INL/JOU-22-69491-Revision-0

Enabling scientific machine learning in MOOSE using Libtorch

July 2023

Peter German, Dewen Yushu

INL/JOU-22-69491-Revision-0

Enabling scientific machine learning in MOOSE using Libtorch

Peter German, Dewen Yushu

July 2023

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Enabling scientific machine learning in MOOSE using Libtorch

Péter German[a,*], Dewen Yushu[b]

[a] Computational Frameworks Department, Idaho National Laboratory, Idaho Falls, ID 83415, United States of America
[b] Computational Mechanics and Materials Department, Idaho National Laboratory, Idaho Falls, ID 83415, United States of America

ARTICLE INFO

Article history:
Received 21 December 2022
Accepted 24 July 2023

Keywords:
MOOSE
Libtorch
Scientific machine learning
Reinforcement learning

ABSTRACT

A neural-network-based machine learning interface has been developed for the Multiphysics Object-Oriented Simulation Environment (`MOOSE`). The interface relies on Libtorch, the C++ front-end of PyTorch, and enables an online interaction between modern machine learning algorithms and all the existing simulation, modeling, and analysis processes available in `MOOSE`. New capabilities in `MOOSE` include the native generation and training of artificial neural networks together with options to load pretrained neural networks in TorchScript format. Furthermore, the `MOOSE` stochastic tools module (`MOOSE-STM`) has been enhanced with neural network-based surrogate and reduced-order model generation options for efficient stochastic analyses. Lastly, a reinforcement learning capability has been added to `MOOSE-STM` for the interactive control and optimization of complex multiphysics problems.

Code metadata

Current code version	N/A (uses continuous stable branch)
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00435
Legal Code License	LGPL 2.1
Code versioning system used	Continuous stable branch with git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	C++17 compiler (GCC or Clang) Memory: 16GB+ Disk: 30GB+ OS: Mac OS 10.13+, Linux (POSIX) Deps: MPI, PETSc, libMesh, Libtorch 1.4+
If available Link to developer documentation/manual	https://mooseframework.inl.gov/
Support email for questions	https://github.com/idaholab/moose/discussions

1. Motivation and significance

The modeling and simulation of complex systems utilizing computational software have become a cornerstone of modern industrial design and research practices considering that it can greatly reduce financial burden by partially (or fully) eliminating the need for expensive experiments and measurements. At the same time, scientific machine learning (ML), including neural network (NN) based algorithms, has reached a level where its application can greatly enhance the simulation-based design and research processes. However, most closed- and open-source software used for the numerical solution of complex multiphysics problems have poor support when it comes to inclusive modern ML capabilities. In many cases, this support is limited to file-based communication at the input–output level, which does not allow an online interaction between the numerical simulation and ML algorithms. Clearly, this loose-coupling is error-prone and not suitable for use cases that require constant interaction with the simulation environment (e.g., reinforcement learning [RL]). For this reason, an ML interface has been developed within the Multiphysics Object-Oriented Simulation Environment (`MOOSE`) [1] using Libtorch (C++ Frontend of PyTorch) [2], which enables direct access to NN-based learning algorithms within multiphysics simulations.

`MOOSE` is a C++ based open-source simulation framework that has been widely used for the modeling and simulation of complex systems and processes, such as mechanical, thermal, chemical,

* Corresponding author.
E-mail address: peter.german@inl.gov (Péter German).

electrochemical behavior of nuclear materials [3], additively manufactured materials [4,5], electromagnetic phenomena [6], etc. By enabling a tight interaction between MOOSE and Libtorch, we provide functionalities such as classification, surrogate and reduced-order model generation, and NN-based controller design natively within MOOSE and consequently in every MOOSE-based application (i.e., Bison [7], Griffin [8], Marmot [9], etc.). This is achieved by dynamically linking the MOOSE and Libtorch libraries. The main applications of this enhancement include but are not limited to the substitution of materials' constitutive relations by NN-based models in modeling nuclear materials, neutron transport, phase-field simulations, NN-based control and optimization of complex physical systems, etc.

2. Software description

The newly implemented functionalities provided by Libtorch are part of MOOSE and therefore follow its design considerations, continuous integration, and documentation directions [10]. The interface has been tested (using continuous integration) on Linux and Intel Mac machines. The new capabilities have been enabled by dynamically linking MOOSE with Libtorch, which can be easily done by setup and configure scripts distributed within MOOSE.

2.1. Software architecture

MOOSE, as shown in Fig. 1, can be divided into core capabilities, located in the framework, and physics-specific capabilities, which have been placed in physics modules. The Libtorch-based ML functionalities have been divided between the framework and the MOOSE stochastic tools module (MOOSE-STM) [11], which incorporates algorithms necessary for efficient stochastic analysis, surrogate generation, and data analysis. Even though the syntax of Libtorch is directly available in MOOSE and MOOSE-based applications, several wrapper classes have been created to simplify the utilization, creation, and training of NN models in MOOSE. Fig. 1 presents the integration of Libtorch-based functionalities in MOOSE:

2.2. Software functionalities

2.2.1. Capabilities in the framework

As shown in Fig. 2, the framework is now equipped with a general NN module (i.e., LibtorchNeuralNet), which contains an evaluation (forward) function to make sure that the different types of NN models can be used in an abstract way. Two major classes inherit from this class. The first class is LibtorchArtificialNeuralNet. It allows the user to set up an NN model with a given architecture using the constructor presented below:

```
LibtorchArtificialNeuralNet::LibtorchArtificial
NeuralNet(
    const std::string name,
    const unsigned int num_inputs,
    const unsigned int num_outputs,
    const std::vector<unsigned int> & num_neurons_
    per_layer,
    const std::vector<std::string> & activation_
    function);
```

This allows the user to specify the number of hidden layers together with the number of neurons and activation function used for each hidden layer.

The other class is LibtorchTorchScriptNeuralNet. It reads a TorchScript file, which can be the output of a completed training process using the Python API of PyTorch. This capability provides a streamlined method for training NNs in Python and deploying them in MOOSE, which can be useful if users do not need a direct integration between the NN models and the system or if they need specific training capabilities only available in the Python-based layers of PyTorch.

A class LibtorchArtificialNeuralNetTrainer has also been added for training LibtorchArtificialNeuralNetworks. It utilizes a standard backpropagation-based gradient descent algorithm [12], which takes parameters like the learning rate or optimizer type. The trainer is designed to enable training in parallel using Message Passing Interface (MPI) and a centralized scheme (based on MPI_Allreduce) for the computation of the gradient [13]. This scheme splits the samples in every training batch among the processes and computes a gradient estimate on each process. The final gradient estimate is then obtained by averaging the gradients over all the processes. This involves an MPI_Allreduce operation which can be expensive if called frequently. As a last step, the parameters of the neural network are updated using the averaged gradient on every process independently. One can instantiate a trainer using a LibtorchArtificialNeuralNetwork object and a Parallel::Communicator, which is a wrapper around MPI_Comm, located in libMesh [14], the library that provides the algorithms for the discretization of partial differential equations (PDE) in MOOSE:

```
LibtorchArtificialNeuralNetTrainer(
    std::shared_ptr<LibtorchNeuralNet<torch::nn
    :: Module>> nn,
    const Parallel::Communicator & comm);
```

and train it with the train(...) function, which requires a data set and corresponding training options:

```
virtual void train(LibtorchDataset & dataset,
                   const LibtorchTrainingOptions &
                   options);
```

The detailed description of each object is available in the in-code Doxygen documentation of MOOSE. The weak- and strong-scaling of the training process are shown in Fig. 3. The study was performed on an AMD EPYC 7702 processor with up to 32 cores and using Rocky-8 operating system. We see that the centralized scheme performs well if the number of batches (gradient updates in each epoch) is relatively small. As the number of batches increases, the communication costs increase simultaneously. This behavior is attributed to the fact that in the centralized parallel training scheme an average gradient is computed over the processes using MPI_Allreduce which has a considerable overhead in terms of communication. Future work is targeted at the addition of a decentralized training system (such as [15]) to avoid extensive communication costs.

Additionally, the framework has been extended to be able to read or receive NNs that can control physical processes. These controllers (LibtorchNeuralNetControl and LibtorchDRLControl) are able to read either parameter files for fixed NN architectures or TorchScript files of arbitrary NNs trained using the Python API of PyTorch.

2.2.2. Capabilities in the stochastic tools module

The MOOSE-STM builds upon the utility classes in the framework and contains more complex structures. For example, it can now train artificial NNs to be surrogates of MOOSE-based applications for stochastic analyses (i.e., sensitivity analysis, uncertainty quantification). This is achieved by LibtorchANNTrainer and LibtorchANNSurrogate. These objects have user interfaces that enable the user to customize the NN architecture together with the training parameters at an input-file level:

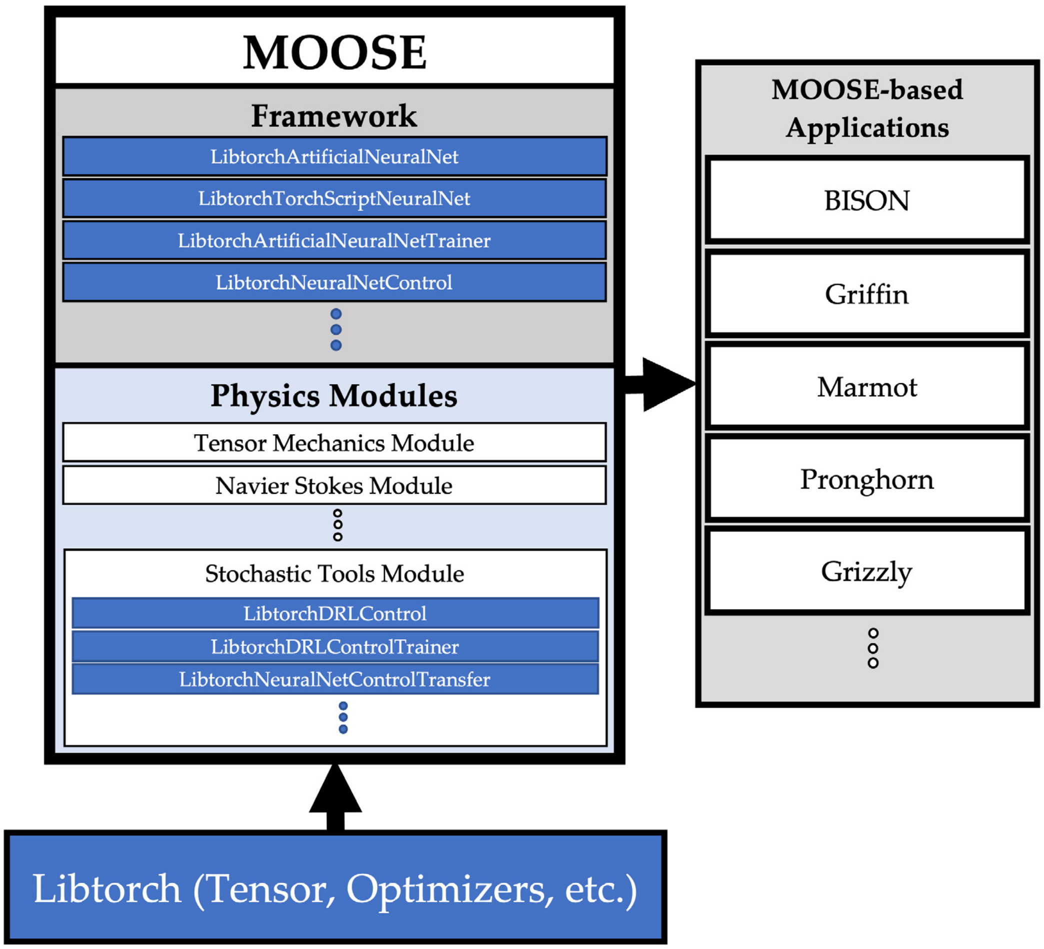
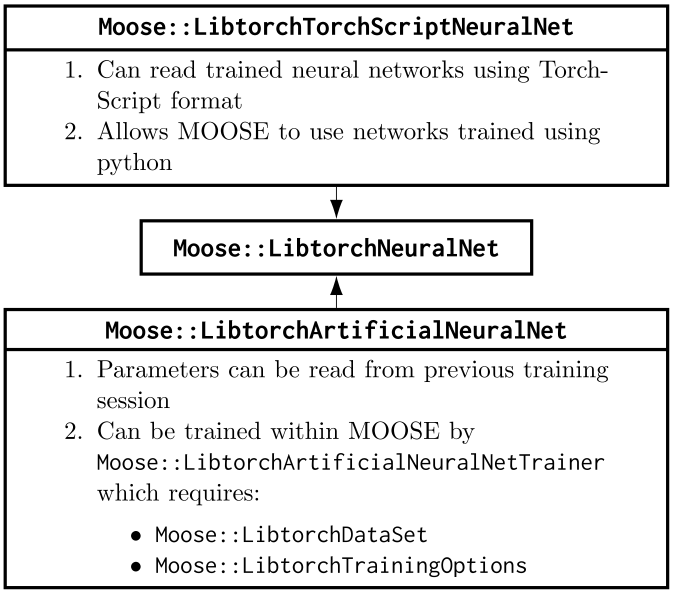


Fig. 1. The flowchart of MOOSE utilizing Libtorch with dark blue fields representing objects directly harnessing functionalities from Libtorch.

```
[Trainers]
  [train]
    type = LibtorchANNTrainer
    sampler = sample
    response = values/g_values
    num_epochs = 40
    num_batches = 10
    num_neurons_per_layer = '64 32'
    learning_rate = 0.001
    activation_function = 'relu relu'
  []
[]
```

Another functionality added to the MOOSE-STM is the generation of controllers using Proximal Policy Optimization (PPO) [16], which belongs to the family of RL algorithms. The implementation is application-agnostic within MOOSE as long as the observations on the system are measured using MOOSE's Postprocessors and the simulations are controlled using the Control system. The training also utilizes the MultiApp system, meaning that there is a main application that can run and communicate with the sub-applications that simulate complex multiphysics problems, for more details see [3]. In this scenario, the main application will contain a LibtorchDRLControlTrainer object that trains the critic and actor (control) NNs commonly used in the PPO algorithm [16]. The control NN in each iteration is transferred to the sub-application by LibtorchNeuralNetControlTransfer, which uses it to control the physical process and gather data for further training. The reward of the training process can be monitored using Postprocessors such as DRLRewardReporter.


Moose::LibtorchTorchScriptNeuralNet

1. Can read trained neural networks using TorchScript format
2. Allows MOOSE to use networks trained using python

Moose::LibtorchNeuralNet

Moose::LibtorchArtificialNeuralNet

1. Parameters can be read from previous training session
2. Can be trained within MOOSE by Moose::LibtorchArtificialNeuralNetTrainer which requires:
   - Moose::LibtorchDataSet
   - Moose::LibtorchTrainingOptions

Fig. 2. The neural network wrapper classes in MOOSE framework together with their possible training options.

3. Illustrative examples

To illustrate the applicability of Libtorch within MOOSE, we present two examples. The first example is meant to demonstrate

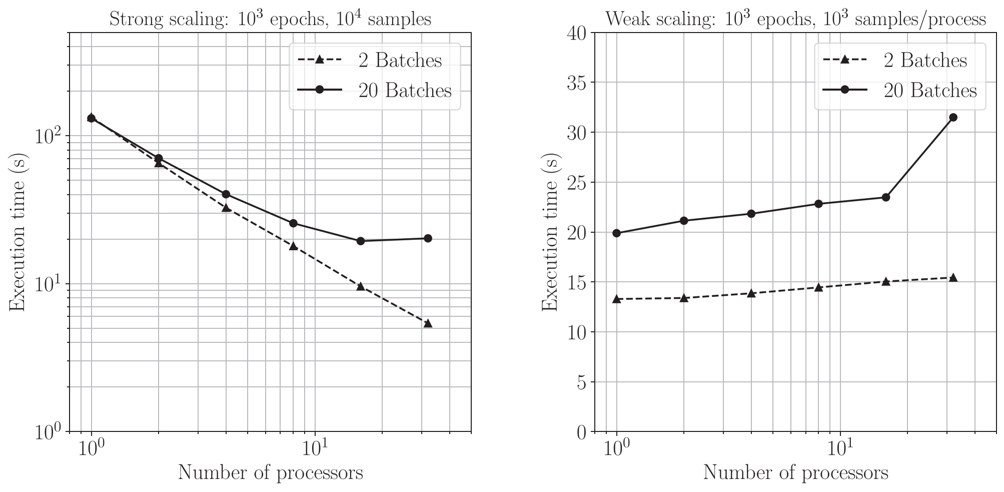


Fig. 3. Strong (left) and weak (right) scaling of the training of an NN in MOOSE using a centralized scheme.

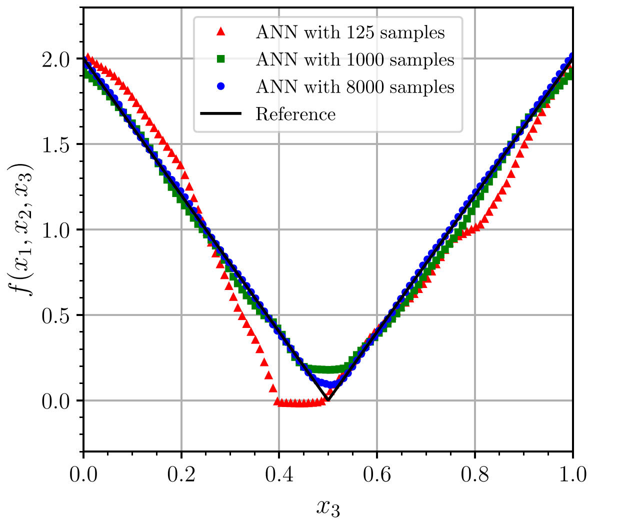


Fig. 4. Approximation of $f(x_1, x_2, x_3) = \prod_{i=1}^{3} |4x_i - 2|$ with NNs within MOOSE-STM. Results are plotted along the $(x_1 = 0.25,\ x_2 = 0.25,\ x_3)$ line.

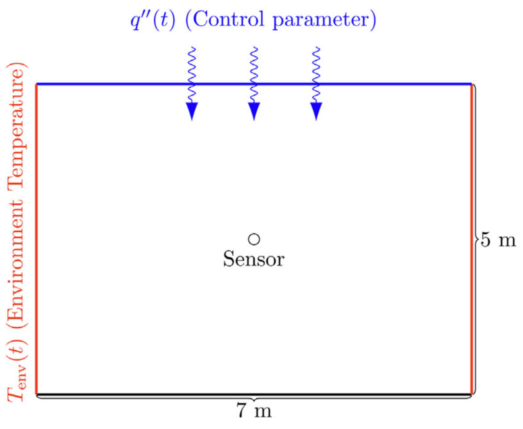

$q''(t)$ (Control parameter)
$T_{\text{env}}(t)$ (Environment Temperature)
Sensor
5 m
7 m

Fig. 5. The problem setup of the deep reinforcement learning controller for air conditioning in MOOSE-STM.

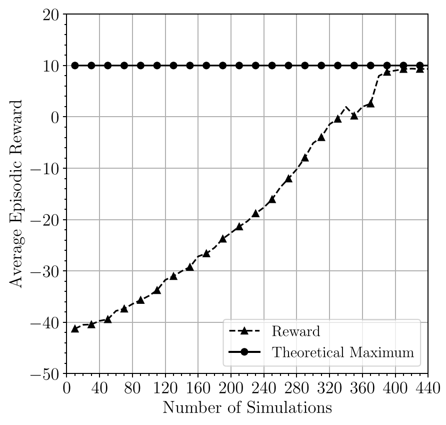
Table 1
Root Mean Square Error (RMSE) between the neural network approximations and target function evaluations over a test set of 10^6 samples.

# of samples used for training	Test set RMSE
125	0.265
1,000	0.088
8,000	0.032

the applicability of Libtorch-based NNs in MOOSE to approximate functions. This capability can be of interest to users who want to use NNs as surrogates of material properties or other physical processes. The function of interest in this example is $f(x_1, x_2, x_3) = \prod_{i=1}^{3} |4x_i - 2|$, meaning that we attempt to fit a function in a three-dimensional space. We pick samples of the inputs in the $(x_1, x_2, x_3) \in [0, 1]^3$ interval uniformly using a tensor product approach and fix the NN architecture to have three hidden layers with 128, 64, and 32 neurons each. Fig. 4 presents three NN function approximations over a line parallel to the x_3 axis, at $x_1 = x_2 = 0.25$. The presented networks were trained by utilizing different amounts of data, with 5, 10, and 20 grid points per dimension, resulting in 125, 1,000 and 8,000 samples in total. Furthermore, Table 1 presents the Root Mean Square Error (RMSE) over a tests set consisting of 10^6 additional samples over the same domain. It is clear that the more samples available for the training, the more accurate the NN is.

In the second example, we train a controller for keeping the temperature within a room at an optimal value under changing environment conditions using RL. The problem setting is depicted in Fig. 6. It consists of a 2D room where the temperature of the side walls changes with the environmental temperature, which varies throughout the day using the following function: $T_{env}(t) = 273 + 15\sin\left(\frac{\pi t}{86400}\right)$. The top wall, on the other hand, acts as an air-conditioner capable of heating or cooling the room using a controlled heat flux (Neumann boundary condition). The room is assumed to be filled with air, and the temperature profile in the room is determined by solving a transient heat-conduction equation.

The goal of this example is to train an NN-based controller that can keep the temperature at a comfortable level in the middle of the room. This means that the observation (input for the controller) is the temperature at the sensor location (see Fig. 5), while the control value (output of the controller) is the heat flux value on the top. A PPO algorithm [16] is used for training and needs two NNs: a critic and an actor (controller). The control NN is chosen to be a simple two-layer feed-forward NN with 16 and 6 neurons on the hidden layers. The critic NN, which tries to approximate the return function, is slightly more complex; it has two hidden layers with 64 and 27 neurons, respectively. Fig. 6 shows that the trained control NN can keep the temperature at



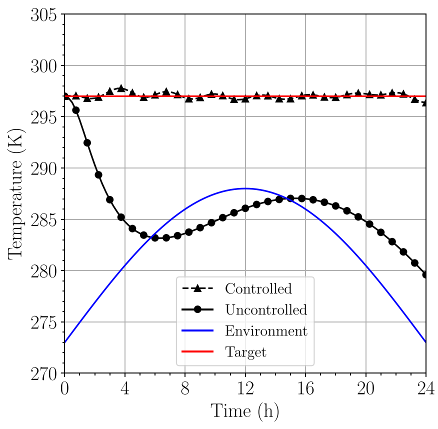


Fig. 6. The evolution of the reward during training (left) and the results obtained using the final controller (right), where 1 episode = 10 simulations.

the location of the sensor very close to the desired value. The remaining oscillations are the result of allowing variability in the control action during training to avoid overfitting. This example is readily available in MOOSE-STM.

4. Impact

A Libtorch interface has been developed in MOOSE to enable integrated ML solutions. The interface has been designed to enable:

- Full and flexible access to Libtorch syntax within MOOSE for a variety of research topics;
- Allow online interaction between the ML and numerical methods that are used to solve multiphysics problems;
- User-friendly setup procedure and utilization of NN-based algorithms at the input file level.

This opens the door towards the adaptation of advanced ML algorithms in MOOSE. Furthermore, every MOOSE-based application inherits this capability. This entails that NN-based learning is available for a wide variety of complex physics problems that MOOSE-based applications tackle. These applications include but are not limited to advanced thermal hydraulics, nuclear material performance, particle transport, seismic, etc.

A key focus of this deployment is the development of RL capabilities in MOOSE that require a close interaction between the simulated system and the NN training process. This cannot be achieved with file-based interaction alone at the input–output level. Practical examples can be the training of controllers for complex problems, such as advanced manufacturing processes, nuclear systems, etc.

Lastly, an advantage of enabling these capabilities is that MOOSE has been developed using nuclear quality assurance standards [10], allowing for an easier adaptation for design and research projects in the industry.

5. Limitations

The linking between MOOSE and Libtorch has certain limitations that have been actively worked on. First, Mac computers with Apple Silicon chips are not fully supported yet. Second, if the user chooses to install the dependencies of MOOSE using the official MOOSE anaconda packages, a conflict arises between the GLIBC versions of the conda-based compiler stack on Linux (uses Version 2.17 to support older Linux distributions) and the Libtorch distributions (uses Version 2.28+). Nevertheless, modern Linux distributions come with up-to-date GLIBC versions, which allow the manual installation of MOOSE dependencies, and consequently, the linking with Libtorch. Lastly, we note that MOOSE is a central processing unit (CPU) based library and is set up to work with the CPU-based functionalities of Libtorch using MPI for parallelism. Enabling the graphics processing unit (GPU) based capabilities of Libtorch within MOOSE is part of future work.

6. Conclusions

Scientific ML capabilities have been enabled for complex multiphysics simulations by developing an interface between MOOSE and Libtorch. Multiple wrapper classes have been added to MOOSE and the MOOSE-STM to create, train, and evaluate NN models using Libtorch. The wrappers can be used to train an NN model within MOOSE or to read pre-trained NN models using the TorchScript format. Furthermore, MOOSE-based training algorithms allow CPU-based parallel training using an MPI with a centralized scheme. The Control system of MOOSE has also been extended to allow the utilization of NNs as control agents in complex multiphysics simulations. Building on this, a PPO algorithm has been implemented in the MOOSE-STM to natively train RL-based NN models for the control of multiphysics simulations. The PPO algorithm in MOOSE is agnostic of physics so users can harness it for every MOOSE-based application with input file-level interactions alone.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Peter German reports financial support was provided by Idaho National Laboratory. Dewen Yushu reports financial support was provided by Idaho National Laboratory.

Data availability

The code to produce the data is available in the moose repository.

Acknowledgments

This work was supported through the Idaho National Laboratory Directed Research & Development, United States Program under Department of Energy Idaho Operations Office contract no. DE-AC07-05ID14517. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes. We would also like to give special thanks to Dr. Zonggen Yi for his help in the development of the PPO algorithm.

References

[1] Lindsay AD, Gaston DR, Permann CJ, Miller JM, Andrš D, Slaughter AE, et al. 2.0 - MOOSE: Enabling massively parallel multiphysics simulation. SoftwareX 2022;20:101202. http://dx.doi.org/10.1016/j.softx.2022.101202, URL https://www.sciencedirect.com/science/article/pii/S2352711022001200.

[2] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché Buc F, Fox E, Garnett R, editors. Advances in neural information processing systems. Vol. 32. Curran Associates, Inc; 2019, p. 8024–35, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[3] Gaston DR, Permann CJ, Peterson JW, Slaughter AE, Andrš D, Wang Y, et al. Physics-based multiscale coupling for full core nuclear reactor simulation. Ann Nucl Energy 2015;84:45–54.

[4] Grilli N, Hu D, Yushu D, Chen F, Yan W. Crystal plasticity model of residual stress in additive manufacturing. 2021, arXiv preprint arXiv:2105.13257.

[5] Yushu D, McMurtrey MD, Jiang W, Kong F. Directed energy deposition process modeling: A geometry-free thermo-mechanical model with adaptive subdomain construction. Int J Adv Manuf Technol 2022;1–20.

[6] Icenhour CT. Development and validation of open source software for electromagnetics simulation and multiphysics coupling (Ph.D. thesis), North Carolina State University; 2023.

[7] Williamson R, Gamble K, Perez D, Novascone S, Pastore G, Gardner R, et al. Validating the BISON fuel performance code to integral LWR experiments. Nucl Eng Des 2016;301:232–44.

[8] Wang Y, Schunert S, Ortensi J, Laboure V, DeHart M, Prince Z, et al. Rattlesnake: A MOOSE-based multiphysics multischeme radiation transport application. Nucl Technol 2021;207(7):1047–72.

[9] Schwen D, Aagesen LK, Peterson JW, Tonks MR. Rapid multiphase-field model development using a modular free energy based approach with automatic differentiation in MOOSE/MARMOT. Comput Mater Sci 2017;132:36–45.

[10] Slaughter A, Permann CJ, Miller J, Alger B, Novascone S. Continuous integration, in-code documentation, and automation for nuclear quality assurance conformance. Nucl Technol 2021;1–8. http://dx.doi.org/10.1080/00295450.2020.1826804.

[11] Prince Z, Slaughter A, German P, Halvic I, Jiang W, Spencer B, et al. Moose stochastic tools: A module for performing parallel, memory-efficient in situ stochastic simulations. 2022, SSRN, URL http://dx.doi.org/10.2139/ssrn.4049487.

[12] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. nature 1986;323(6088):533–6.

[13] Seide F, Fu H, Droppo J, Li G, Yu D. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In: Fifteenth annual conference of the international speech communication association. Citeseer; 2014.

[14] Kirk BS, Peterson JW, Stogner RH, Carey GF. `libMesh`: A C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng Comput 2006;22(3–4):237–54. http://dx.doi.org/10.1007/s00366-006-0049-3.

[15] Ghosh S, Aquino B, Gupta V. EventGraD: Event-triggered communication in parallel machine learning. Neurocomputing 2022;483:474–87.

[16] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, arXiv preprint arXiv:1707.06347.