* In one respect, the findings of this study are not particularly surprising. It has been known for a long time that multi-layer feed-forward neural networks offer an extremely powerful alternative to traditional interpolation methods, with i.e. the use of neural networks to parametrise deep-inelastic structure functions having been stablished 20 years ago, with subsequent applications becoming a basically mainstream technique in global analyses of non-perturbative QCD quantities from (nuclear) parton distributions to fragmentation functions. Similar techniques have been applied to speed up event generation in Monte Carlo programs as well as to facilitate the evaluation of complex high-dimensional functions that arise in the context of perturbative QCD and electroweak calculation.

Although on the one hand our study confirms the growing results of the success of neural network interpolation, it also shows that they are not always the best choice for precision, especially in low dimensions. Furthermore, we believe that a systematic study such as this one provides a strong anchor for justifying the use of one approximant over the other for interpolation with large training data.

* I would argue that at this point it is the choice of a traditional interpolation algorithm that should be justified, while the use of neural networks are universal unbiased interpolants to parametrise functions in HEP is more or less an off-the-shelf technique. While the authors mention some of these results in their introduction, it may be a bit more accurate to emphasize that by now these are more or less standard techniques in HEP, and that it has been acknowledge that the use of traditional interpolation is restricted to a problems of certain simplicity. In particular, the statement "Although the previous analyses show promise in their respective methods" should be removed: there is a very large body of scientific work that demonstrates that NN-based methods outperform traditional interpolation techniques for all applications except those too simple to present a major bottleneck in any physics study. I believe that stating this in the introduction would reflect better the state of the art in the field. In other works, while the general answer of when it is worth moving to ML-based interpolation methods depends indeed on the problem, it is almost always found that when quality and performance of interpolation starts to limit how far we go into a given problem, it is the point while moving to ML-based tools is the right choice.

We have modified our introduction to reflect the point raised above.

* In this respect, I think the title does not reflect accurately the contents of the paper. The examples the authors consider are not representative of the whole field of HEP but instead are focused on functions that appear in the context of higher-order calculations. So I think it would be appropriate if the authors modified the title to better reflect their focus.

A more precise title that we have chosen is “Comparing Machine Learning and Interpolation Methods for Loop-Level Calculations”

* I also think that one should mention that in many physics problems that appear in HEP the functions to be parametrised have some physical interpretation, and for instance they are expected to be continuous and smooth. So the general problem of function interpolation in HEP should also address this point. I mention this because the authors try some interpolators such as nearest neighbours which are discontinuous, and hence cannot be applied to some problems of relevance in HEP. For the higher-order QCD functions that the authors study in this study this is

probably since since in these case they cannot be associated a physical interpretation, but this is only a subset of the relevant applications of ML-based regression for HEP problems. It would be interesting to consider how results change if some instance one imposes some smoothness criterion on the parametrised functions (my hunch is that if anything this requirement will further strengthen the superior performance of the MLP method). So address this point, would be interesting to add figures of merit such as arc-length to the ones already considered in the paper.

Indeed nearest neighbors is discontinuous, but it was chosen as a baseline interpolant that is inexpensive to implement. Imposing a smoothness criterion would be interesting for physically-interpretable functions, however we don't believe it is necessary when focusing on higher-order integral functions for loop calculations.

* Another reason why it should not surprise anyone that ML-based regressors outperform traditional interpolation techniques in high dimensions is that the latter are designed in general for low-dimensional problems, and simply cannot be trusted to work in high dimensions. For example, techniques like chebyshev polynomial interpolation are based on fitting the coefficients of the chebyshev polynomial expansion to the input data, and such fits are quite instable in high-dimensions due to the cancellations between different terms in the expansion. This problem is absent in ML-based methods, where by construction one starts with an smooth function which is then via the training adjusted to describe as well as possible the data.

The reviewer rightly points out that ML methods are by design well-suited for higher dimensions, our results however show that it is not a guarantee that ML methods will outperform RBF for example in 9 dimensions. Neural networks on the other hand do show superiority across all considered functions. Thus, our results are not completely in-line with the ML-superiority view in high dimensions.

* Yes another relevant topic that I am a but surprised the authors do not consider is that of overlearning. In general NN-based regressors are overly flexible and the input data is never "perfect", but will fluctuate around some true value. Given sufficiently large training times, the NN model will learn these fluctuations and thus deviate systematically from this underlying law. Without considering this possible issue in detail it is not possible to draw any solid conclusion about the performance of ML regressors. I understand that for this specific application the data is "perfect", but I would encourage the authors to consider the case where some fluctuations are added to the data, and reassess their analysis in that case.

We are aware of the possibility of overfitting, which is why we reserve a validation dataset and monitor its behavior. We mention in the cross-validation section "During training, we use the validation data to monitor the performance of the model on data it hasn't been trained on. This allows us to stop training when our models start to overfit." Furthermore, the amount of training data we use far exceeds the parameters in the neural network for example, so the chances of overfitting are low.

* The choice of training algorithm and of the model hyperparameters is also very important in order to assess the performance of ML-based regressors. A poor choice for example of NN architecture may lead to a model under-performing, but the conclusion here is not that the ML regressors is not appropriate but that the method adopted to select the model hyperparameters is non-optimal. Likewise, a wrong choice of training algorithm (or even of family of algorithm,

from SGD-like to GA-like techniques) can complicate the interpretation of the results and the benchmarking of the performance of ML regressors as compared to traditional interpolation techniques.

* The authors state that "performing hyperparameter optimization is computationally expensive so we rely on empirical tests to guide the settings" which in practice means setting the hyperparameters of the model by trial and error. Given that the whole point of this study is to robustly estimate the performance of ML regressors, I believe that the authors should investigate in a bit more systematic way how the choice of model hyperparameters affect the findings of their work. At the very least, the choice of adopted hyperparameters should be better justified; results for different sets of hyperparameters compared; and results using different minimisers (at least for the MLP analysis) benchmarked.

We vary hyperparameters and choose the best performing ones. For example, we fix the nodes, layers, and loss function, then vary the activation between ELU and GELU and find that they either have similar performance or GELU outperforms ELU, so we choose GELU. Similarly, we fix all but the loss function and compare MSE, MAE, and MAPE, then choose the best performing. We also check SGD vs Adam and find Adam outperforms SGD consistently. This method does not guarantee to find the optimal hyperparameters in the large space of possibilities, but it provides a practical way of finding a good set of hyperparameters.

We keep track of all the runs where we record relevant parameters and figures of merit. For example, for the neural network we record the nodes, layers, epochs, batch size, activation, loss function, optimizer, # of training points, # of testing points, and the figures of merit. We keep similar records for SVGP and LGBM.

* For the reasons stated above, I am surprised that they use a "EarlyStopping callback that stops training when no improvement has been made over 400 epochs". This seems to me in general a recipe for overlearning in practical applications (not in toy scenarios of course). The authors should investigate how the presence of noise in their input data (for example coming from MC fluctuations) affects the outcome of their analysis.

This might be a worry if we track the training loss with EarlyStopping, however we track the validation loss which would grow in the case of overfitting.

* Why the authors choose "an architecture of 8 hidden layers with 64 nodes each" for the MLP? As pointed out above, a single layer with sufficiently large number of neurons suffices for a general regression task, so the choice of 8 hidden layers seems a bit difficult to justify to me.

Although in theory a single layer is enough, in practice the number of nodes required in one layer may be too large for anything practical. On the other hand, having deep networks are known to be much more practical than a single layer. The architecture we choose here is similar to many regression/generative models used in e.g. https://arxiv.org/pdf/1707.00028.pdf, https://www.scipost.org/SciPostPhys.9.4.053/pdf .

* Concerning the use of Gaussian Processes, it may be appropriate to mention that feed-forward neural networks can be understood as a collection of GPs. So in this respect the techniques discussed in 3.1 and 3.3 are not really independent but rather they are closely related. It is thus a

bit weird that the performance of the GP method is so inferior as compared to the MLP, do the authors understand this point?

This correspondence is true for infinitely wide neural networks, so we do not think it would be relevant for our case. We do, however, state some reasons for the inferiority of SVGP in the discussion section.

* Are the figures of merit evaluated over all the data points or not? What happens if one uses say 80% to construct the interpolation and 20% for the validation? Do the results presented in the paper change a lot? An important benefit of MLP models is that they are reasonably stable upon extrapolation, which is not always the case with traditional interpolation techniques.

The figures of merit are evaluated over 1 million unseen testing points. This provides a fair comparison between the methods since they have not been trained/fitted to this dataset.

* Another important consideration when choosing an interpolation/regression strategy is related to uncertainty propagation. As mentioned above in most applications one is fitting to data with some fluctuations over the underlying "truth", and it is important to be able to estimate and propagate all uncertainties to the final model. The authors should discuss how the various interpolation/regression strategies considered should be combined with error propagation methods, a key ingredient of realistic applications.

The question of uncertainties and error propagation is important for realistic applications and is the subject of future work as mentioned in the conclusions section. We do not think that uncertainties in this analysis will change anything significantly. Having established the superiority of neural networks in high dimensions, future work will focus on different estimates of uncertainty such as ensembling or Monte Carlo dropout.