Continual learning autoencoder training for a particle-in-cell simulation via streaming

Patrick Stiller
Institute for Radiation
Helmholtz-Zentrum Dresden-Rossendorf
p.stiller@hzdr.de

Varun Makdani
Institute for Radiation
Helmholtz-Zentrum Dresden-Rossendorf
v.makdani@hzdr.de

Franz Pöschel
Center of Advanced Systems Understanding Görlitz
Helmholtz-Zentrum Dresden-Rossendorf
f.poeschel@hzdr.de

Richard Pausch
Institute for Radiation
Helmholtz-Zentrum Dresden-Rossendorf
r.pausch@hzdr.de

Alexander Debus
Institute for Radiation
Helmholtz-Zenrum Dresden-Rossendorf
a.debus@hzdr.de

Michael Bussmann
Center of Advanced Systems Understanding Görlitz
Helmholtz-Zenrum Dresden-Rossendorf
m.bussmann@hzdr.de

Nico Hoffmann
Institute for Radiation
Helmholtz-Zentrum Dresden-Rossendorf
n.hoffmann@hzdr.de

Abstract

The upcoming exascale era will provide a new generation of physics simulations. These simulations will have a high spatiotemporal resolution, which will impact the training of machine learning models since storing a high amount of simulation data on disk is nearly impossible. Therefore, we need to rethink the training of machine learning models for simulations for the upcoming exascale era. This work presents an approach that trains a neural network concurrently to a running simulation without storing data on a disk. The training pipeline accesses the training data by in-memory streaming. Furthermore, we apply methods from the domain of continual learning to enhance the generalization of the model. We tested our pipeline on the training of a 3d autoencoder trained concurrently to laser wakefield acceleration particle-in-cell simulation. Furthermore, we experimented with various continual learning methods and their effect on the generalization.

1 Introduction

Numerical simulations of complex systems such as Laser-Plasma acceleration are computationally costly and run on the largest HPC systems in the world. Neural Network based surrogate models of these simulations drastically speeds up the analysis due to fast inference times promising in situ analysis of experimental data. Such surrogate models can be applied for forward and inverse problems various fields of science, such as quantum physics [1], medical science [2], and plasma physics [3]. High-fidelity simulations in the upcoming exascale era can provide high-resolution training data for surrogate models. However, providing a training dataset stored on a disk is nearly impossible for these simulations. Therefore, we need to rethink the training of surrogate models to tackle the memory- and space limitations of current HPC systems. We conducted in-memory coupling to access the training data sequentially without storing them on disk. In-memory coupling allows streaming simulation data from multiple compute nodes at network speeds. However, in such a streaming setup, data, once consumed, cannot be regained. In such a learning environment, avoiding catastrophic forgetting is a big challenge. In this work, we showcase the prototype of this novel training pipeline. We applied techniques from continual learning into our streaming training pipeline to foster the generalization of the trained model and avoid catastrophic forgetting. We have experimented with various regularization and replay-based methods to avoid forgetting in a continuous learning regime. Such as A-GEM [4] and Online EWC [5]. We tested our pipeline on time-dependent 3D-electric field data generated from an LWFA (laser wakefield acceleration) simulation on PIConGPU [6]. Additionally, we extended the A-GEM algorithm for autoencoder training to reduce the memory consumption of this promising approach. This framework for continual learning is developed in an extensible manner, enabling its ability to be applied to model other simulation data. The paper is structured as follows. In section 2, we describe our conducted method. We discuss the results, the effect of different continual learning methods in our pipeline, and their memory consumption in section 3. Section 4 contains the concluding remarks.

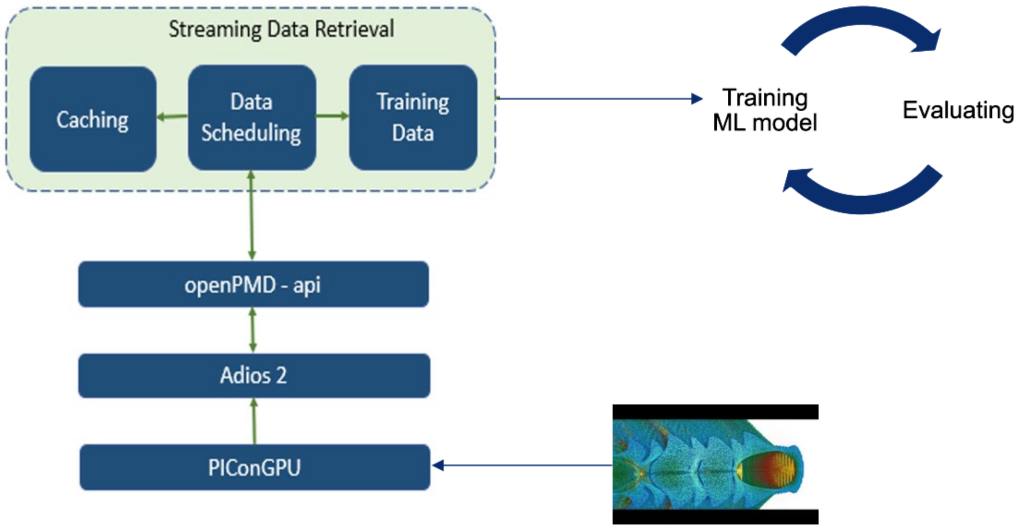

Figure 1: Illustration of the training pipeline. We access our electrical field data from a PIConGPU simulation via the Adios2 Backend of the openPMD-api. Additionally, we added a data scheduling unit to prevent the simulation from stopping

2 Method

We developed a training pipeline that allows in-memory access via streaming to the simulation data while the simulation is running (Figure 1). The Adios2 library allows access to the simulation data via in-memory streaming. Our OpenPMD unit converts the streamed Adios2 data into an OpenPMD data frame which allows our training engine to access the training data. The training engine consists of a data scheduler, which stops and starts the simulation and stores intermediate results in a cache. Caching the data reduces the number of stops in the simulation. The neural network uses the full cache for the next training iteration.

The disadvantage of this pipeline is that the trained model cannot revisit simulation data once consumed. Therefore, we applied continual learning techniques to improve the generalization and robustness of the trained model. We performed a 3d autoencoder training to learn a low-dimensional representation of the underlying simulation data and apply it for subsequent simulations. We have proposed a method, inspired by the replay-based method A-GEM[4], that significantly reduces the memory requirement for autoencoder training. The basic idea of the A-GEM algorithm is a projection of the current gradient g to a reference gradient g_{ref} to avoid catastrophic forgetting. The reference gradient is computed with respect to input data m, which is sampled from episodic memory $\mathcal{M}$ (see equation 1). The projection only will be applied if $\tilde{g}^{\top} g_{ref} \geq 0$ is fulfilled.

$$\tilde{g} = g - \frac{g^{\mathsf{T}} g_{\mathrm{ref}}}{g_{\mathrm{ref}}^{\mathsf{T}} g_{\mathrm{ref}}} g_{\mathrm{ref}} \tag{1}$$

However, applying an episodic memory will be too expensive for high-fidelity simulation data. Therefore, we conducted a method that exploits the low dimensionality of the latent space of the autoencoder and stores them instead in the episodic memory. Hence it allows using A-GEM for such massive data as experienced in the use-case of LWFA simulation, especially for the upcoming exascale era. The loss for reference-gradient (g_{ref}) can then be calculated based on the change in the reconstruction of latent space by randomly sampling encoded reference memory from memory replay and then passing it through the decoder and then again with the encoder.

$$g_{ref} = \nabla_{\theta} \| f_E(f_D(e)) - e \|_2^2 \quad \text{where } e \subset \cup_{k<t} \mathcal{E}_k \tag{2}$$

here $f_E(f_D(e))$ is the encoder output on input data e sampled from replay memory ($\mathcal{E}$) containing samples of learned encoded latent space representations of previously learned tasks. Additionally, we adapted the method for convolutional autoencoders. Every convolutional layer extracts different features using different filters and on various output channels. In the case of A-GEM, layers with a large magnitude of gradients influence the dot product for the condition of parameter constraint ($\tilde{g}^{\top} g_{ref} \geq 0$), even the layers which do not require change are affected by other layers, which intuitively may increase loss on the objective. This is the basis of the second modification, where the condition is checked layer-wise, and the gradients are projected layer-wise as well.

3 Results

We trained a 3d convolutional autoencoder on the electrical field which is streamed sequentially from a PIConGPU [6] LWFA simulation via the Adios2-backend [[7]] of OpenPMD [8]. Each simulation step denotes a training task for the autoencoder. The proposed neural network architecture is inspired by [9] and [10] reduced order models for fluid simulations and shadowgraphy. We integrated the Online-EWC [5] and the A-GEM algorithm [4] into our training pipeline to compare their effect on the generalization. We used an episodic memory size of 10 for A-GEM and our proposed method. The training and the simulations run on a Tesla V100. We trained 40 epochs per streamed electrical field and used the Adam optimizer [11].

Table 1 compares the post-training performance of Online-EWC, A-GEM, and our proposed method. We used the average L1-Norm to measure the pixel-wise similarity. Furthermore, we used the SSIM norm [12] to measure structural similarity. Additionally, we performed basic training to quantify the effect of the continual learning techniques. Figure 2 visualizes the electrical fields after a full training beside a simulation. We observed that the autoencoder memorizes, as expected, only the most recent electrical field if he got trained without catastrophic forgetting protection. The Online-EWC algorithm learns a mixture of seen states which has no improvement in reconstruction quality. The A-GEM algorithm performs the best in terms of reconstruction quality (pixel-wise and structural similarity), followed by our proposed method. However, all methods cannot reconstruct the high-oscillating patterns in the electrical field data. We need more research to make the autoencoder robust for that pattern.

Table 1 additionally compares the memory overhead for the proposed methods. Since A-GEM stores ten electric fields simultaneously, we measured the highest additional memory consumption. The A-GEM method would not be feasible for use cases with higher resolution of the electric fields. The additional memory usage of the online EWC highly depends on the model size since it stores the fisher information of the model of the previous simulation step. Our proposed method provides the lowest additional memory usage because it only stores latent representations of the previous time steps.

Method	L1 - Loss	SSIM	Memory Overhead
No Protection	0.16	0.54	0 MB
Online - EWC	0.19	0.18	94 MB
A-GEM	**0.11**	**0.59**	2400 MB
Encode-Layerwise (Proposed)	0.13	0.46	**0.03 MB**

Table 1: Comparison of the autoencoder performance trained with different continual learning techniques. All values are the average values through all simulation time-steps. Additionally, we considered the memory overhead of each method

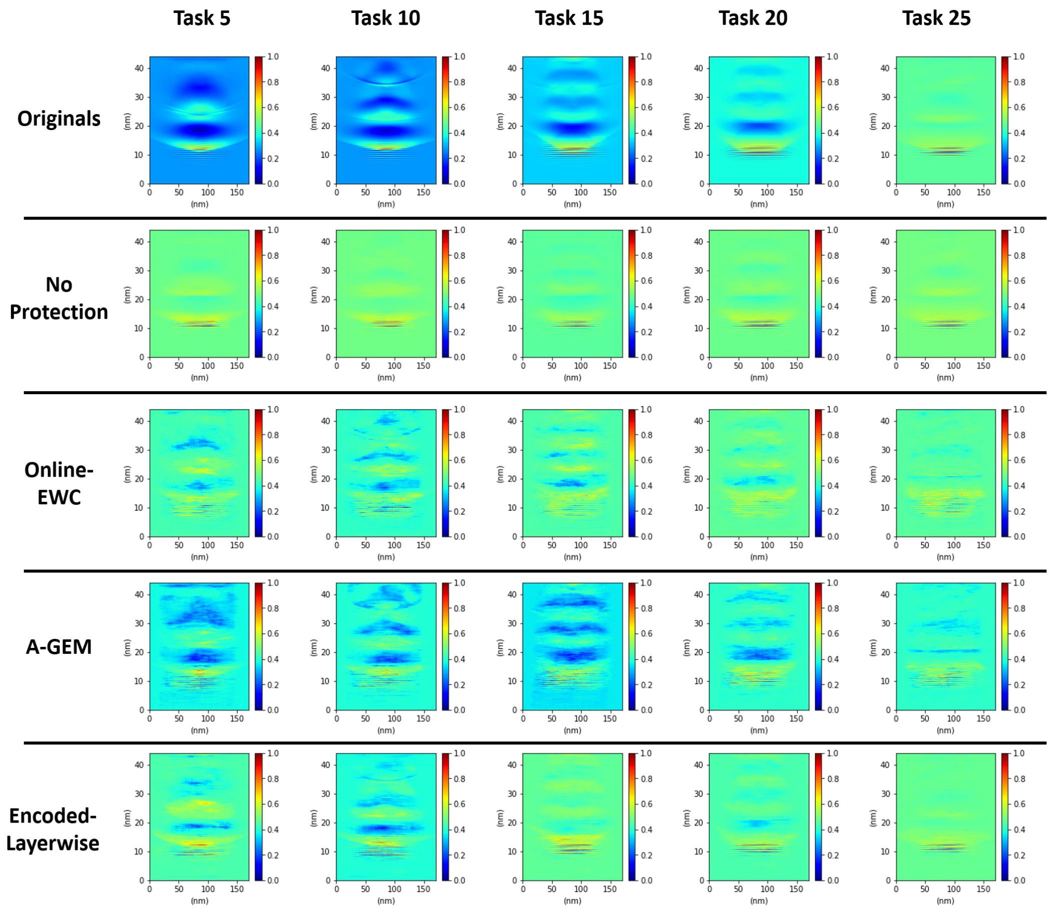

Figure 2: Post-training reconstructions of electrical fields by autoencoders trained with different continual learning techniques.The first row visualizes the ground truth electrical field. Each task corresponds to a simulation step

4 Conclusion

In this work, we presented a novel training pipeline for training an autoencoder concurrently to a running simulation while accessing the training data via streaming. We applied various continual learning techniques to improve the autoencoder's robustness and generalization and reduce the effect of catastrophic forgetting. We compared the applied methods with respect to reconstruction error, recovering structures, and memory consumption. Additionally, we extended the A-GEM algorithm for autoencoder training which heavily decreased the memory consumption of A-GEM. We conclude that replay-based methods, such as A-GEM and our proposed method, provide more accurate reconstruction since they access older simulation steps. Unfortunately, the trained autoencoder is only able to recover some high-oscillating patterns when it is trained in an online manner. We need to do more research to improve the trained autoencoder's reconstruction quality. In the future, we want to apply our approach on a larger scale and more sophisticated simulations.

References

[1] Patrick Stiller, Friedrich Bethke, Maximilian Böhme, Richard Pausch, Sunna Torge, Alexander Debus, Jan Vorberger, Michael Bussmann, and Nico Hoffmann. Large-scale neural solvers for partial differential equations, 2020.

[2] Nico Hoffmann, Edmund Koch, Gerald Steiner, Uwe Petersohn, and Matthias Kirsch. Learning thermal process representations for intraoperative analysis of cortical perfusion during ischemic strokes. In Gustavo Carneiro, Diana Mateus, Loïc Peter, Andrew Bradley, João Manuel R. S. Tavares, Vasileios Belagiannis, João Paulo Papa, Jacinto C. Nascimento, Marco Loog, Zhi Lu, Jaime S. Cardoso, and Julien Cornebise, editors, *Deep Learning and Data Labeling for Medical Applications*, pages 152–160, Cham, 2016. Springer International Publishing.

[3] Friedrich Bethke, Richard Pausch, Patrick Stiller, Alexander Debus, Michael Bussmann, and Nico Hoffmann. Invertible surrogate models: Joint surrogate modelling and reconstruction of laser-wakefield acceleration by invertible neural networks, 2021.

[4] Chaudhry Arslan, Ranzato Marc'Aurelio, Rohrbach Marcus, and Elhoseiny Mohamed. Efficient lifelong learning with a-gem.

[5] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[6] Burau Heiko, Renée Widera, Wolfgang Hönig, Guido Juckeland, Alexander Debus, Thomas Kluge, Ulrich Schramm, Tomas E. Cowan, Roland Sauerbrey, and Michael Bussmann. Picongpu: A fully relativistic particle-in-cell code for a gpu cluster. https://ieeexplore.ieee.org/document/5556015, 2010.

[7] Ruonan Wang et al. William F.Godoy, Norbert Podhorszki. *ADIOS 2: The Adaptable Input Output System. A framework for high-performance data management.* https://www.sciencedirect.com/science/article/pii/S2352711019302560, 2020.

[8] Axel Huebl, Remi Lehe, and Jean-Luc Vay et al. openpmd 1.0.0: A meta data standard for particle and mesh based data.

[9] Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum (Proc. Eurographics), 38, 2 (2019), 59-70*, 2018.

[10] Anna Willmann, Patrick Stiller, Alexander Debus, Arie Irman, Richard Pausch, Yen-Yu Chang, Michael Bussmann, and Nico Hoffmann. Data-driven shadowgraph simulation of a 3d object, 2021.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[12] Jim Nilsson and Tomas Akenine-Möller. Understanding ssim, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [N/A]
- Did you include the license to the code and datasets? [N/A]
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 3 and Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 3
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]