Response to Referee # 1

RoseNNa: A performant, portable library for neural network inference with application to computational fluid dynamics

Ajay Bati and Spencer H. Bryngelson

We are grateful for the referee's efforts in improving the quality of this paper. We quote the comments of the referee and discuss changes made to the paper in response to these comments in the following.

Referee #1

This paper reports on a new tool that enables the transformation of neural network work models from Python code into Fortran code. This is a useful tool as it expands on various other tools that try to do similar conversions. The programmers have reviewed the available tools in the Introduction section.

The value of this work is that Fortran remains a primary language for floating point numerical calculations in HPC. Linking Fortran to C code is now readily available in modern compilers and run-time systems. Using this tool, one can integrate the use of neural networks into existing Fortran and C software suites. The code is available via a GitHub site. It is straightforward to download and use. The authors have reported on performance statistics in the paper, in section 4.2.

The tool will be a valuable asset for the scientific research community and the paper should be published, in my opinion. I would like the author however to address two particular software engineering aspects in a final manuscript.

RoseNNa Error Handling.

1. How are errors handled by the tool?

    We thank the reviewer for raising this point, which should have been included in the original manuscript. Errors are handled in the "Model Topology Encoding" step of the tool's pipeline. RoseNNa handles a specific subset of machine learning components (activation functions, layers, layer options, etc.) that incorporate a majority of what is used in the CFD and PDE-solver community. There are two types of errors that users may find while using RoseNNa: implicit and explicit errors. Explicit errors occur when there is a lack of support for a certain feature and an error will be displayed. Implicit errors occur when the encoded model outputs values inconsistent with the ground truth model (or the Python-native model) output values.

    We provided an additional section, section 3.2, that clarifies RoseNNa's error handling. It reads: "RoseNNa supports a subset of neural network capabilities, as mentioned in Section 2. These include MLP, LSTM, Convolutional, and Max/AvgPool layers and Tanh, ReLU, and Sigmoid activation functions. While functionality can be readily extended, unsupported features detected during the Model Topology Encoding stage will be noticed by RoseNNa and raised as an error. These are explicit errors, with the user immediately alerted of an unsupported feature. There are, however, implicit errors that may occur when the encoded model's predicted outputs are not the same as its Python-native model counterpart (the input). This can be associated with incorrect pre-processing or implementation of an internal feature. Users should write additional test cases for their converted models to reduce the possibility of raising errors."

2. The text at the start of section 4.2 mentions 'commonly used' network architectures. Does the tool work only for any type of architecture? If not, can the tool detect the type of architecture and terminate with an error message?

   The referee raises a great point, and we have not addressed this thoroughly in the text. We provide a description here and include the new manuscript text.

   The tool is not exhaustive and only supports certain features, which are mentioned in section 2.4. These include specific activation functions, layers, layer options, and so on. It can, of course, be readily extended to include more. We have further clarified that RoseNNa currently only supports LSTM, MLP, convolutions, and pooling layers in section 2.4, which now reads: "RoseNNa was designed to support a broad range of neural network architectures in CFD. As discussed in section 1, these primarily include MLPs and LSTM RNNs. RoseNNa also supports other architectures, such as convolutional and pooling layers, which are generally popular and could become more broadly used in CFD solvers in the future. The tool was built and tested using a CPU, and we currently support single and double precision models. One can expand RoseNNa for different architectures, activation functions, model precision, and other cases as needed. Adding these new features to the tool requires only a basic understanding of the architecture functionality, how ONNX encodes it, and following the RoseNNa contributor's guide for implementation."

   If the user attempts to encode a model with an unsupported feature, RoseNNa detects the architecture type in the "Model Topology Encoding" stage and returns an error message "{feature} NOT SUPPORTED BY RoseNNa CURRENTLY!".

   We have added details regarding architecture feature detection at pre-processing in section 2. We also clarify architecture error handling in section 3.2 (see our reply to comment 1).

3. The text discusses the number of hidden layers and the number of nodes, as well as various activation functions. What is the largest number of hidden layers and how many nodes can the tool handle? Does it report an error if a larger network is passed as input, or is the tool in principle designed to handle an arbitrary number? The text towards the end of section 4.2 mentions for example 15 layers of 100 neurons.

   RoseNNa can handle an arbitrary number of nodes, where a node is any feature the tool supports. This is due to the way we encode and decode the input model. We emphasize this information in section 2.4, which discusses RoseNNa's capabilities and further explains how the encoding process enables the handling of arbitrary sizes in sections 2.2 and 2.3. Representative new text includes: "The decoding stage (fig. 1 (f)) references each component described above, also looping over the recorded features. Using fypp, the layers are called in order with their respective weights, constants, and other supplementary options. The encoding and decoding process does not have a well-defined upper-limit on input size, limited implicitly by the hardware memory. As such, RoseNNa can handle hardware-supported inputs, in part owing to fypp's ability to process arbitrary-length Python code."

4. In a similar way, does the tool monitor the types of activation functions used, and is it limited to a subset of these functions?

This is a good point. Similar to architecture detection, we also monitor the activation functions. We mention which activation functions we target in section 2.1: " Currently, RoseNNa must support only a subset of all possible neural network layers (here: LSTMs, convolutions, pooling layers, and MLPs) and their features and activation functions (here: Tanh, ReLU, Sigmoid). During the "Encoding" process, these layers and their features are detected, and hyperparameters and their ordering are recorded. RoseNNa only proceeds if the detected input features are supported. Otherwise, a log indicating the unsupported feature is sent to standard output. "

We further clarified what is currently supported (and not supported) in section 2.4 (please see the additional new text in our reply to comment 2).

Numerical results.

1. Some neural networks are written to use 16-bit floating point numbers. Can the tool handle these?

   Another good point raised by the referee. At present, RoseNNa supports single- and double-precision models. Extended precision support can be easily integrated and will be supported in the future via Fortran intrinsics. We have included additional text in the new section 2.4 to clarify this: "RoseNNA was built and tested using CPU execution, currently supporting single- and double-precision models. One can expand RoseNNa for different architectures, activation functions, model precision, and other cases as needed by following the RoseNNa contributor's guide. Adding these new features to the tool requires only a basic understanding of the architecture functionality and how ONNX encodes it. "

2. It is well known that changing the order of floating point operations can impact the accuracy of the numerical results. It would be useful to know if the authors have looked at the absolute values in some test cases and compared inferencing in Python against the code output from their tool. How, if at all, do the values change?

We use a suite of continuous integration tests on GitHub that execute with each pull request and commit to the RoseNNa repository. The tests are designed to determine if the RoseNNa-encoded model computes the same values as the Python-native one. This is especially important for design changes, which are more likely prone to user error in our estimation. We include this information and add specific details about tests in a new section, section 2.5, which reads:

"While the conversion process may compile and a given input runs through the converted model successfully, we confirm that the pre-processing and feature functionality is correct. For this, we use continuous integration (CI) testing to ensure the converted model output matches the outputs of the original Python-native model. Any GitHub pull request or commits trigger a CI run of the test suite to ensure previous functionality remains intact and added functionality is correct.

At the time of writing, RoseNNa has 17 tests, each executed via CI with each pull request and commit. RoseNNa tests the core functionality of each layer (LSTM, MLP, Maxpool, Avgpool, and Convolutions). Then, it creates cases to test different hyperparameters of these layers, including size, bias, and stride. Composition tests are also included, combining different layers and activation functions. We advise users who are extending RoseNNa's functionality to follow the test suite documentation and add core, option, and composition tests to ensure pre-processing and internal functionality are correctly implemented."

Response to Referee # 2

RoseNNa: A performant, portable library for neural network inference with application to computational fluid dynamics

Ajay Bati and Spencer H. Bryngelson

We are grateful for the referee's efforts in improving the quality of this paper. We quote the comments of the referee and discuss changes made to the paper in response to these comments in the following.

Referee #2

The authors have developed a portable library to interface neural network from C and Fortran called RoseNNa. Based on ONNX backend and fypp library RoseNNa offers a simple way to incorporate Deep Learning models trained by Python libraries like Pytorch into existing scientific applications. The work presented here will be very useful to the scientific computing community. As mentioned by the authors, Pytorch C++ API also offers C++ API to access pre-trained ML models. I personally use it in my large Fortran code base, but I agree with the authors that it would requires layers of Fortran/C/C++, which might not be ideal for users who only want to use small models in their software.

This paper is well-written, and the performance presented in the manuscript is decent. I believe this manuscript is worth publishing on CPC, but I would like the authors' comments on the following points in the manuscript.

1. It seems that the authors are not targeting GPU performance. Would the performance presented in Fig. 2, 3, and 4 change if a GPU being used in the benchmark?

   This is a good question without a particularly sharp answer, unfortunately. The short answer, in our view, is that the results might be somewhat different, though this difference would depend on many other factors that are out of our control at the level of the RoseNNa tool, and are unlikely to be qualitatively important since RoseNNa would operate at the per-thread level of the GPU in PDE solver settings.

   This is because we target neural networks that operate on a per-grid-cell (or per glob of grid cells) basis, which are indeed appropriate for PDE solvers on discretized spatio-temporal domains: Each neural network inference invocation is part of a larger GPU kernel on domain-decomposed PDEs. Since the GPU kernels are likely to saturate the GPU SMs and warps at the grid cell level, it seems likely that any neural network inference tool would operate on a single GPU thread, anyway. As a result, performance comparisons are unlikely to *qualitatively* change.

   It is the user and the design of their PDE solver that determines how to handle and offload these exterior loops over the spatial computational domain onto a GPU. For example, HIP, CUDA, OpenMP, CUDA, Kokkos, RAJA, and more are all possibilities, and each entails different handling at the level of the PDE solver. As a result, it would be nearly impossible to maintain a library that accommodates every GPU offloading tool at the level of the individual grid cell.

As a bellwether of inference performance, we evaluate RoseNNa on CPU hardware. We have included new text to clarify the above points for the reader: "Note that we anticipate these results could look different on GPU hardware, and, surely, inference times would be lower for both PyTorch and RoseNNa. However, it is challenging to ascertain these differences since RoseNNa is intended to be invoked at the individual grid cell level (or a glob of grid cells). As such, the performance depends strongly on how the user handles the GPU kernels that compute other PDE-relevant operations at each grid cell. In practice, the GPU threads will likely be saturated and RoseNNa, or any other neural network inference library, would operate on a single GPU thread."

2. How many threads are used in the benchmark with MKL and OpenBLAS?

   Indeed the thread use is an important piece of information that we should have included in the original manuscript. We use 1 thread to evaluate the models for both MKL and OpenBLAS cases. This is now emphasized in section 4.2, where we describe our hardware invocation. The following text is included: "The same models curated in PyTorch were converted to and tested in RoseNNa. Then, we took the ratio of the medians of the 100 RoseNNa and 100 PyTorch times. This process was repeated 25 times for each point in figs. 2 to 4. Each test was curated under the same hardware setting: A single thread of a Intel Xeon Gold 6226 CPU."

3. The authors should have some references for "Most MLPs used in CFD are shallow to enable reasonable computation runtimes." What are the common use cases of ML models in CFD applications? It will be useful for users to see some use cases of MLP in CFD applications.

   We agree with the referee; more examples can help motivate the primary RoseNNa use cases. We have updated our introduction to include more examples of MLPs used in CFD. The updated text includes: "Computational fluid dynamics (CFD) practitioners have been developing neural-network-based models to enhance traditional closures models and numerical methods. For example, Fukami et al. [1] implemented a convolutional autoencoder and multilayer perceptron (MLP) to speedup turbulence simulations, and Zhu et al. [2] showed how multiple artificial neural networks (ANNs) can model turbulence at high Reynolds numbers. Laubscher and Rousseau [3] used variational autoencoders and MLPs to predict cell-by-cell distributions of temperature, velocity, and species mass fractions for turbulent jet diffusion flames. Pirnia et al. [4] used an ANN to predict drag force on smaller particles and their results showed promise in improving drag force weighting in the coarse-grid method. Lastly, Baymani et al. [5] evaluated the capabilities of their feed-forward neural network by examining the electroosmotic flow through a two-dimensional microchannel. They found their MLP model was a fast solution to the Navier-Stokes equations. Of course, there are many other such examples. These trained models show promising results but are often not integrated into high-performance solvers to deploy the model at scale."

4. I have never come across fypp. Since RoseNNa depends on fypp, readers would like to know how the project is reliable, or used in many applications so that their development effort will not be wasted once the project slows down or gets terminated. Could the authors comment on this?

A good point that should be discussed. We do incur some risk by using a dependency, though presumably, nearly all software libraries do. However, the referee raises a useful point as fypp is not as well established as some other libraries (LAPACK, BLAS, and so on). Since we rely only upon basic fypp functionality, it appears unlikely that a breaking change could occur. In the case that a feature of fypp in RoseNNa's codebase breaks due to some updates to the fypp library or Python version changes, it would be feasible for a third party to edit the fypp codebase due to its relative simplicity (the entire fypp codebase is about 3000 lines).

We use fypp because its basic features are enough to accommodate not only our current implementation but future extensions. Generally, extensions to our tool will include additional layer capabilities and activation functions and the options that describe them. As now stated in section 2.4: "Adding these new features to the tool requires only a basic understanding of the architecture functionality, how ONNX encodes it, and following the RoseNNa contributor's guide for implementation." The last step requires integrating the implemented feature into the tool using fypp, and the existing features of fypp are sufficient to accomplish this task.

We emphasize RoseNNa's reliance on only basic features of fypp in section 2.3 to clarify that even a deprecated fypp version is sufficient for RoseNNa extension. The relevant text now reads: "In our implementation, fypp translates a neural network's properties into Fortran code *before* compile-time, thus exposing compiler optimizations. This decoding process is unique to each neural network, and so is re-run for different neural network models. The basic functionality of fypp is sufficient to extend RoseNNa with new features. Fypp is well maintained and is updated due to its wide usage in the Fortran community, though even if it were not, a deprecated fypp version would suffice for the RoseNNa's purposes."