *ER* method, with only a slight advantage for some methods. For example, in the slow stream, although GSS and MIR require 6 and 2.5 times the computational cost of *ER*, respectively, they only outperform *ER* in backward transfer by 1.5% and 0.7%, respectively. Similarly, in the slow stream, they outperform *ER* in forward transfer (at 1/3 of the stream) by only 0.4% and 0.5%, respectively. In the fast stream results shown in Figure 4, MIR and GSS also slightly outperformed *ER*, even though they suffered from some training delay, indicating that they have a slight advantage in the forward/backward transfer metrics that they were optimized for. Importantly, this slight increase in GSS, and MIR performance comes at the cost of lower Average Online Accuracy, which is crucial in real-time continual learning. When considering Average Online Accuracy on both the slow stream and fast stream, the role of training efficiency becomes apparent.

It is worth noting that the backward transfer evaluation metric is slightly different from the objective of real-time online continual learning. In offline continual learning, methods are typically evaluated at the end of the stream on a held-out test set. However, this evaluation setup is irrelevant to real-time online continual learning, where streams constantly undergo rapid distribution shifts. Due to this stream dynamic, performance on samples from previous distributions is not a good indicator of whether models will make correct predictions on new incoming samples. This is because, in OCL with a naturally distributed stream, only the current stream samples are relevant for evaluation. For example, consider a naturally distributed stream revealing images of electronic devices. It is less likely for a stream to reveal images of 1995 devices as opposed to images of devices from 2022. This is the classical motivation behind online continual learning from a naturally distributed stream as opposed to offline continual learning.

As for the forward transfer metric, we note that it may not be an accurate indicator of model's ability to predict future samples, especially in streams with rapidly changing distributions. Since forward transfer is calculated as a running average of accuracy over future distributions, it assigns equal weights to all future samples. However, this might be misleading in dynamic, real-time streams where near future distributions are more likely to be observed than those in the far future. In other words, a model that performs well on forward transfer may still struggle to adapt to the near future distribution. Therefore, while forward transfer can be a helpful metric in assessing a model's adaptability to future distributions, it should be interpreted with caution in real-time streams and supplemented with other evaluation metrics that focus on the near future distribution, such as the Average Online Accuracy.

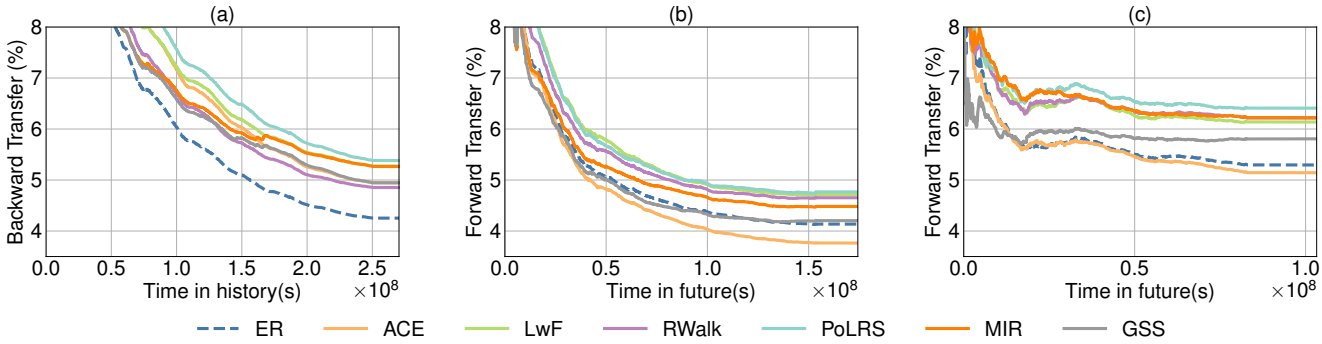

Figure 4. **Performance on Held-out Samples - Fast Stream**. (a) Backward transfer for a model obtained at the end of the stream. (b, c) Forward transfer for a model obtained at 1/3 and 2/3 of the stream respectively. We note that all methods have comparable performance to *ER* in both backward and forward transfer metrics with only a maximum gap of 1.1% across all plots.

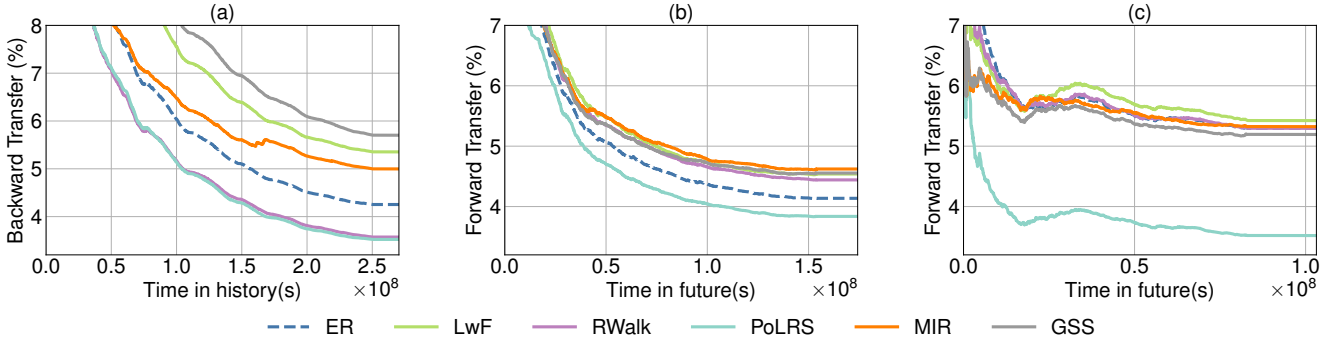

Figure 5. **Performance on Held-out Samples - Slow Stream**. (a) Backward transfer for a model obtained at the end of the stream. (b, c) Forward transfer for a model obtained at 1/3 and 2/3 of the stream respectively. We observe that all methods have comparable performance to *ER* in both backward and forward transfer metrics. Even the most computationally intensive methods, namely GSS, and MIR, only outperform *ER* in backward/forward transfer by a maximum of 1.5%, which are the metrics they were optimized for. However, this minor advantage comes at the cost of much lower Average Online Accuracy. Note the ACE is not included in the plots because it has a training complexity of 1, and therefore, it was not evaluated in the slow stream.

E. A Faster Stream

In the fast stream experiments reported in the main manuscript, we assumed that *ER* is as fast as the stream for ease of comparison. As a result, *ER* had a stream-model relative complexity of $\mathcal{C}_{\mathcal{S}}(ER) = 1$, allowing it to train on the entire data stream. In contrast, methods with higher computational complexity than *ER* had a stream-model relative complexity larger than 1, leading to some training delay. One could argue that our setup gave *ER* an advantage by normalizing the stream speed to it. In this section, we explore a setting where the stream is faster than all evaluated methods.

We repeat the fast stream experiments but with a stream speed that is twice as fast as *ER*. Under this new stream speed, we outline in Table 3 the corresponding training complexity and delay for each considered method. In Figure 6, we report the performance of considered OCL methods in the faster stream. Even when the stream is faster than *ER*, we observe similar findings to the main manuscript. In particular, *ER* still outperforms all other considered methods.

CL Strategy	**Method**$(\mathcal{A})$	$\mathcal{C}_{\mathcal{S}}(\mathcal{A})$	**Delay**
Experience Replay	ER	2	1
Regularization	ACE	2	1
	LwF	8/3	3/2*
	RWalk	4	3
LR Scheduler	PoLRS	6	5
Sampling Strategies	MIR	5	4
	GSS	12	11

Table 3. **A Faster Stream Evaluation - Training complexity and delay of considered OCL methods when the stream speed is twice as fast as *ER*.** *Note that we rounded down the complexity of LwF from 1.67 to 1.5 for ease of implementation.

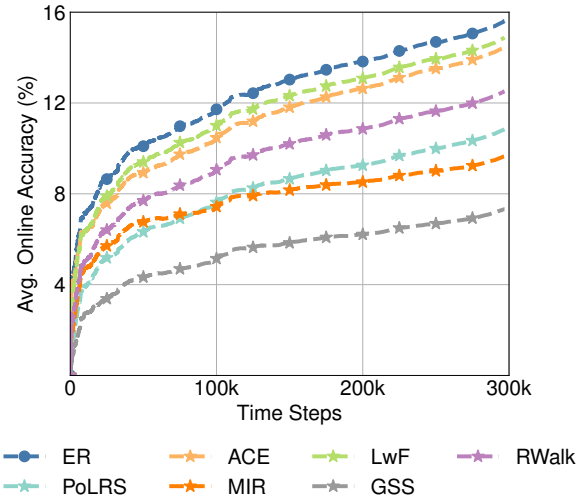

Figure 6. **A Faster Stream Evaluation.** We set the stream speed to twice as fast as *ER*. Even when the stream speed is faster than *ER*, *ER* still outperforms all other considered methods, similar to the main manuscript experiments.

F. A Slower Stream

In the slow stream experiments presented in the main paper, we demonstrate that *ER*++ outperforms all compared OCL methods. These findings raise the following question: *What if the stream is slowed down to the point where more computationally expensive methods can perform additional gradient steps?* To explore this question, we allow each method to perform two Gradient Descent (GD) steps per time step. Accordingly, we adjust *ER*++ to match the complexity of the boosted version of each method shown in Table 4. Note that performing two GD steps on LwF, RWalk, and PoLRS results in doubling their original complexity. In contrast, making an additional GD step on MIR and GSS only increases their complexity by one. This is because LwF, RWalk, and PoLRS require additional backward and/or forward passes with each GD step, while the sampling-based approaches, MIR and GSS, only require them with each new incoming data.

In Figure 7, we report the performance of each OCL method with one and two GD steps per time step, where the results for one GD step are shown for reference only. We compare the boosted version of each method against *ER* and its corresponding *ER*++. Consistent with the slow stream results presented in the main paper, *ER*++ outperforms all the considered OCL methods. Compared to other methods, LwF has the smallest accuracy gap to *ER*++, suggesting that it is a highly efficient method. Although adding an extra GD step to GSS and MIR has reduced the previous gap to *ER*++ from 17% to 14% and 11% to 10%, respectively, both methods lag behind LwF, which is a less computationally expensive method. In summary, the slower stream results indicate that *ER*++ remains the optimal choice for achieving the highest Average Online Accuracy while using computational resources efficiently.

CL Strategy	**Method**($\mathcal{A}$)	$\mathcal{C}_{\mathcal{S}}(\mathcal{A})$
Regularization	LwF	5½*
	RWalk	4
LR Scheduler	PoLRS	6
Sampling Strategies	MIR	7½*
	GSS	7*

Table 4. **A Slower Stream Evaluation - Training complexity of considered OCL methods when each method is performing two GD steps per time step.** *Note that complexities of these methods were rounded down for ease of implementation.

LR	0.001	0.005	0.01	0.05
ER	8.2	**10.9**	9.7	4.4
ACE	8.1	**8.8**	7.9	3.4
LwF	8.3	**11.1**	10.0	4.3
RWalk	8.1	**11.0**	10.3	4.4
PoLRS	**10.8**	10.6	9.3	3.9
MIR	8.0	**10.8**	8.5	3.1
GSS	7.5	**9.8**	9.0	3.6

Table 5. **Learning Rate Grid Search**. We perform cross-validation on each of the OCL methods considered, testing them on four different learning rate values. The Average Online Accuracy of the chosen learning rate value is denoted in bold for each method.

G. Learning Rate Grid Search

In order to determine the best learning rate for the methods in the slow and fast streams discussed in the main manuscript, we conduct a gird search across 4 values: $\{0.001, 0.005, 0.01, 0.05\}$, which we report in Table 5. We note that in practice, the stream speed is often unknown during training. Since the training delay is determined by the unknown stream speed, we conduct the grid search while assuming no training delay for all methods.

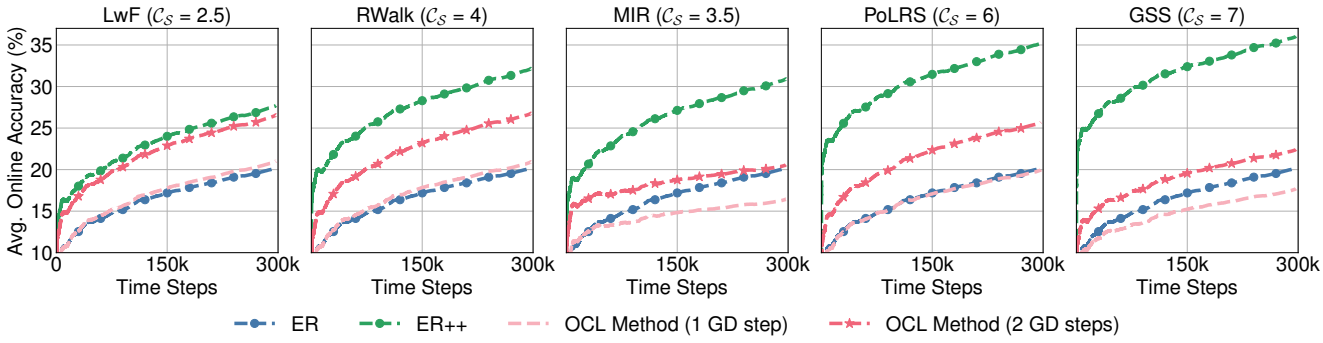

Figure 7. **A Slower Stream Evaluation**. We enable each OCL method to perform two GD steps per time step. We compare each method against *ER* and *ER++*, which performs extra GD steps per time step to match the complexity $\mathcal{C}_S$ of the compared-against method. For reference, we also show the performance of each OCL method with only a single GD step. Even when OCL methods perform additional GD steps, *ER++* still outperforms them all.