

Neural network emulation of flow in heavy-ion collisions at intermediate energies

Nicholas Cox ^{*}, Xavier Grundler [†] and Bao-An Li [‡]

Department of Physics and Astronomy, Texas A&M University-Commerce, Commerce, Texas 75429, USA

 (Received 26 June 2024; revised 23 August 2024; accepted 30 September 2024; published xxxxxxxxx)

Applications of new techniques in machine learning are speeding up progress in research in various fields. In this work, we construct and evaluate a deep neural network (DNN) to be used within a Bayesian statistical framework as a faster and more reliable alternative to the Gaussian process (GP) emulator of an isospin-dependent Boltzmann-Uehling-Uhlenbeck (IBUU) transport model simulator of heavy-ion reactions at intermediate beam energies. We found strong evidence of the DNN being able to emulate the IBUU simulator's prediction on the strengths of protons' directed and elliptical flow very efficiently even with small training datasets and with accuracy about ten times higher than the GP. Limitations of our present work and future improvements are also discussed.

DOI: [10.1103/PhysRevC.00.004600](https://doi.org/10.1103/PhysRevC.00.004600)

I. INTRODUCTION

Various neural networks (NNs) have been widely used as useful tools in machine learning to extract more connections within datasets and also as a way to generate or simulate physical events [1]. Since the pioneering work of Ref. [2], promising results have been found in using NNs to address many critical issues in nuclear physics; see, e.g., Refs. [3–5] for recent reviews. In the area of heavy-ion reactions at intermediate energies (around tens of MeV/nucleon to about 2 GeV/nucleon before the quark degree of freedom becomes important), interesting results have been found from applying NN techniques. For example, these techniques have been used in determining the centrality [6], extracting nuclear symmetry energy [7], and classifying orientation and deformation of colliding nuclei [8] on an event-by-event basis. One particularly useful feature of NNs is their potential applications in efficiently and reliably emulating solutions/outputs of computationally expensive and physically complicated models/processes; see, e.g., Refs [9–18] for recent examples in low-energy nuclear physics. A huge draw to NNs is the possibility that they need much shorter computation times than traditional methods [19]. Indeed, very encouraging results and some challenges have been found in exploring such possibilities in nuclear physics, mostly in emulating nuclear structure and fission based on energy density functional theories so far [3–5].

In this work, we explore the feasibility of using deep neural networks (DNNs) to emulate an isospin-dependent Boltzmann-Uehling-Uhlenbeck (IBUU) transport model simulator [20,21] in predicting both the directed flow (v_1) and elliptical flow (v_2) [22–24] in heavy-ion reactions at

intermediate energies. Such an emulator is invaluable in further quantifying uncertainties of parameters characterizing the equation of state (EOS) (e.g., incompressibility K) and transport properties (e.g., in-medium nucleon-nucleon cross section σ_{NN}^{med}) of hot and dense matter formed in heavy-ion reactions [25,26]. Within a Bayesian framework using a Gaussian process (GP) [27] emulator trained with IBUU events generated by using 89 sets of input model parameters [28], Li and Xie inferred the most probable incompressibility K and the modification factor (X) of in-medium nucleon-nucleon cross sections compared to their free-space values from protons' directed flow (measured using the mid-rapidity slope F_1 of v_1) and elliptical flow in mid-central Au+Au collisions at $E_{\text{beam}}/A = 1.23$ GeV taken by the HADES Collaboration [29,30]. While the GP emulator has been the standard emulation technique for model-to-data comparison in Bayesian analyses (Bayesian+GP) of heavy-ion collisions from low to ultrarelativistic beam energies (see, e.g., Refs. [31–42]), to reduce the interpolation uncertainty inherent in GP emulators, especially for large numbers of model parameters and multi-messenger observables, more efficient and accurate emulation techniques are being sought by the community. Moreover, generating a sufficiently large training dataset for GP can be computationally impractical. Therefore, alternative techniques to more efficiently and accurately emulate transport model simulators of heavy-ion reactions are certainly needed.

Here we focus on developing and evaluating a DNN as an alternative of GP in emulating heavy-ion reactions at intermediate beam energies. To determine if DNNs can replace all or part of the Bayesian+GP process with greater efficiency, we explore two possibilities. First, we test to see if DNN could replace the GP with greater efficiency, i.e., could we build a better emulator using the same amount of or less training events than GP? Second, we explore directly inferring the X and K values without the costly Markov chain Monte Carlo (MCMC) process required for the Bayesian analysis. We found strong evidence of the DNN being able to emulate

^{*}Contact author: ncox6@leomail.tamuc.edu

[†]Contact author: xgrundler@leomail.tamuc.edu

[‡]Contact author: Bao-An.Li@tamuc.edu

very efficiently the IBUU simulator's results (predicting F_1 and v_2 given X and K) even with small training datasets with an accuracy about ten times higher than the GP. On the other hand, the DNN's direct application in the inverse process is much less accurate. Therefore, a Bayesian+DNN inference will be a viable alternative to the more computationally costly Bayesian+GP approach.

The rest of the paper is organized as follows. In Sec. II, we explain the models we used and our methods for scoring them. We then present and discuss our results in Sec. III. Finally, we summarize in Sec. IV.

II. MODELS AND METHODS

A. Inputs and outputs of the IBUU simulator

For completeness and ease of our presentation, we briefly recall here the Bayesian+IBUU (GP) framework as well as the major inputs and outputs of the IBUU simulator used in Ref. [28]. Transport model simulations of heavy-ion reactions are computationally expensive and thus cannot be used directly in generating normally multimillion steps in the MCMC sampling of posterior probability distribution functions (PDFs) of model parameters in Bayesian analyses. Therefore, a GP emulator was used successfully in Ref. [28] to mimic predictions of the IBUU simulator in its simplest setup. As reviewed in detail in Refs. [20,21], the IBUU simulator normally has multiple choices for the baryon mean-field potential including its isospin and momentum dependence [44,45] as well as in-medium baryon-baryon cross sections [46]. In its first application within the Bayesian+IBUU (GP) framework, only the two most important input parameters (X and K) are randomly initialized in their prior uncertainty ranges while others are fixed at their currently known most probable values in each MCMC step, and no momentum-dependence of single nucleon potentials was considered [28]. The latter requires much more computing time but is very important for understanding several interesting features of heavy-ion collisions at intermediate energies. Once a faster and more reliable emulator is developed, this certainly should be considered. The two most important input parameters X and K for the IBUU simulator are defined as

- (i) The in-medium baryon-baryon scattering modification factor $X \equiv \frac{\sigma_{NN}^{\text{med}}}{\sigma_{NN}^{\text{free}}}$ where $\sigma_{NN}^{\text{free}}$ is the free-space cross section.
- (ii) The incompressibility quantifying the stiffness of the symmetric nuclear matter EOS $E_0(\rho)$ via $K = 9\rho_0^2(\frac{d^2E_0(\rho)}{d\rho^2})$ at its saturation density $\rho_0 = 0.16 \text{ fm}^{-3}$.

The goal of Ref. [28] was to infer the PDFs of X and K from the experimental observations of nuclear collective flow. The observables chosen were proton directed (or transverse) flow (v_1) and elliptical flow (v_2). They are coefficients of the Fourier decomposition of the particle azimuthal angle ϕ distribution $\frac{2\pi}{N} \frac{dN}{d\phi} = 1 + 2 \sum_{n=1}^{\infty} v_n \cos(n\phi)$, where ϕ is measured with respect to the reaction plane in x - o - z for a beam in the z direction. Their values at rapidity y and transverse momentum p_t can be

evaluated from $v_1(y, p_t) = \langle \cos(\phi) \rangle(y, p_t) = \frac{1}{n} \sum_{i=1}^n \frac{p_{ix}}{p_{ti}}$ and $v_2(y, p_t) = \langle \cos(2\phi) \rangle(y, p_t) = \frac{1}{n} \sum_{i=1}^n \frac{p_{ix}^2 - p_{iy}^2}{p_{ti}^2}$, where p_{ix} and p_{iy} are the x and y components of the i th particle momentum, respectively. In Ref. [28] and in the community generally, rather than using v_1 directly, its slope $F_1 = dv_1/dy'|_{y'=0}$ at mid-rapidity in the center-of-mass (cm) frame is often used to measure the strength of directed flow. We will only use F_1 from here on. In mid-central (10–30% centrality) Au+Au collisions at $E_{\text{beam}}/A=1.23 \text{ GeV}$ measured by the HADES Collaboration [29,30], $F_1 \equiv dv_1/dy'|_{y'=0} = 0.46 \pm 0.03$ and $v_2 = -0.06 \pm 0.01$ data for $|y_{\text{cm}}| \leq 0.05$ and $p_t \geq 0.3 \text{ GeV}/c$ were observed for free protons. These values are at approximately the maximum and minimum of the excitation functions of F_1 and v_2 , respectively, accumulated over the last 40 years [25]. Namely, the strengths of both directed flow (positive) and elliptic flow (negative) are the strongest around this beam energy. The F_1 and v_2 data from the HADES experiments are thus particularly interesting and informative. Indeed, these two experimental observations have already provided some strong constraints on the PDFs of X and K [28].

In generating the IBUU training and testing datasets, for each set of the X and K parameters generated on a Latin hyperlattice [43], we used 200 test particles/nucleon and 100 impact parameters b with their probability density given by $P(b) \propto b$ between $b = 6$ and 9 fm corresponding to approximately (10–30)% centrality. The accumulated 20 000 effective nucleus-nucleus collision events are used to evaluate the F_1 and v_2 for each X - K parameter set. Shown in Figure 1 are the F_1 and v_2 values for 89 X - K parameter sets used for training the GP emulator in Ref. [28]. This dataset will also be used in this work.

B. A deep neural network as an emulator of IBUU

A neural network is like a regression, except that it uses perceptrons. A perceptron is an individual neuron or point, that can have values varying between 0 and 1. The output is based on the activation function of the perceptron as well as its weights and biases. If the perceptron reaches a certain value or higher, based on its inputs, then it will activate. The neural network will automatically adjust the weights and biases of each perceptron during the training phase to converge on some set of values that can solve given inputs for their corresponding outputs. Neural networks have layers of perceptrons, starting with the input layer, where the data goes in; then hidden layers, which is where the computation happens and the user has little control while the machine calculates the best possible values of weights and biases; and ending up with the output layer. The goal of the neural network in training is to minimize the loss between its outputs and the actual outputs using a loss function.

We used two different, popular Python packages to create DNNs in order to determine which would be easiest and most efficient. The first is Scikit-Learn [47,48]. The second is TensorFlow [49], which runs on the Keras API [50]. It should be noted that TensorFlow is designed to be run on GPUs, but can be used with CPUs as was done in this study.

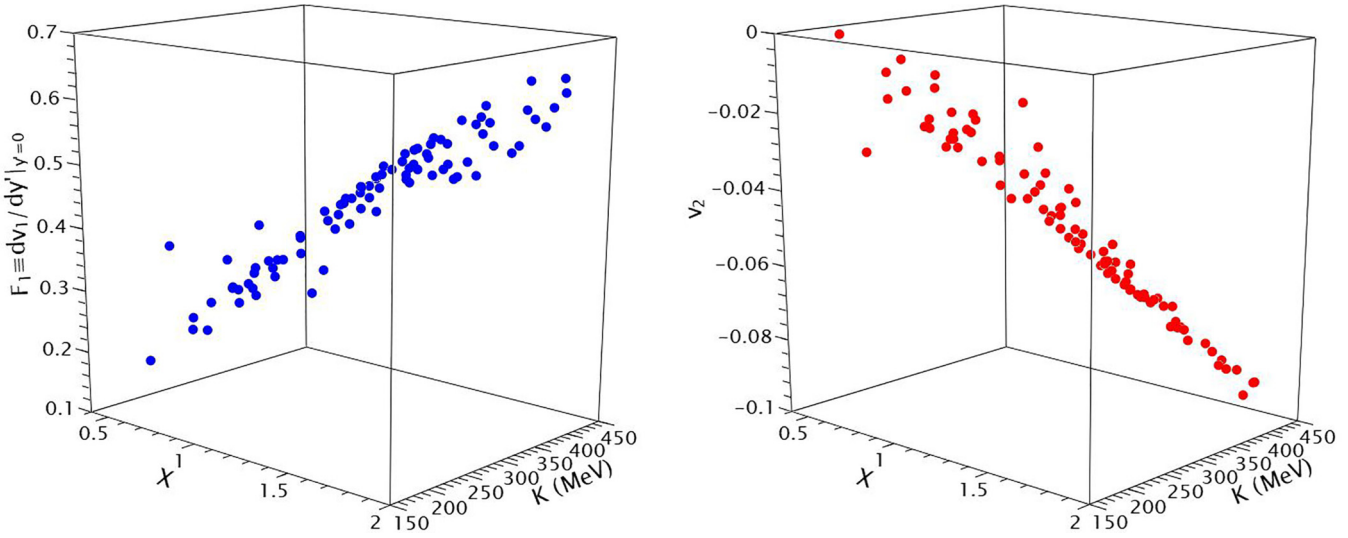


FIG. 1. 89 sets of predicted slope F_1 of the free proton directed flow at mid-rapidity $dv_1/dy'|_{y'=0}$ (left) and elliptical flow v_2 (right), respectively, as functions of the incompressibility K (right axis) and in-medium baryon-baryon scattering cross section modification factor X (left axis) generated on a Latin hyperlattice [43] in the X - K plane using the IBUU transport model for mid-central Au+Au reactions at 1.35 GeV/nucleon [28].

All models used four hidden layers as shown in Fig. 2. The number of perceptrons in the first and last hidden layers were chosen to match the number of inputs and outputs. The two middle layers each had six perceptrons, which allowed the DNN to vary more weights and biases in order to better optimize results. All layers are interconnected, i.e., dense. This means the output of each perceptron is passed to each perceptron in the next layer. Figure 2 shows this graphically. We set most layers to use the hyperbolic tangent function (tanh) as their activation function except the first and last hidden layers in the TensorFlow models, which allowed for more variability in the layers. For these, we used a linear activation function. Except where mentioned, all data were scaled using Scikit-Learn's StandardScaler (SS), which scales each feature (i.e., F_1 and v_2) separately to have a zero mean and unit variance. The scaled data were used for training the DNN and calculating R^2 ; for all other purposes we report the unscaled values. The Scikit-Learn

models used the multilayered perceptron regressor (MLPR) with the limited-memory Broyden-Fletcher-Goldfarb-Shanno optimizer (L-BFGS) [51], which, according to the Scikit-Learn User Guide, is well-designed for small datasets. The TensorFlow models used a Keras Sequential model with the Adam optimizer [52]. The loss function across all models was the mean squared error (MSE) between the DNN output $X_{\text{obs},i}$ and the expected/experimental/actual output $X_{\text{exp},i}$:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (X_{\text{obs},i} - X_{\text{exp},i})^2 \quad (1)$$

where n is the number of observations made. For the Scikit-Learn models, we manually adjusted the max number of iterations to optimize the DNN predictions as measured by R^2 :

$$R^2 = 1 - \frac{\text{SSE}}{\text{TSS}} \quad (2)$$

where SSE is the sum of squared errors (also sometimes written as RSS for the residual sum of squares) and TSS is the total sum of squares as discussed in Ref. [53]. For the TensorFlow models, we manually adjusted the number of training epochs for the same goal. All other model parameters were left at their default values. Data was split 75% for training and 25% for validation. To gauge the success of our DNN, we first got some analytical results to be used as references. As discussed in Ref. [28] and shown in Fig. 1, F_1 and v_2 have an approximately linear relationship with respect to X and K . As a check on this, we performed a multiple regression assuming a linear fit on the simulation data. The high R^2 corroborates the previous observation. Thus, it is expected that predicting F_1 or v_2 from X and K should not be difficult. To see if inferring the model parameters from the observables would be as trivial, the same linear multiple regression was done with F_1 and v_2 as the independent variables and X and K

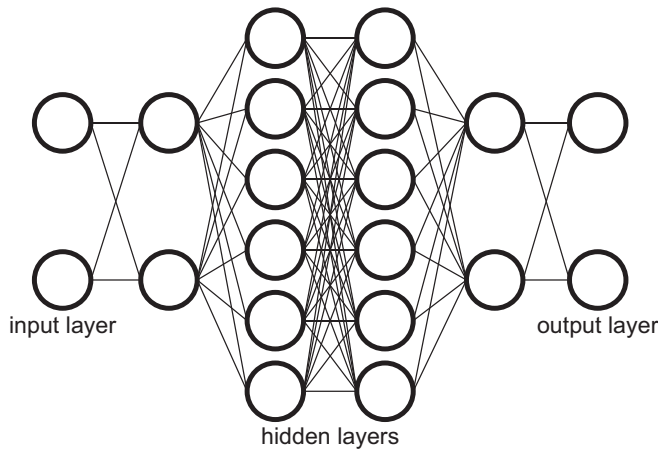


FIG. 2. Diagram of a DNN with four hidden layers.

TABLE I. Results of regression based on IBUU predictions

Equation	R^2
$F_1 = -0.0251 + 0.206X + 0.000663K$	0.927
$v_2 = 0.0519 - 0.0400X - 0.000158K$	0.959
$X = -0.130 + 2.92F_1 - 1.34v_2$	0.782
$K = 403 - 881F_1 - 6400v_2$	0.587

as the dependent variables. Not surprisingly, simply reversing the problem did not yield excellent results in terms of the R^2 values. Table I summarizes the results for all regressions. These will be used as the minimum success level for the DNN in our work. Ideally, the models will perform better (have a higher R^2 value) than these basic linear approximations.

III. RESULTS AND DISCUSSIONS

During our attempts to optimize the DNN structure, we noticed a great deal of inconsistency in our models' training. Without changing the code or DNN structure, the R^2 value could change significantly, as much as 0.4 with certain setups. Possible reasons for this are discussed later. Below we report our best results with quantified variations. Once we decided upon a DNN structure that produced high R^2 values most often, we ran the same code five times and picked the best. To quantify the consistency of each model, we report the range ΔR^2 of the DNN predictions compared to the IBUU simulations for the test dataset in Table II.

Also included in Table II is the average time to train the DNN model and the average time per prediction on a PC. While no training is prohibitively expensive to do once, prediction time is important. With over a million predictions, the difference between Scikit-Learn and TensorFlow is over half an hour, which is important to consider in an emulator. As a reference, we notice that Ref. [28] used 50 million steps (calls to the GP emulator) in their MCMC process. This would require a total of less than four minutes of computation time on a PC by the Scikit-Learn DNN. Thus, DNN can be computationally efficient enough to replace the GP.

A. Emulation

Figure 3 summarizes the results from the DNN for predicting F_1 and v_2 from X and K . For the Scikit-Learn model, instead of using SS, a different scaler was used: the Max

TABLE II. SL is for Scikit-Learn, TF is for TensorFlow, emu (emulation) is for the DNN models emulating the IBUU simulation, and inv (inversion) is for the DNN models that inferred X and K from F_1 and v_2 .

Model	ΔR^2	Train time (s)	Prediction time(s)
SL emu	0.109	5.44×10^{-2}	3.86×10^{-6}
TF emu	0.226	3.87	2.29×10^{-3}
SL inv	0.126	7.23×10^{-2}	3.96×10^{-6}
TF inv	0.312	5.79	2.24×10^{-3}

Absolute Scaler (MAS), which, according to the Scikit-Learn User Guide, is better for sparse data because it does not shift or center the data. Rather, it divides each feature by its maximum absolute value. We did not use it regularly, however, because in most cases it failed to improve performance and caused the TensorFlow models to become even more inconsistent. For these results, the maximum number of iterations was set to 80, an unusually low value. This prevented the model from converging every time, but allowing more time often caused overfitting. The TensorFlow model was given 150 epochs to train. The high R^2 values show that both the Scikit-Learn and TensorFlow models improved on the linear regression.

To quantify the errors of the emulators, we define here the DNN error for an arbitrary variable X as

$$\text{DNN error}_{\text{run}}(x) = \frac{[X(\text{DNN})_{\text{run}} - X(\text{IBUU})_{\text{run}}]^2}{\sigma_X^2(\text{DNN}) + \sigma_X^2(\text{IBUU})}, \quad (3)$$

where $\sigma_X^2(\text{DNN})$ and $\sigma_X^2(\text{IBUU})$ are the variances of X from the DNN emulator and IBUU simulator, respectively. In essence, it measures the individual MSE of each variable X with respect to the total error of simulations and emulations involved, enabling a more fair comparison of the accuracies obtained with different approaches. In Fig. 4, we report the DNN error of the observables F_1 and v_2 , for each point shown in Fig. 3. Comparing the DNN errors obtained with the Scikit-Learn and TensorFlow, the latter generally has larger errors in predicting both F_1 and v_2 . More quantitatively, the largest DNN error for F_1 (v_2), indicating the point the DNN has the weakest predictive power, is 2.64×10^{-2} (6.30×10^{-2}) for Scikit-Learn versus 8.22×10^{-2} (1.31×10^{-1}) for TensorFlow. The absolute MSE values in predicting F_1 and v_2 are also indicated in Fig. 4. The results show strong signs of DNN being able to accurately emulate the IBUU simulator's predictions even with the small training dataset. Moreover, by comparing the DNN's MSE values with that of GP (Fig. 3 of Ref. [28]) trained by using the same set of IBUU predictions, we notice that overall the DNN is about ten times more accurate. More quantitatively, as shown in Fig. 4, the DNN total MSE is 1.19×10^{-4} with Scikit-Learn and 2.17×10^{-4} with TensorFlow, while the GP total MSE shown in Fig. 3 of Ref. [28] is 9.20×10^{-4} .

B. Inversion

Unsurprisingly, the direct inversion of the problem did not produce such stunning results, and, unfortunately, the models did not make significant improvement over the simple linear multiple regression. Figure 5 summarizes the results of trying to infer X and K from F_1 and v_2 . Shown in Fig. 6 are the corresponding DNN errors in predicting X and K , respectively, for each point shown in Fig. 5. While the Scikit-Learn models were again stopped short of convergence at 150 iterations, the TensorFlow models actually did best at a slightly longer 250 epochs. It can be seen qualitatively that the models did a worse job predicting K than they did X . The R^2 values are the same because they score the entire model, not X and K predictions separately. That result is not surprising, however, given that the values for R^2 from the multiple regression were much

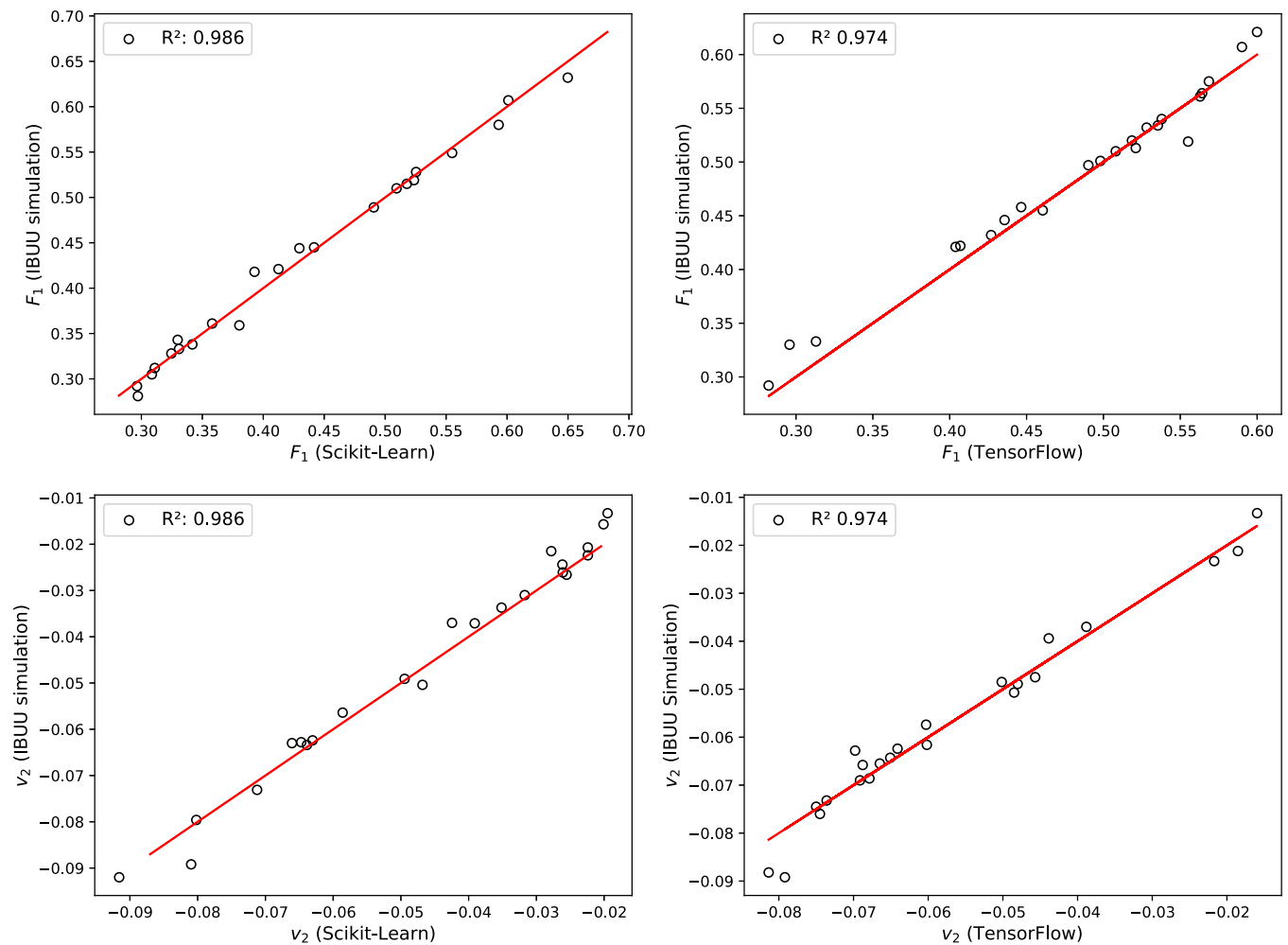


FIG. 3. DNN predictions are black circles. The line of the perfect match between the DNN predictions and the IBUU simulations is red.

smaller for K than for X . It is also seen that the Scikit-Learn and TensorFlow models have roughly the same errors for inferring both X and K .

In the inversion problems, all models had poor accuracy, so concrete conclusions are difficult. Two possibilities for the issue seem likely. The first is that F_1 and v_2 simply do

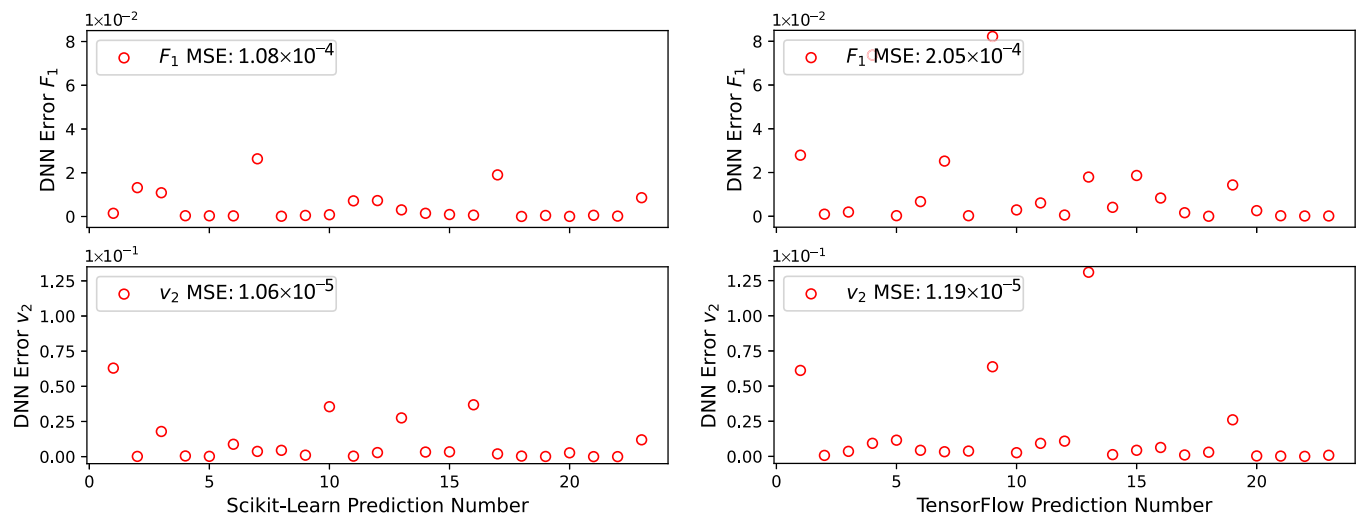
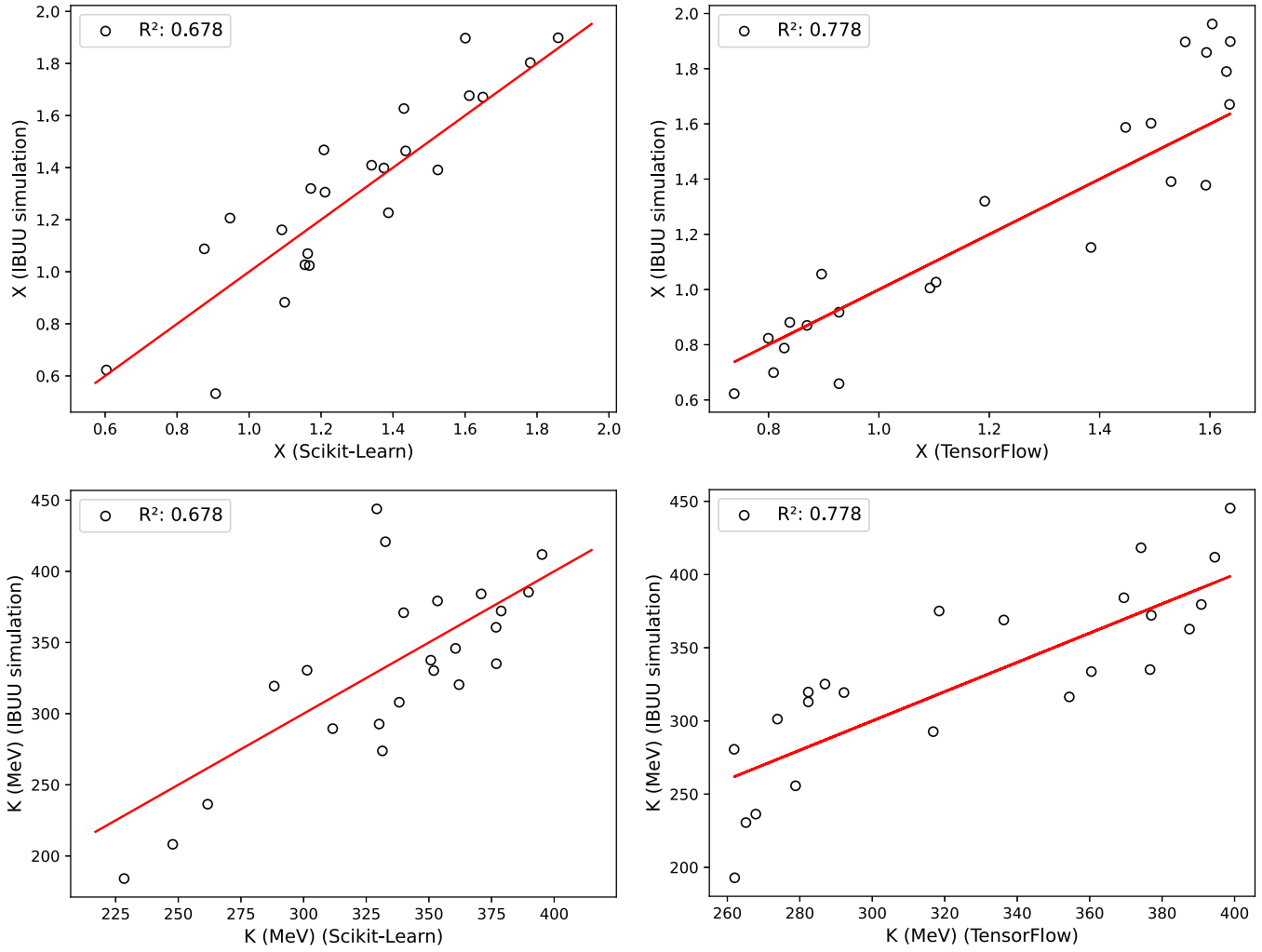


FIG. 4. DNN error for predicting F_1 and v_2 for each point in the test dataset shown in Fig. 3.


FIG. 5. Same as Fig. 3, but now for X and K (MeV).

not carry enough information to constrain X and K tightly. It is well known that a degeneracy (i.e., combinations of small/big K with big/small X have the same physical effects)

exists between these two parameters in producing nuclear collective flow (see, e.g., Refs. [54–64]), which complicates inference problems. This degeneracy arises because the

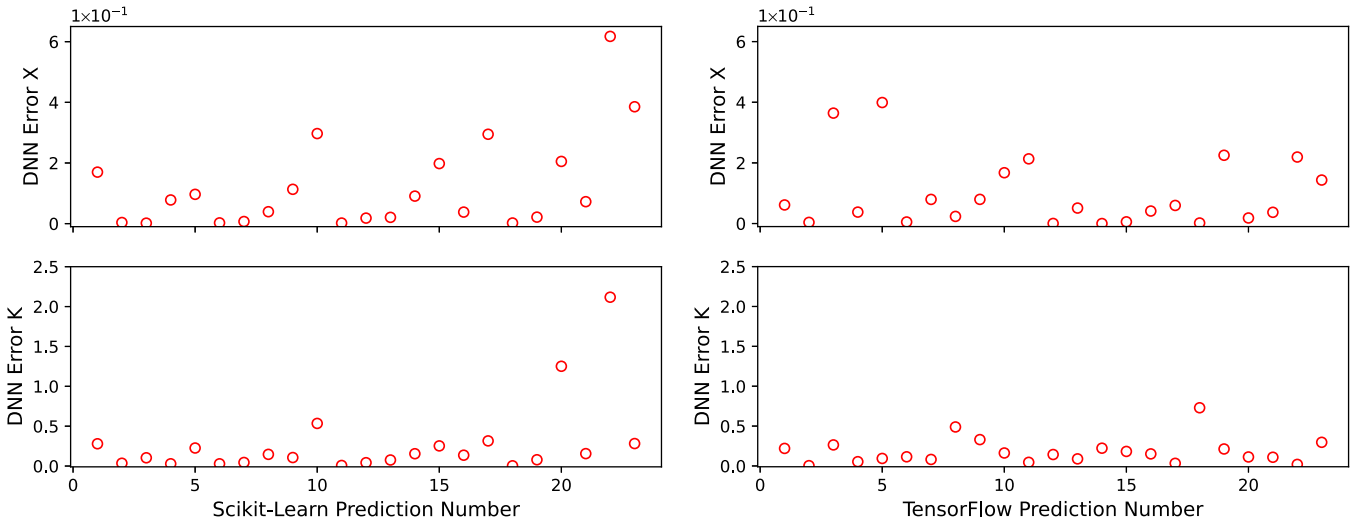

FIG. 6. DNN error for predicting X and K , for each point shown in Fig. 5.

TABLE III. Maximum R^2 value achieved in five runs of each DNN setup with training/testing splits of 50/50 and 90/10.

$R^2_{\max}$	50/50	90/10
SL emu	0.958	0.969
TF emu	0.930	0.968
SL inv	0.677	0.656
TF inv	0.655	0.786

nuclear collective flow is generated by the pressure gradient during nucleus-nucleus collisions. And the pressure gradient can be created by either a relatively long-range nuclear force through a mean-field potential (its stiffness is characterized by K) or a short-range nuclear force through nucleon-nucleon scattering (its frequency and effect are characterized by X). Perhaps some other observable could help. This explanation would also account for the low R^2 values from the multiple regressions. The second reason is a lack of training data. While it appears to be possible to train DNN for emulation on

only 89 data points, it is possible that more data are required for the more complex inversion problem. An idea for getting around the lack of IBUU simulation data is to first train a DNN as an emulator to produce large amounts of data, and then use this to train the inversion network. One danger in this is that any small error at the beginning of the process could be propagated and compounded through several models. Future work could assess the feasibility of this and quantitatively evaluate the systematic error. The above experience gained in using directly the DNN as an inversion tool is very educational. Nevertheless, we emphasize that the value of DNN designed here is not for direct inversions but as an emulator for solving inverse problems within the Bayesian+DNN framework.

C. Effects of training/testing data splitting

Up to this point, we have used 75% of the 89 data points for training all of our models and 25% to test their accuracy. This 75/25 split, or similar, such as 80/20 or 70/30, is commonly used for designing DNN, but it is not a firm rule; see, e.g., Ref. [65] and references therein.

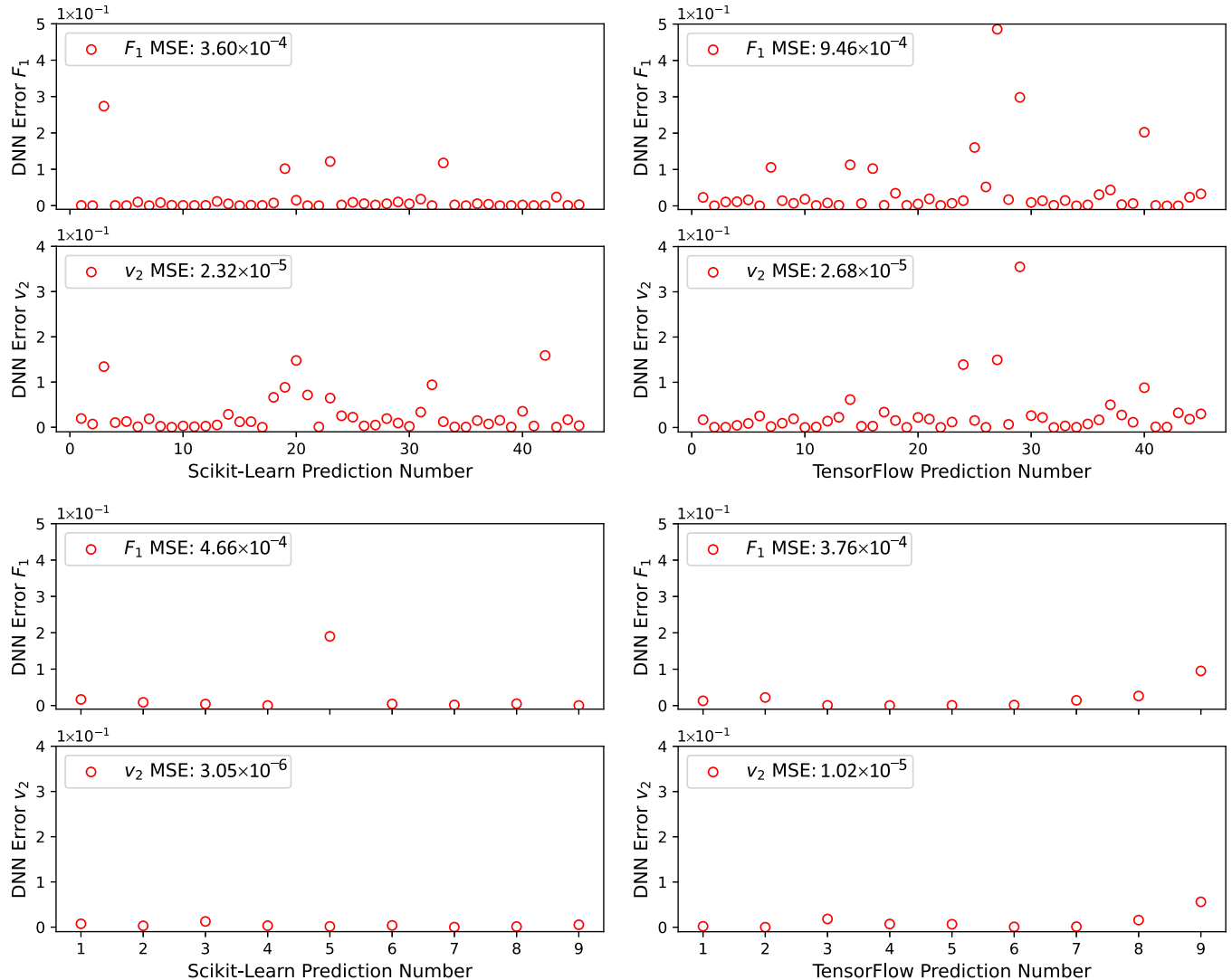


FIG. 7. DNN error for predicting F_1 and v_2 with a 50/50 split (top) and a 90/10 split (bottom).

To determine if a different splitting ratio could improve our results, we tried two other scenarios: 50/50 and 90/10 training/testing ratios. In both cases, consistency remained elusive, so each was run five times. We report the maximum R^2 value each model achieved in Table III.

All attempts to infer X and K with the DNN continued to produce poor results, so we will not discuss them further. Beginning with the 50% training, 50% testing case, we note that neither Scikit-Learn nor TensorFlow achieved R^2 values very close to our previous results, and this is reflected in the absolute MSE values shown in Fig. 7. The total MSE for Scikit-Learn is 3.83×10^{-4} , more than three times worse than before, and 9.73×10^{-4} for TensorFlow, which is now on the order of the GP used in Ref. [28]. This demonstrates that DNN can replace a GP even with only half the training data. Greater accuracy, however, still requires providing most of the data to the DNN for training.

In the case of 90% training, 10% testing, both models were able to reproduce their previous success. In fact, Scikit-Learn achieved a MSE for v_2 significantly smaller than before, 3.05×10^{-6} down from 1.06×10^{-5} . However, F_1 demonstrates a possible issue with using this large training to testing ratio. The absolute MSE is up slightly, but more importantly the DNN error outlier for Scikit-Learn (TensorFlow) is 1.90×10^{-1} (9.53×10^{-2}). For TensorFlow that is a small increase from 6.30×10^{-2} , but for Scikit-Learn the new outlier is more than five times its previous value of 2.64×10^{-2} . Thus, using a higher percentage of data for training can improve some results at the risk of having some extreme outliers.

This is by no means an exhaustive analysis of training to testing ratios. It demonstrates, however, that the frequently used 75/25 data splitting does indeed work well in our case.

D. Caveats and future perspectives

A Gaussian process typically provides two functions: accurate predictions of a simulation and an estimated error for each prediction, i.e., mean and variance. In this study, DNNs were able to make accurate predictions. This is more impressive given the small dataset, and useful because of the long simulation times to create even a few data points. Unfortunately, the models used here were not capable of providing a variance for each prediction. The best method for accomplishing this is still an active topic of investigation, with possibilities ranging from measuring Monte Carlo dropout (MC-dropout) [66,67] to deep ensembles [68] to mean-variance estimation neural networks [69,70]. Future work should explore if any of these methods or others can be easily implemented without significantly increasing the training time, which currently takes only a few seconds for the models used here. For applications that do not require variance predictions, both Scikit-Learn and TensorFlow can be used to implement a quickly trainable DNN for emulations.

Above we have reported our best results with quantified variations, from the models that gave strong R^2 often, but some models behaved inconsistently. This is mainly due to our choice of randomly splitting the data (for a chosen training/testing ratio) and allowing the initial weights and biases of each model to initialize randomly *each time*. We chose

this method because picking the optimal training/testing split would require prior knowledge that experiments cannot provide. As far as the DNN is concerned, the data *are* random. Ideally, the DNN could be accurately trained regardless of which data are used for training and which for testing. We think that, with a dataset this small, even slight differences in the training/testing split can lead to a very inconsistent DNN. Oddly, using MAS to scale the data worsened this for the TensorFlow models despite decreasing the loss function, which is why we did not use it. This is a very frustrating setback after managing to avoid over-fitting, the most often cited problem for small datasets.

An additional challenge was the regular manual adjustments. In order to produce optimal results, we experimented with different data scalers, layer structures, activation functions, optimizers, numbers of iterations and epochs, and more. This manual adjustment allowed us to better learn and understand various features of DNN. For larger projects, however, it may be worth using an automated method such as another neural network that optimizes the hyperparameter setup for DNN as was done in [19], which used Scikit-Learn, or with the Keras Tuner [71] for TensorFlow.

IV. SUMMARY

To summarize, we found deep neural networks can accurately predict the values of the nuclear flow parameters F_1 and v_2 given the two input parameters X and K , even though only a small training dataset was provided. Moreover, it is about ten times more accurate than the GP emulator trained with the same dataset.

The simplest of DNN models, however, is unable to fully replace a Gaussian emulator because it does not provide a variance prediction. The DNN used here was also incapable of solving directly the inverse problem to accurately infer X and K from F_1 and v_2 alone. On the other hand, the value of a faster and more reliable DNN emulator lies in its potential use in solving the inverse reaction problem within a Bayesian+DNN framework to infer the PDFs of model parameters.

In the forward modeling with transport models, the whole phase space distribution functions of all particles in the final state of heavy-ion reactions are predicted. One can construct many observables to be compared with the experimental data in a Bayesian framework. To emulate multiple observables, one may need to generalize the simple DNN studied here to a committee of DNNs. Moreover, given a set of input parameters, transport model predictions for final observables are unique within statistical fluctuations. However, it is generally degenerate in the inverse process to go from a few final state observables to the input model parameters. Unraveling links between final state observables with input model parameters with inversion techniques has been a longstanding and challenging goal for the heavy-ion reaction community. Statistical inversions within the Bayesian framework using faster emulators for transport model simulators have been shown by the community to be promising. Our findings and experiences from constructing and testing a DNN as a faster and more reliable emulator of heavy-ion reactions at intermediate energies contribute positively to efforts in this direction.

ACKNOWLEDGMENTS

We would like to thank J. Richter for allowing us to use his NN code to learn about neural networks and for giving us a reference point to see if ours worked. Many thanks also go

to P. Krastev, W.-J. Xie, and K. Zhou for helpful discussions and communications. This research was sponsored in No. by the US Department of Energy under Award Number DE-SC0013702 and the TAMU-Commerce Division of Research and Economic Development.

- [1] C. Andrieu, N. de Freitas, A. Doucet *et al.*, An introduction to MCMC for machine learning, *Mach. Learn.* **50**, 5 (2003).
- [2] S. Gazula, J. W. Clark, and H. Bohr, Learning and prediction of nuclear stability by neural networks, *Nucl. Phys. A* **540**, 1 (1992).
- [3] A. Boehnlein *et al.*, Colloquium: Machine learning in nuclear physics, *Rev. Mod. Phys.* **94**, 031003 (2022).
- [4] K. Zhou, L. Wang, L. G. Pang *et al.*, Exploring QCD matter in extreme conditions with machine learning, *Prog. Part. Nucl. Phys.* **135**, 104084 (2023).
- [5] W. He, Q. Li, Y. Ma *et al.*, Machine learning in nuclear physics at low and intermediate energies, *Sci. China Phys. Mech. Astron.* **66**, 282001 (2023).
- [6] S. A. Bass, A. Bischoff, C. Hartnack *et al.*, Neural networks for impact parameter determination, *J. Phys. G* **20**, L21 (1994).
- [7] Y. Wang, Z. Gao, H. Lu *et al.*, Decoding the nuclear symmetry energy event-by-event in heavy-ion collisions with machine learning, *Phys. Lett. B* **835**, 137508 (2022).
- [8] Z. X. Yang, X. H. Fan, Z. P. Li *et al.*, A neural network approach for orienting heavy-ion collision events, *Phys. Lett. B* **848**, 138359 (2024).
- [9] R.-D. Lasserri, D. Regnier, J.-P. Ebran, and A. Penon, Taming nuclear complexity with a committee of multilayer neural networks, *Phys. Rev. Lett.* **124**, 162502 (2020).
- [10] Z. A. Wang, J. Pei, Optimizing multilayer Bayesian neural networks for evaluation of fission yields, *Phys. Rev. C* **104**, 064608 (2021).
- [11] A. Sarkar and D. Lee, Self-learning emulators and eigenvector continuation, *Phys. Rev. Res.* **4**, 023214 (2022).
- [12] A. E. Lovell, A. T. Mohan, T. M. Sprouse, and M. R. Mumpower, Nuclear masses learned from a probabilistic neural network, *Phys. Rev. C* **106**, 014305 (2022).
- [13] O. M. Molchanov, K. D. Launey, A. Mercenne, G. H. Sargsyan, T. Dytrych, and J. P. Draayer, Machine learning approach to pattern recognition in nuclear dynamics from the *ab initio* symmetry-adapted no-core shell model, *Phys. Rev. C* **105**, 034306 (2022).
- [14] M. Knöll, T. Wolfgruber, M. L. Agel *et al.*, Machine learning for the prediction of converged energies from *ab initio* nuclear structure calculations, *Phys. Lett. B* **839**, 137781 (2023).
- [15] G. P. A. Nobre, D. A. Brown, S. J. Hollick, S. Scoville, and P. Rodríguez, Novel machine-learning method for spin classification of neutron resonances, *Phys. Rev. C* **107**, 034612 (2023).
- [16] Y. L. Yang and P. W. Zhao, Deep-neural-network approach to solving the *ab initio* nuclear structure problem, *Phys. Rev. C* **107**, 034320 (2023).
- [17] N. Hizawa, K. Hagino, and K. Yoshida, Analysis of a Skyrme energy density functional with deep learning, *Phys. Rev. C* **108**, 034311 (2023).
- [18] D. Lay, E. Flynn, S. A. Giuliani, W. Nazarewicz, and L. Neufcourt, Neural network emulation of spontaneous fission, *Phys. Rev. C* **109**, 044305 (2024).
- [19] M. F. Kasim, D. Watson-Parris, L. Deaconu *et al.*, Building high accuracy emulators for scientific simulations with deep neural architecture search, *Mach. Learn. Sci. Technol.* **3**, 015013 (2022).
- [20] B. A. Li, C. M. Ko, and W. Bauer, Isospin physics in heavy ion collisions at intermediate-energies, *Int. J. Mod. Phys. E* **07**, 147 (1998).
- [21] B. A. Li, L. W. Chen, and C. M. Ko, Recent progress and new challenges in isospin physics with heavy-ion reactions, *Phys. Rep.* **464**, 113 (2008).
- [22] P. Danielewicz and G. Odyniec, Transverse momentum analysis of collective motion in relativistic nuclear collisions, *Phys. Lett. B* **157**, 146 (1985).
- [23] J. Y. Ollitrault, Flow systematics from SIS to SPS energies, *Nucl. Phys. A* **638**, 195c (1998).
- [24] A. M. Poskanzer and S. A. Voloshin, Methods for analyzing anisotropic flow in relativistic nuclear collisions, *Phys. Rev. C* **58**, 1671 (1998).
- [25] A. Sorensen *et al.*, Dense nuclear matter equation of state from heavy-ion collisions, *Prog. Part. Nucl. Phys.* **134**, 104080 (2024).
- [26] H. Wolter *et al.*, Transport model comparison studies of intermediate-energy heavy-ion collisions, *Prog. Part. Nucl. Phys.* **125**, 103962 (2022).
- [27] C. Rasmussen and C. C. K. I. Williams, *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, 2006).
- [28] B. A. Li and W. J. Xie, Bayesian inference of in-medium baryon-baryon scattering cross sections from HADES proton flow data, *Nucl. Phys. A* **1039**, 122726 (2023).
- [29] J. Adamczewski-Musch *et al.*, Directed, Elliptic, and higher order flow harmonics of protons, deuterons, and tritons in Au + Au collisions at $\sqrt{s_{NN}} = 2.4$ GeV, *Phys. Rev. Lett.* **125**, 262301 (2020).
- [30] J. Adamczewski-Musch *et al.*, Proton, deuteron and triton flow measurements in Au+Au collisions at $\sqrt{s_{NN}} = 2.4$ GeV, *Eur. Phys. J. A* **59**, 80 (2023).
- [31] J. Novak, K. Novak, S. Pratt, J. Vredevoogd, C. E. Coleman-Smith, and R. L. Wolpert, Determining fundamental properties of matter created in ultrarelativistic heavy-ion collisions, *Phys. Rev. C* **89**, 034917 (2014).
- [32] J. E. Bernhard, P. W. Marcy, C. E. Coleman-Smith, S. Huzurbazar, R. L. Wolpert, and S. A. Bass, Quantifying properties of hot and dense QCD matter through systematic model-to-data comparison, *Phys. Rev. C* **91**, 054910 (2015).
- [33] S. Pratt, E. Sangaline, P. Sorensen, H. Wang, Constraining the equation of state of superhadronic matter from heavy-ion collisions, *Phys. Rev. Lett.* **114**, 202301 (2015).
- [34] J. E. Bernhard, J. S. Moreland, S. A. Bass, J. Liu, and U. Heinz, Applying Bayesian parameter estimation to relativistic heavy-ion collisions: Simultaneous characterization of the initial state and quark-gluon plasma medium, *Phys. Rev. C* **94**, 024907 (2016).

- [35] E. Sangaline and S. Pratt, Toward a deeper understanding of how experiments constrain the underlying physics of heavy-ion collisions, *Phys. Rev. C* **93**, 024908 (2016).
- [36] J. E. Bernhard, J. S. Moreland, and S. A. Bass, Bayesian estimation of the specific shear and bulk viscosity of quark-gluon plasma, *Nat. Phys.* **15**, 1113 (2019).
- [37] B. Weiss, J. F. Paquet, and S. A. Bass, Computational budget optimization for Bayesian parameter estimation in heavy-ion collisions, *J. Phys. G: Nucl. Part. Phys.* **50**, 065104 (2023).
- [38] M. Omana Kuttan, J. Steinheimer, K. Zhou, and H. Stoecker, QCD equation of state of dense nuclear matter from a Bayesian analysis of heavy-ion collision data, *Phys. Rev. Lett.* **131**, 202303 (2023).
- [39] M. Heffernan, C. Gale, S. Jeon, and J. F. Paquet, Bayesian quantification of strongly interacting matter with color glass condensate initial conditions, *Phys. Rev. C* **109**, 065207 (2024).
- [40] J. M. Wang, X. G. Deng, W. J. Xie *et al.*, Bayesian inference of nuclear incompressibility from proton elliptic flow in central Au+Au collisions at 400 MeV/nucleon, [arXiv:2406.07051](https://arxiv.org/abs/2406.07051).
- [41] D. R. Phillips *et al.* (BAND Collaboration), Get on the BAND wagon: A Bayesian framework for quantifying model uncertainties in nuclear dynamics, *J. Phys. G* **48**, 072001 (2021).
- [42] W. Fan *et al.* (JETSCAPE Collaboration), New metric improving Bayesian calibration of a multistage approach studying hadron and inclusive jet suppression, *Phys. Rev. C* **109**, 064903 (2024).
- [43] M. D. McKay, R. J. Beckman, and W. J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, *Technometrics* **42**, 55 (2000).
- [44] B. A. Li, C. B. Das, S. Das Gupta, and C. Gale, Momentum dependence of the symmetry potential and nuclear reactions induced by neutron rich nuclei at RIA, *Phys. Rev. C* **69**, 011603(R) (2004).
- [45] B. A. Li, C. B. Das, and S. Das Gupta *et al.*, Effects of momentum dependent symmetry potential on heavy ion collisions induced by neutron rich nuclei, *Nucl. Phys. A* **735**, 563 (2004).
- [46] B. A. Li and L. W. Chen, Nucleon-nucleon cross sections in neutron-rich matter and isospin transport in heavy-ion reactions at intermediate energies, *Phys. Rev. C* **72**, 064611 (2005).
- [47] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, Scikit-Learn: Machine learning in Python, *J. Mach. Learn. Res.* **12**, 2825 (2011).
- [48] L. Buitin *et al.*, J. Louppe, and M. Blondel *et al.*, API design for machine learning software: experiences from the scikit-learn project, in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, (Springer, Berlin, 2013), pp. 108–122.
- [49] M. Abadi, A. Agarwal, and P. Barham *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from <http://www.tensorflow.org>, 2015.
- [50] F. Chollet, Keras, <https://github.com/fchollet/keras> 2015.
- [51] D. C. Liu and J. Nocedal, On the limited memory BFGS method for large scale optimization, *Math. Program.* **45**, 503 (1989).
- [52] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [53] J. Richter and B. A. Li, Empirical radius formulas for canonical neutron stars from bidirectionally selecting features of equations of state in extended Bayesian analyses of observational data, *Phys. Rev. C* **108**, 055803 (2023).
- [54] G. F. Bertsch, G. E. Brown, V. Koch, and B. A. Li, Pion collectivity in relativistic heavy ion collisions, *Nucl. Phys. A* **490**, 745 (1988).
- [55] Y. M. Zheng, C. M. Ko, B. A. Li, and B. Zhang, Elliptic flow in heavy ion collisions near the balance energy, *Phys. Rev. Lett.* **83**, 2534 (1999).
- [56] B. A. Li and A. T. Sustich, Differential flow in heavy ion collisions at balance energies, *Phys. Rev. Lett.* **82**, 5004 (1999).
- [57] P. Danielewicz, Hadronic transport models, *Acta Phys. Pol. B* **33**, 45 (2002).
- [58] B. A. Li, B. J. L. W. Chen, and J. Xu, Nucleon effective masses in neutron-rich matter, *Prog. Part. Nucl. Phys.* **99**, 29 (2018).
- [59] T. Gaitanos, C. Fuchs and H. H. Wolter, Nuclear stopping and flow in heavy ion collisions and the in-medium NN cross section, *Phys. Lett. B* **609**, 241 (2005).
- [60] Y. Zhang, Z. Li, and P. Danielewicz, In-medium NN cross sections determined from stopping and collective flow in intermediate-energy heavy-ion collisions, *Phys. Rev. C* **75**, 034615 (2007).
- [61] P. Li, Y. Wang, Q. Li, C. Guo, and H. Zhang, Effects of the in-medium nucleon-nucleon cross section on collective flow and nuclear stopping in heavy-ion collisions in the Fermi-energy domain, *Phys. Rev. C* **97**, 044620 (2018).
- [62] P. Li, Y. Wang, Q. Li, and H. Zhang, Accessing the in-medium effects on nucleon-nucleon elastic cross section with collective flows and nuclear stopping, *Phys. Lett. B* **828**, 137019 (2022).
- [63] D. Persram and C. Gale, Elliptic flow in intermediate-energy heavy ion collisions and in-medium effects, *Phys. Rev. C* **65**, 064611 (2002).
- [64] C. Fuchs, A. Faessler, and M. El-Shabshiry, Off-shell behavior of the in-medium nucleon-nucleon cross-section, *Phys. Rev. C* **64**, 024003 (2001).
- [65] J. V. Roshan, Optimal ratio for data splitting, *Stat. Anal. Data Min. ASA Data Sci. J.* **15**, 2022 (2022).
- [66] J. Postels, F. Ferroni, H. Coskun *et al.*, Sampling-free epistemic uncertainty estimation using approximated variance propagation, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (IEEE, Piscataway, NJ, 2019), pp. 2931–2940.
- [67] Y. Gal and Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning (unpublished).
- [68] B. Lakshminarayanan, A. Pritzel, and C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, [arXiv:1612.01474](https://arxiv.org/abs/1612.01474).
- [69] D. Nix and A. Weigend, Estimating the mean and variance of the target probability distribution, in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)* (IEEE, Piscataway, NJ, 1994), Vol. 1, pp. 55–60.
- [70] L. Sluijterman, E. Cator, and T. Heskes, Optimal training of mean variance estimation neural networks, [arXiv:2302.08875](https://arxiv.org/abs/2302.08875).
- [71] T. O'Malley, E. Bursztein, and J. Long *et al.*, Kerastuner, <https://github.com/keras-team/keras-tuner>, 2019.