HYPPO: A Surrogate-Based Multi-Level Parallelism Tool for Hyperparameter Optimization

Vincent Dumont[*††], Casey Garner[†*], Anuradha Trivedi[‡*], Chelsea Jones[§*], Vidya Ganapati[¶*], Juliane Mueller[*], Talita Perciano[*], Mariam Kiran[*], and Marc Day[∥]

[*] *Computing Sciences Area, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*
[†] *Department of Mathematics, University of Minnesota Twin Cities, Twin Cities, MN 55414, USA*
[‡] *Department of Biomedical Engineering, Georgia Institute of Technology and Emory University, Atlanta, GA 30332, USA*
[§] *Department of Statistical Sciences & Operations Research, Virginia Commonwealth University, Richmond, VA 23284, USA*
[¶] *Department of Engineering, Swarthmore College, Swarthmore, PA 19081, USA*
[∥] *Computational Science Center, National Renewable Energy Laboratory, Golden, CO 80401, USA*

[††] Corresponding Author: vincentdumont11@gmail.com

Abstract—We present a new software, HYPPO, that enables the automatic tuning of hyperparameters of various deep learning (DL) models. Unlike other hyperparameter optimization (HPO) methods, HYPPO uses adaptive surrogate models and directly accounts for uncertainty in model predictions to find accurate and reliable models that make robust predictions. Using asynchronous nested parallelism, we are able to significantly alleviate the computational burden of training complex architectures and quantifying the uncertainty. HYPPO is implemented in Python and can be used with both TensorFlow and PyTorch libraries. We demonstrate various software features on time-series prediction and image classification problems as well as a scientific application in computed tomography image reconstruction. Finally, we show that (1) we can reduce by an order of magnitude the number of evaluations necessary to find the most optimal region in the hyperparameter space and (2) we can reduce by two orders of magnitude the throughput for such HPO process to complete.

I. INTRODUCTION

Deep learning models are increasingly used throughout the sciences to achieve various goals, including the acceleration of time-consuming computer simulations, for making predictions in cases where the underlying physical behavior are poorly understood, for deriving new insights into physical relationships, and for decision support. E.g., in [1], a low-fidelity simulation model is augmented with predictions made by a U-Net to obtain high-fidelity data at a fraction of the computational cost of running a high-fidelity simulation. In [2], DL models are used to make long-term predictions of groundwater levels in California in order to enable computationally fast predictions of future water availability. [3] used convolutional neural networks (CNNs) for anomaly detection in scientific workflows. DL models can also be used to co-optimize hardware acquisition parameters with image reconstruction in computational imaging, as in [4, 5]. [6] used neural networks to predict the CPU times of Ordinary Differential Equation (ODE) solvers, thus enabling the optimal selection of ODE solvers.

One obstacle to using DL models in new scientific applications is the decision about which hyperparameters (e.g., number of layers, nodes per layer, batch size) should be used. The hyperparameters impact the accuracy of the DL model's predictions and should therefore be tuned. However, the predictive performance is subject to variability that arises due to the use of stochastic optimizers such as stochastic gradient descent (SGD) for training the models. This variability must be taken into account during hyperparameter optimization (HPO) in order to achieve models that produce accurate predictions reliably.

There are many libraries defined for hyperparameter tuning. Each of them has different strengths, compatibility, and weakness. Some of the frequently used libraries are Ray-Tune which supports state-of-the-art algorithms such as population-based training [7], Bayes Optimization Search (BayesOpt-Search) [8, 9]; Optuna, which provides easy parallelization [10]; Hyperopt, which works both serial and parallel ways and supports discrete, continuous, conditional dimensions [11]; Polyaxon, which is for large scale applications [12].

However, little work discusses how the prediction uncertainty of deep learning models can be reduced with these approaches. This paper describes a new software tool, HYPPO, for automated HPO that is based on ideas from surrogate modeling, uncertainty quantification, and bilevel black-box optimization. In contrast to other HPO methods such as DeepHyper [13] and MENNDL [14], our HPO implementation directly takes the prediction variability into account to deliver *highly accurate and reliable* DL models. We demonstrate the approach on the CIFAR10 image classification problem, time series prediction and a computed tomography image reconstruction application. We take full advantage of high-performance computing (HPC) environments by using asynchronous nested parallelization techniques to efficiently sample the high-dimensional hyperparameter space.

Relevance, impact, and contribution to the literature:

1) **Automated and adaptive HPO for DL models that directly accounts for uncertainty in model predictions;**
2) **Significant decrease in the number of evaluations necessary to identify the most optimal region in the hyperameter space;**
3) **Nested parallelism significantly accelerates time-to solution (up to two orders of magnitude);**
4) **Compatible with both TensorFlow and PyTorch;**
5) **HYPPO can run on large compute clusters as well as laptops and is able to exploit GPU and CPU;**
6) **Provides reliable and robust models.**

II. MOTIVATION

Much work on hyperparameter search extensively studies applications in CNNs for image recognition [15] or general deep learning models [13]. Further development of various hyperparameter search libraries such as RLlib [16], Hyperband [17], DeepHyper [13], and even hand-tuning help iterate through multiple settings repeatedly until finding fairly well-performing hyperparameters. However, deep learning models can lead to multiple predictions with varying range of values, even after tuning the hyperparameters. Current hyperparameter tuning libraries often ignore this challenge of prediction variability. In this work we particularly target this problem of uncertainty quantification, by building in uncertainty when tuning hyperparameters. The ability to quantify the prediction uncertainty of DL models has the advantage that we can obtain confidence bounds for the DL model's predictions, as illustrated in two examples in Figure 1. The additional information about prediction uncertainty can be very advantageous in various applications. For instance, instead of fully trusting a model architecture because of good predictive performance (which can be interpreted as a single realization of a stochastic process), UQ provides additional information on how reliably that architecture performs. If, for example, one was to predict future groundwater levels to enable sustainable groundwater management (see [2]), having an idea about the prediction variability enables managers to develop strategies that are based on best, worst, and average case performance information and thus lead to robust management decisions. Similarly, in classification tasks, scientists may not only be interested in the class with highest expected probability, but also in information such as the second or third most probable class membership as well as uncertainties of those memberships which will allow further insights into the significance of differences between class memberships. In particular, in scientific applications where determining the correct class may have huge negative impacts if done incorrectly, this additional information is highly relevant. The main contributions of HYPPO include (1) an automated and adaptive search for hyperparameters using uncertainty quantification to evolve the best deep neural network solutions for both PyTorch and Tensorflow codes, and (2) demonstrate the scalability of our solution using high-performance computing (HPC) to leverage multiple threads and parallel processing to improve the overall search time for hyperparameters.

Our experiments cover time series prediction, image classification, and image reconstruction, and present how surrogate modeling can help to find optimal deep learning architectures that minimize the uncertainty in the solutions.

III. BI-LEVEL OPTIMIZATION PROBLEM

Mathematically, we formulate the HPO problem as a bilevel bi-objective optimization problem:

$$\min_{\boldsymbol{\theta}, \mathbf{w}^*} \quad (\ell_1(\boldsymbol{\theta}, \mathbf{w}^*; \mathcal{D}_{\text{val}}), \ell_2(\boldsymbol{\theta}, \mathbf{w}^*; \mathcal{D}_{\text{val}})) \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{\theta} \in \Omega \tag{2}$$

$$\mathbf{w}^* \in \arg\min_{\mathbf{w} \in \mathcal{W}} L(\mathbf{w}; \boldsymbol{\theta}, \mathcal{D}_{\text{train}}). \tag{3}$$

Here, $\boldsymbol{\theta}$ represents the set of hyperparameters to be tuned, Ω describes the space over which the parameters are tuned (in our case an integer lattice), and ℓ_1 and ℓ_2 represent the loss and loss variability of the trained model, respectively. Ideally, we want to find optimal $\boldsymbol{\theta}$ such that both ℓ_1 and ℓ_2 are minimized simultaneously. Note that most commonly only a single objective that represents a mean squared loss is minimized and the variability is not accounted for.

The values of $\ell_i, i = 1, 2$, for each hyperparameter set for the validation dataset $\mathcal{D}_{\text{val}}$ can be computed only after solving the lower level problem (Eq. 3), in which the weights and biases $\mathbf{w}^*$ are obtained by training (e.g., with SGD). Solving this lower level problem is, depending on the size of the training dataset $\mathcal{D}_{\text{train}}$ and the size of the architecture, computationally expensive. The loss variability arises when the same architecture is trained multiple times because different solutions $\mathbf{w}^*$ are obtained (see Section IV, Feature 1, for details), and thus different values for the loss ℓ_1 at the upper-level result. Our goal is to find optimal hyperparameters $\boldsymbol{\theta}^*$ that yield high predictive accuracy (low ℓ_1 values) and low prediction variability (low ℓ_2 values).

In order to tackle this problem, we interpret the HPO problem as a computationally expensive black-box problem in which the black-box evaluation corresponds to a stochastic evaluation of ℓ_1 with variability ℓ_2. Furthermore, to alleviate the complexities of solving a bi-objective optimization problem, we use a single-objective reformulation in which a weighted sum of ℓ_1 and ℓ_2 is minimized and the weights corresponding to each reflect their respective importance.

To solve our HPO problem, we use ideas from the surrogate model-based optimization algorithm presented in [2] and extend it to consider the model uncertainty. We make full use of HPC by exploiting asynchronous nested parallelism strategies which significantly reduces the time to finding optimal architectures.

A. Surrogate Model-based HPO

Surrogate models such as radial basis functions (RBFs) [18] and Gaussian process models (GPs) [19] have been widely used in the derivative-free optimization literature. These models are used to map the parameters $\boldsymbol{\theta}$ to the model performance

82

metrics, $\ell_i(\boldsymbol{\theta})$ (here the dependence of ℓ_i on $\mathbf{w}^*$ is implied by means of interpreting the lower level problem as black-box evaluation). The surrogate models are computationally cheap to build and evaluate and thus enable an efficient and effective exploration of the search space [20]. Adaptive optimization algorithms using global surrogate models usually consist of the following three steps:

1) Create an initial experimental design and evaluate the costly objective function to obtain input-output data pairs.
2) Build a surrogate model based on all data pairs.
3) Solve a computationally cheap auxiliary optimization problem on the surrogate model to select new point(s) at which the expensive function is evaluated and go to step 2.

In order to apply these methods to HPO, special sampling strategies that respect the integer constraints of the hyperparameters are needed (see [2] for details). Moreover, the iterative sampling strategy must take into account the performance variability, which can be done by defining suitable model performance metrics.

IV. HYPPO SOFTWARE: DESIGN & FEATURES

In this section, we provide details of specific features of our HYPPO software and implementation. We describe our approach to accounting for the prediction variability during HPO, the proposed surrogate modeling and sampling approaches, and the asynchronous nested parallelism that enables us to speed up the calculations significantly.

Feature 1: Uncertainty Quantification

HYPPO employs and applies uncertainty quantification (UQ) in conjunction with surrogate model-based HPO. The prediction variability present in Eq. (3) is epistemic where the variability observed manifests itself because Eq. (3) is a non-convex problem being solved in-exactly by stochastic algorithms. Non-convexity limits knowledge of global optimality, while in-exactly solving the lower level problem gives no guarantee that the $\mathbf{w}^*$ used to evaluate the loss ℓ_1 in Eq. (1) is a stationary point to Eq. (3).

The body of potential UQ approaches to bound the performance variability for a fixed hyperparameter set $\boldsymbol{\theta}$ is thinned significantly by the nature of the problem. Since evaluating the loss is an expensive black-box function, any UQ method that depends on heavy sampling quickly becomes computationally intractable. Furthermore, distributional information for the performance value based on $\mathbf{w}^*$ is unknowable beyond trivial cases because such knowledge is tied to the local optima of Eq. (3), which in general are neither computed during training nor enumerated. An additional blockage to distributional knowledge comes from the possibility that different global optima of Eq. (3) likely yield different loss function values in Eq. (1). Finally, flexible UQ approaches capable of handling various DL models are essential for the broad applicability of our software. Considering the inherent difficulties and desired

properties delineated, the central framework we utilize for UQ is Monte Carlo (MC) dropout [21].

MC dropout was developed in the form presented by Gal and Ghahramani [21, 22] as an alternative derivation of *model averaging* from [23]. The key idea behind MC dropout is to use standard dropout during testing. Dropout is a technique often applied to prevent a network from overfitting and thus improve its generalizability [23, 24]. This is achieved by taking each node out of the network with some probability p during training and then multiplying the network's weights by p during test time. Equivalently, as done in PyTorch and TensorFlow dropout layers, nodes can be dropped with probability p and the value of the remaining nodes scaled by $1/(1-p)$ during training resulting in no scaling of the weights during testing. So, MC dropout applies dropout on a network at test time, i.e., when the input is evaluated by forward-propagation, each node is dropped out with probability p and the output of the remaining nodes is scaled by $1/(1-p)$. Therefore, forward-propagating the same input through a network using dropout will produce different outputs on each pass from which we can obtain a measure of variability for the output. Formulating this mathematically, let $\mathbf{y}_t(\mathbf{x})$ be the output of a DL model for input $\mathbf{x}$ on the t-th pass through the network using MC dropout. From T total passes through the network, we can compute a sample mean for the output of input $\mathbf{x}$ as,

$$\bar{\mathbf{y}}(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{y}_t(\mathbf{x}), \tag{4}$$

and a sample variance,

$$\sigma_{\text{model}}^2 = \frac{1}{T} \sum_{t=1}^{T} \left(\mathbf{y}_t(\mathbf{x}) - \bar{\mathbf{y}}(\mathbf{x}) \right)^2, \tag{5}$$

where the squared terms in Eq. (5) are squared element-wise, i.e. $\left[(\mathbf{a} - \mathbf{b})^2 \right]_j = (a_j - b_j)^2$ for all j where a_j is the j-th element of $\mathbf{a}$. A crucial observation is that MC dropout derives variability measures from repeated evaluations of a DL model rather than large numbers of repeated training of the same model or through the construction of another network. Thus, it satisfies the general restrictions imposed by the characteristics of our loss functions in Eq. (1).

Our UQ approach applies MC dropout to a relatively small set of trained models with the same architecture to form a weighted average. Combining MC dropout with repeated training of the same architecture balances the explanatory power of repetitive training with the computational frugality of MC dropout. Assume we have N trained models with the same architecture $\boldsymbol{\theta}$ for which we generate T dropout masks each. We compute the expected output of a trained model with the same architecture as,

$$\boldsymbol{\mu}_{\text{pred}}(\mathbf{x}) = \frac{w_T}{N} \sum_{i=1}^{N} \boldsymbol{y}^i(\mathbf{x}) + \frac{w_D}{NT} \sum_{j=1}^{N} \sum_{t=1}^{T} \boldsymbol{y}_t^j(\mathbf{x}), \tag{6}$$

where $\boldsymbol{y}^i(\mathbf{x})$ is the output of the trained model $i \in \{1, \dots, N\}$ for input $\mathbf{x}$ without dropout at test time, $\boldsymbol{y}_t^j(\mathbf{x})$ is the output

83
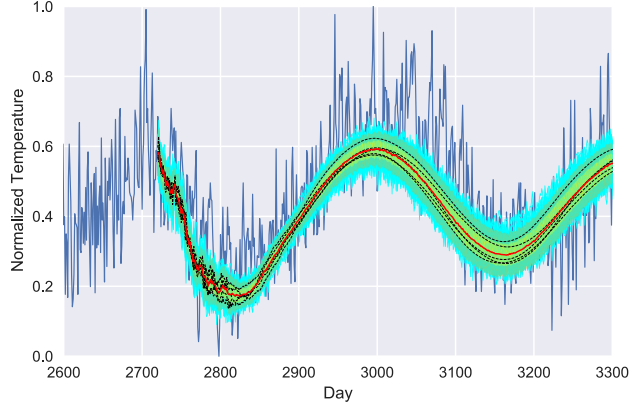
(a) Time series prediction of temperature in Melbourne, Australia

(b) CIFAR10 image classification

Fig. 1: Application of Monte Carlo dropout to a DL models. In Fig. 1a, the yellow and green regions represent the ± 1 and 2 standard deviations from the mean (red line). The cyan lines are the MC dropout predictions and the black dashed lines are the mean predictions from each of the 5 independent trial models. The DL model used "prediction-on-prediction", i.e., the model's predicted values were utilized in making future predictions. In Fig. 1b, the confidence interval around the mean probability for each class is shown for a single input image. While the right class (number 8) is correctly identified, the uncertainty of the output probability provides important information with regard to the stability and accuracy of the trained model.

of the dropout model $t \in \{1, \ldots, T\}$ for trained model $j \in \{1, \ldots, N\}$, and $w_T \geq 0$, $w_D > 0$ with $w_T + w_D = 1$. The variance of the expected output is computed as,

$$\boldsymbol{V}_{\text{model}}(\mathbf{x}) = \frac{w_T}{N} \sum_{i=1}^{N} \left(\boldsymbol{\mu}_{\text{pred}}(\mathbf{x}) - \boldsymbol{y}^i(\mathbf{x}) \right)^2$$
$$+ \frac{w_D}{NT} \sum_{j=1}^{N} \sum_{t=1}^{T} \left(\boldsymbol{\mu}_{\text{pred}}(\mathbf{x}) - \boldsymbol{y}_t^j(\mathbf{x}) \right)^2 . \quad (7)$$

Note that Eqs. (6) and (7) are weighted averages of the computed mean and variances of the trained models and the MC dropout models. These equations enable micro-predictions of the performance uncertainty of a model for a given input. The weighted average parameters w_T and w_D, as well as the number of iterations T of MC dropout per trained model, are user-defined settings. An example of our modified MC dropout approach can be seen in Figure 1 for time series prediction of daily temperature in Melbourne, Australia, and CIFAR10 image classification. Here $w_T = w_D = 0.5$ and $T = 30$, which correspond to our default values. The robustness of the model predictions can be measured, for example, in the average width of the uncertainty bands associated with each day's temperature prediction and with each class membership probability, respectively.

Eqs. (6) and (7) can then be used to obtain confidence intervals for the loss function value corresponding to a given hyperparameter set $\boldsymbol{\theta}$. For example, if $\ell_1 \left(\boldsymbol{\theta}; \mathbf{w}^*, D_{\text{val}} \right)$ is the mean-squared loss function with validation data set $\mathcal{D}_{\text{val}} = \left\{ (\mathbf{x}^i, \mathbf{z}^i) \right\}_{d=1}^{D}$, then we approximate the expectation over $\mathbf{w}^*$

by

$$\mathbb{E}_{\mathbf{w}^*} \left(\ell_1 \right) \approx \frac{1}{2D} \sum_{d=1}^{D} \left\| \mathbf{z}^i - \boldsymbol{\mu}_{\text{pred}}(\mathbf{x}^i) \right\|^2 .$$

The intuition is that $\boldsymbol{\mu}_{\text{pred}}$ serves as a good estimate of the actual expectation of the output over all possible trained models with architecture $\boldsymbol{\theta}$. Now, from computing the outputs of each of the trained models, $\mathbf{y}^i(\mathbf{x}^d)$, and their many dropout masks, $\mathbf{y}_t^i(\mathbf{x}^d)$, for all of the inputs $\mathbf{x}^d \in \mathcal{D}_{\text{val}}$, we are able to compute the value of ℓ_1 for each of the models utilized to compute $\boldsymbol{\mu}_{\text{pred}}$. Taking the standard deviation of these $N+TN$ outer loss values gives an estimate for the variability ℓ_2 in the outer loss function for a model with architecture $\boldsymbol{\theta}$. Letting the value of ℓ_1 computed from $\boldsymbol{\mu}_{\text{pred}}$ be the center and using the aforementioned standard deviation as a radius, we obtain a confidence interval for the outer objective.

Upon obtaining confidence intervals and output variances using Eq. (7) for ℓ_1, we desire to utilize this additional uncertainty information in the HPO. Our software uses uncertainty in different ways. The first option for integrating uncertainty is to use the surrogate models with the value of ℓ_1 taken as its output when computed with $\boldsymbol{\mu}_{\text{pred}}$. The RBF and GP based optimization approaches available in the software (see next section for details) would then run as expected with confidence intervals being computed and tabulated for all new hyperparameter settings.

The second option is to use the confidence intervals to construct an ensemble of RBFs to perform the surrogate modeling; the *RBF ensemble approach* uses multiple RBFs that are generated from the confidence intervals by selecting uniformly at random from the extremes of these intervals, i.e., the lower bound, center, and upper bound. Each response

surface provides an estimate for the value of ℓ_1 at potential hyperparameter candidates, $\boldsymbol{\theta}_i^C$ (see Feature 2 for description of their generation), and we compute a mean $\mu(\boldsymbol{\theta}_i^C)$ and standard deviation $\sigma(\boldsymbol{\theta}_i^C)$ over all ensemble predictions for ℓ_1 at each candidate point. The selection of the best candidate point (and thus new hyperparameters to be evaluated) is determined by which candidate point minimizes a weighted sum of its distance to previously evaluated points and,

$$\mu(\boldsymbol{\theta}_i^C) + \alpha\sigma(\boldsymbol{\theta}_i^C), \qquad (8)$$

where $\alpha \in [-2, 2]$. Note that if $\alpha = 0$, then only the mean value of ℓ_1 predicted by the ensemble is considered when determining the next hyperparameter setting. If $\alpha = 2$, a "pessimistic" approach is taken by penalizing candidate points with large prediction variability. On the other hand, by setting $\alpha = -2$, an "optimistic" stance is taken by preferring candidate points with relatively good mean performance and large standard deviation that may lead to significant improvements.

The first approach can also be adapted to incorporate the computed variances $\boldsymbol{V}_{\text{model}}$ by appending them into the outer loss function ℓ_1. In the HYPPO software, a setting is available to add a regularization term to the outer objective to create a regulated loss function ℓ_{reg},

$$\ell_{\text{reg}}\left(\boldsymbol{\theta}; \mathbf{w}^*, \mathcal{D}_{\text{val}}\right) = \ell_1\left(\boldsymbol{\theta}; \mathbf{w}^*, \mathcal{D}_{\text{val}}\right)$$
$$+ \gamma \sum_{d=1}^{D} g\left(\boldsymbol{V}_{\text{model}}\left(\mathbf{x}^d\right)\right), \quad (9)$$

where $g : \mathbb{R}^d \to \mathbb{R}^+$ and $\gamma > 0$. Thus, choosing γ to be large relative to the first term places a substantial emphasis on minimizing the performance variability during the surrogate modeling. A user could also tailor g to magnify or dampen the variance to varying degrees or even for particular ranges of inputs by defining g in a piece-wise fashion, e.g. $g(\mathbf{x}) = ||\max(0, \mathbf{x})||$. If this setting is applied, then the value of ℓ_{reg} is used by the surrogate models while the confidence interval of ℓ_1 is returned during the HPO as previously noted.

In Fig. 2, we show one typical result provided by HYPPO to the user, showing the distribution of trained model according to their computed loss and confidence limit. Along with the total number of trainable parameters for each model, which characterizes the complexity of the models' architecture, this graph provides insightful information that can help users decide which model is most efficient and suitable for their problem. A combination of low uncertainty, low loss and small number of trainable parameters can be considered as an optimal choice of model.

Feature 2: Surrogate Modeling

Our HPO software contains two different types of surrogate models, namely RBFs and GPs, and corresponding iterative sample strategies. The HPO method is, however, general enough and can easily be extended to other surrogate models. The RBF surrogate model is of the form,

$$m_{\text{RBF}}(\boldsymbol{\theta}) = \sum_{j=1}^{n} \lambda_j \phi\left(||\boldsymbol{\theta} - \boldsymbol{\theta}_j||_2\right) + p(\boldsymbol{\theta}), \qquad (10)$$
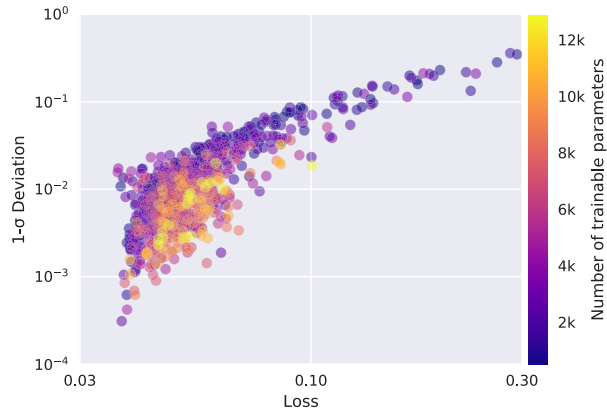

Fig. 2: Distributions of the loss, standard deviation and total number of trainable parameters across 825 trained multi-layer perceptron (MLP) models with different hyperparameter sets. The models were used to do time series prediction of the daily temperature in Melbourne, Australia. While complex architectures, i.e. high number of trainable parameters, are found to be clustered, low-complexity models can be identified in the low-loss, low-uncertainty region, making them suitable model solutions.

where $\boldsymbol{\theta}_j$ are the sets of hyperparameters for which the objective function has previously been evaluated, $\|\cdot\|_2$ denotes the Euclidean norm, $\phi(r) = r^3$, and $p(\boldsymbol{\theta}) = \beta_0 + \boldsymbol{\beta}^T\boldsymbol{\theta}$ is a polynomial tail whose form depends on the choice for $\phi(\cdot)$. A system of equations (see Eq. 6 in [2]), dependent on the value of ℓ_1 (or ℓ_{reg}) at the $\boldsymbol{\theta}_j$'s, is solved to compute λ, β_0 and $\boldsymbol{\beta}$. When using the RBF ensemble approach as described in the previous section, the various values sampled from the confidence intervals are used as the right-hand side of Eq. 6 in [2] thus generating multiple response surfaces.

When making iterative sampling decisions with the RBF model, we follow the ideas in [25]. We generate a large set of candidate points by perturbing the best point found so far and by randomly selecting points from the search space, making sure we satisfy the integer constraints. For each candidate point, we use the RBF to predict its function value (e.g., loss) and we also compute the distance of each candidate point to all previously evaluated hyperparameters. A weighted sum of these two criteria is then used to select the best candidate point at which the next expensive evaluation (model training, MC dropout) is performed. The weights cycle through a predefined pattern to enable a repeated transition between local and global search.

The second type of surrogate model implemented in our software is GPs. GPs have the advantage that in addition to providing a prediction of the function value, they also return an uncertainty estimate of the prediction. When using a GP, we treat the function as a realization of a stochastic process:

$$m_{\text{GP}}(\boldsymbol{\theta}) = \nu + Z(\boldsymbol{\theta}), \qquad (11)$$
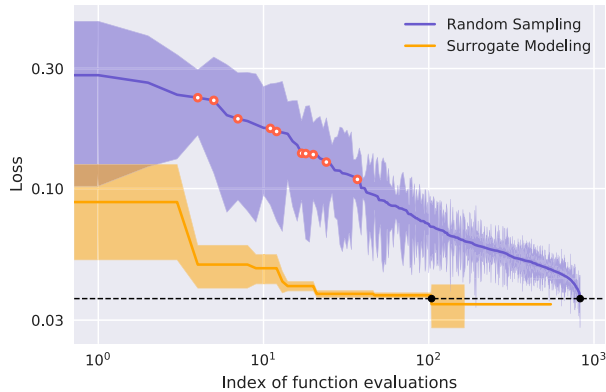
Fig. 3: Convergence plot showing the loss value and its 1-σ deviation for all function evaluations. In purple, we show the sorted loss values for a random set of 825 hyperparameter samples generated using low-discrepancy sampling. From that set, 10 evaluations with high losses were selected (red points) and used as initial evaluations for surrogate modeling. In orange, we show the best error loss as the surrogate modeling is iterating and were able to decrease by an order of magnitude the amount of evaluations required to find the most optimal region in the hyperparameter space.

where ν represents the mean of the stochastic process and $Z(\boldsymbol{\theta}) \sim \mathcal{N}(0, s^2)$. Therefore, the term $Z(\boldsymbol{\theta})$ represents a deviation from the mean ν. It is assumed that there exists a correlation between the errors and that it depends on the distance between the hyperparameters. For an unsampled set of hyperparameters, the GP prediction represents the realization of a random variable with mean ν and variance s^2. Further details on how to compute the mean and variance can be found in [2, Eqs. 7-13]. When using the GP surrogate model, we maximize the expected improvement [26] auxiliary function using a genetic algorithm that can handle the integer constraints. This sampling strategy uses the GP-predicted value of the loss ℓ_1 and the corresponding GP-predicted uncertainty to balance a local and a global search for improved solutions in the hyperparameter space. The process of utilizing MC dropout with GP remains unchanged compared to the implementation with RBF. The only difference is that no ensemble approach, as detailed for the RBFs, is needed as the GP provides the prediction mean and standard deviation.

In Fig. 3, we demonstrate the effectiveness of using surrogate modeling and compare the increase of convergence speed to reach the optimal point in the hyperparameter space, where the lowest loss model can be achieved with an order of magnitude fewer iterations.

The improvement in HPO performance with HYPPO from a different library, e.g. DeepHyper, is shown in Fig. 4. To do the comparison, we used the polynomial fit problem provided in DeepHyper's online documentation[1] and we extended
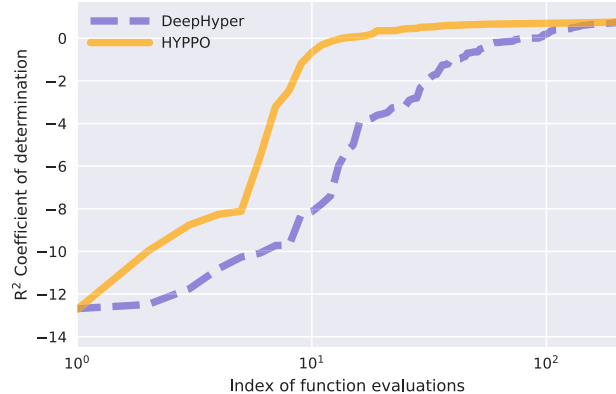
Fig. 4: Comparison of convergence plot between our HYPPO software and the DeepHyper HPO library. We used the polynomial fit problem provided in the DeepHyper's online documentation, allowing 200 iterations. In this example, the R^2 value (also known as "Coefficient of determination") is the metric being maximized.

the complexity of the problem by increasing the number of hyperparameters to be optimized to six: (1) number of nodes per layer, (2) number of layers, (3) dropout rate, (4) learning rate, (5) epochs, and (6) batch size. We applied HPO with both HYPPO and DeepHyper libraries independently and evaluated the highdimensional hyperparameter space at a total of 200 iterations with each method. Ten initial evaluations were used in HYPPO to build the initial surrogate model. Our analysis shows that HYPPO and DeepHyper both find eventually models with the same performance, but HYPPO finds better hyperparameters faster (fewer iterations) than DeepHyper, adding further motivation to using HYPPO.

Feature 3: Asynchronous Nested Parallelism

One of the challenges in carrying out HPO for machine learning applications is the computational requirement needed to train multiple sets of hyperparameters. In order to address this problem, the HYPPO software handles parallelization across multiple hyperparameter evaluations and enables distributed training for each evaluation. Here, we describe how the nested parallelism feature is being implemented and used to maximize efficiency across all allocated resources.

1) Nested parallelism: In Fig. 5, we show how we set up the parallelization scheme in the software. The HYPPO software uses GNU parallel [27] to do distributed training of multiple models in parallel. The program can automatically generate a SLURM script using the number of SLURM steps to be executed in parallel (that is, the number of individual `srun` instances) and the number of SLURM tasks to be executed in each step, as defined by the user in the input configuration file. Since a single processor (either GPU or CPU, the user decides) will be assigned for each task, the total number of processors used in a SLURM job is therefore equal to the number of steps times the number of tasks. For
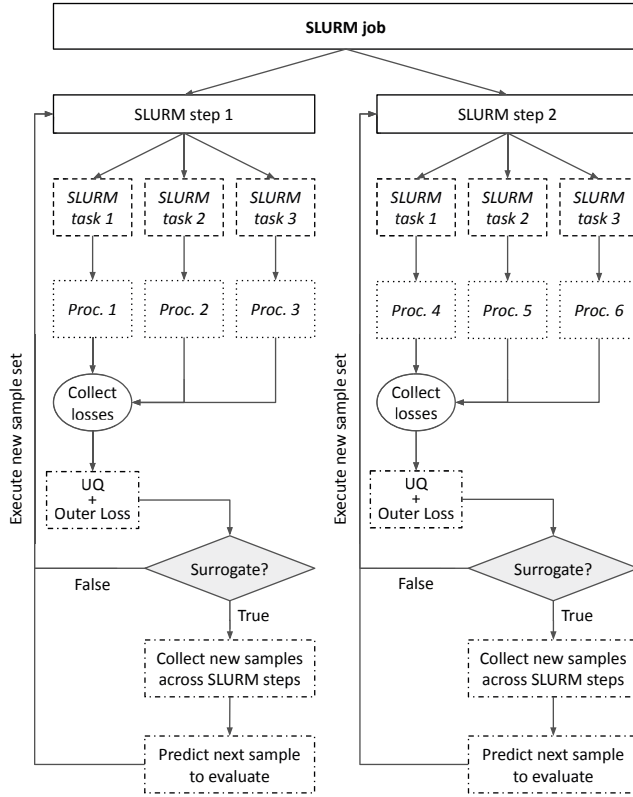
Fig. 5: Diagram illustrating the nested parallelization process across 6 processors using 2 parallel steps and 3 processors per step. A single processor is used per SLURM task. If the user requests uncertainty quantification to be performed on the trained models, the outer loss will be provided along with its confidence interval. When surrogate modeling is performed, the algorithm will gather newly evaluated sample sets across all parallel steps and make a prediction for the next set of hyperparameters to evaluate.

instance, proper SLURM directives for a setup with 2 `srun` instances running in parallel across 3 GPUs for each step will read as follows:

```
#SBATCH --ntasks 6
#SBATCH --gpus-per-task 1
```

In this case, `--ntasks` is equal to the total number of processors to be allocated for the job. Each of the 2 SLURM steps can then be executed in parallel using GNU parallel with a `--jobs` command set to 2. Finally, in order to avoid individual `srun` steps to be executed on the same GPUs, we use the `--exclusive` command for the `srun` call which will ensure that separate processors will be dedicated to each job step.

Once the program starts, workers will be initialized according to the SLURM settings. If the used machine learning library is PyTorch, the program will do the initialization using the `torch.distributed` package. On the other hand, if

the Tensorflow library is used, the Horovod package will be used to initialize the workers. While the software is flexible enough to work with external training modules, the library used in the external package needs to be specified in the input configuration file so that the program can initialize the workers accordingly.

Using the SLURM environment variables, we can then loop over the randomly generated hyperparameter sets, ensuring that each step executes a different set of hyperparameters. This can be achieved using Python's slicing feature where the subsequence is defined using the step ID and the total number of steps in the job. When multiple processors are requested for each evaluation, the loss for the outer objective function will always be calculated in the first processor (e.g., GPU0 if GPU processors are used) and the value will be recorded in its corresponding log file. In the meantime, the remaining processors will wait for the task to be completed by searching for the recorded value in the first processor's log file.

2) Data / Trial Parallelization: As quantifying the uncertainty becomes more accurate with an increasing number of trials (i.e., repeatedly training the same architecture), the HYPPO software offers the possibility, in a single SLURM step, to parallelize the trials instead of the data to be trained on. Thus, in the example in Figure 5, the SLURM job corresponds to performing HPO, SLURM steps 1 and 2 correspond to two different sets of hyperparameters to be evaluated, and SLURM tasks 1-3 correspond to performing three separate trainings of each hyperparameter set. For example, suppose for a specific hyperparameter set, the model is to be trained nine times. In that case, each GPU will execute three consecutive independent trainings, and slicing will be applied to the trial indices using the extracted MPI rank and size values to ensure that the total number of trials to be computed across all GPUs in each step is equal to the number of trials requested by the user. On the other hand, if the user prioritizes the parallelization over the data (train in parallel), then in the example in Figure 5, the SLURM job still corresponds to performing HPO, SLURM steps 1 and 2 correspond to two different sets of hyperparameters to be evaluated, and SLURM tasks 1-3 perform the parallel training using a third of the data per GPU. Again, if the goal was to train each hyperparameter set nine times, each GPU will execute all nine independent training trials sequentially.

3) Asynchronous update: The surrogate model-based optimization is updated every time a new input-output data pair is obtained during the surrogate model-based optimization. This process is usually done sequentially (one evaluation per iteration and update of the model) or synchronously parallel (multiple evaluations simultaneously and update the model only after all evaluations are completed). This may require many iterations before the surrogate model adequately represent the loss function over the entire hyperparameter space. Moreover, in HPO, each hyperparameter evaluation may require a different amount of time because architectures of different sizes have different numbers of weights and biases that the training process optimizes. Using the aforementioned
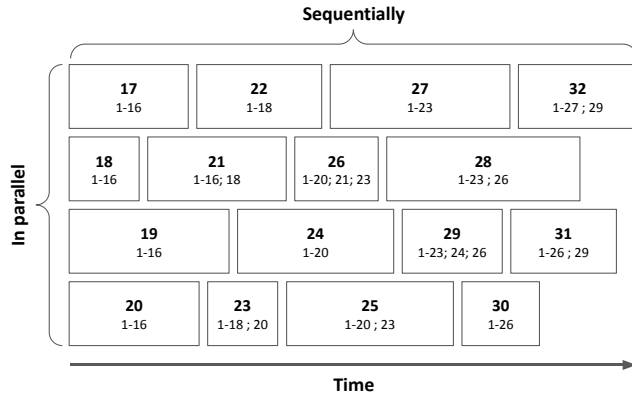
87

Fig. 6: Illustration of asynchronous surrogate modeling. Each rectangle represents an evaluation, the top number corresponds to the evaluation index and the bottom values are the indices of the evaluations that were used by the surrogate model to determine the current hyperparameter set. In this example, 16 initial evaluations were done prior to starting the surrogate modeling and asynchronous parallelism. The model was then fitted and predicted 4 new sample sets to evaluate (evaluations 17 through 20 shown in bold). Once the first evaluation completed, i.e. evaluation 18 in this example, a surrogate model was refitted using the newly computed evaluation (in addition to the first 16 evaluations) and a new set of hyperparameters to evaluate was predicted (here, evaluation 21).

nested parallelization feature, we can use this approach to predict not one but multiple sets of samples to be evaluated asynchronously in parallel. The surrogate model is then updated each time a hyperparameter set has been evaluated.

Nevertheless, unlike the initial evaluations that are performed independent of each other, the surrogate modeling process requires communication between the SLURM steps in order to retrieve newly evaluated hyperparameter sets from other `srun` instances and update the model. In Fig. 6 we illustrate how evaluations can complete at different times during the surrogate modeling process. After each completed evaluation, the HYPPO software reads through all the log files generated and constantly updated by each processor to search for newly computed sample sets. Once the search is complete, the surrogate model is updated and a new sample set is computed.

V. CASE STUDY: CT IMAGE RECONSTRUCTION

This section describes a relevant scientific application from computed tomography (CT) and demonstrates the impact of our HPO process on the results. CT is a three-dimensional imaging technique that measures a series of two-dimensional projections of an object rotated on a fixed axis relative to the direction of an X-ray beam. The collection of projections from different angles at the same cross-sectional slice of the object is called a 2D sinogram, which is the input to a reconstruction algorithm. The final 3D reconstructed volume

is called a tomogram, which is assembled from the independent reconstructions of each measured sinogram. Tomographic imaging is used in various fields, including biology [28], medicine [29], materials science [30], and geoscience [31], allowing for novel observations that enable enhanced structural analysis of subjects of interest.

High-quality tomographic reconstruction generally requires measuring projections at many angles. However, collecting sparsely sampled CT data (i.e., sparse distribution of angle measurements) has the benefit of exposing the subject to less radiation and improving temporal resolution. In sparse angle CT, where few angles are available in the measured sinograms, computing the reconstruction is a severely ill-posed inverse problem. When using standard algorithms, the reconstructions will be noisy and obstructed with streak-like artifacts. In recent years, a class of algorithms using deep learning has come into prominence [32, 33, 34, 35, 36, 37, 38]. Such approaches show superior performance compared to standard algorithms by using a trained neural network to solve the inverse problem of sparse angle CT reconstruction.

Here, we use HPO to optimize a deep neural network architecture for performing sinogram inpainting, an approach that has found success in other works [39, 40]. The missing angles of a sparsely sampled sinogram are filled in by a trained neural network, after which the completed sinogram can be reconstructed using any standard algorithm.

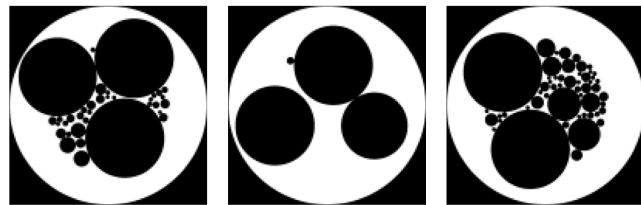### A. Data and Architecture

Fig. 7: Test data examples for the CT reconstruction problem.

We use a simulated dataset created with *XDesign*, a Python package for generating X-ray imaging phantoms [41]. The dataset comprises 17500 images of 128×128 pixels (13500 training, 1500 validation, 2500 test examples) of circles of various sizes, emulating the different feature scales present in experimental data as shown in Fig. 7. We use TomoPy [42], a Python package for tomographic image reconstruction and analysis, to generate sinograms of these images with 20 angles. To emulate sparse angle CT, every other angle is removed from the sinogram and Poisson noise is added.

The neural network architecture used in this problem takes the form of a U-Net, a type of CNN that has shown success in biomedical image processing problems [43]. The input of the U-Net is the sparse angle sinogram, and the desired output is the completed sinogram. Our variation of the U-Net architecture consists of an equivalent number of downsampling and upsampling blocks. Within each block, several intermediate convolution layers preserve the size of the input,

88

and a final convolution layer increases the number of feature maps (channels). We selected eight hyperparameters for HPO, enumerated in Table I. HPO aims to minimize the mean squared error between the actual and predicted sinograms of the validation dataset.

TABLE I: A total of eight hyperparameters were selected for hyperparameter optimization: (1) number of output feature maps of the initial block, (2) multiplier for the number of feature maps in each subsequent block, (3) number of blocks, (4) intermediate layers, (5) convolutional kernel size and (6) stride of the final layer in each block, (7) dropout probability and (8) intermediate layer kernel size. The first eight rows show the value of each hyperparameter and the last four rows contain the training results and image metric quantities averaged over the test dataset. Columns (a) and (d) are the results of the network trained with the minimum and maximum values for each hyperparameter, respectively. Columns (b) and (c) are the results of training the neural network with the best and worst hyperparameter values sampled by HYPPO, respectively.

Parameters	(a)	(b)	(c)	(d)
(1)	8	9	10	12
(2)	1.0	1.0	1.2	1.4
(3)	2	2	3	4
(4)	1	1	4	4
(5)	2	3	4	5
(6)	1	1	2	2
(7)	0.00	0.01	0.08	0.1
(8)	2	3	5	5
MSE	2.98E−5	3.39E−5	1.56E−3	2.46E−3
PSNR	35.2	34.6	18.7	16.4
SSIM	0.965	0.988	0.970	0.955

The resulting sinogram from the trained U-Net is reconstructed by TomoPy using the Simultaneous Iterative Reconstruction Technique (SIRT) [44], which has shown the ability to produce accurate reconstructions given incomplete datasets. SIRT iteratively minimizes the error between the measured projections and the projections calculated from the reconstruction at the current iteration k. The average error is then backprojected to refine the reconstruction at each iteration. For an inverse problem $\mathbf{Ax} = \mathbf{b}$, the SIRT update equation is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{CA}^\top \mathbf{R} \left(\mathbf{b} - \mathbf{Ax}_k \right),$$

where $\mathbf{C}$ and $\mathbf{R}$ are diagonal matrices that contain the inverse of the sum of the columns and rows, respectively, of $\mathbf{A}$.

B. Results

We analyze two aspects of applying HPO to the case study of CT image reconstruction: 1) job speedup as a function of the number of SLURM steps and SLURM tasks for the specific configuration of running the evaluation operation (see Sec. IV-2) and 2) improvement due to optimizing the hyperparameters. HYPPO is executed on the Cori supercomputer, housed at the National Energy Research Scientific Computing Center at Lawrence Berkeley National Laboratory (NERSC). Cori is a
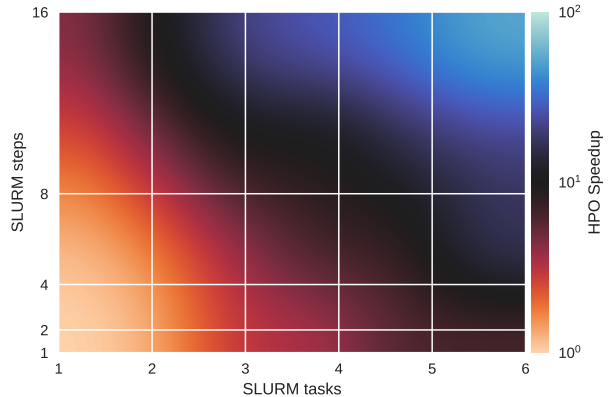

Fig. 8: Job speedup as a function of the number of SLURM steps and number of SLURM tasks is shown for evaluating 50 different hyperparameter sets and five trials for each. Up to 96 GPU processors from the NERSC/Cori cluster were used to evaluate performance and scalability of our software. Lower speedup values correspond to better performance.

Cray XC40 system comprised of 2,388 nodes each containing two 2.3 GHz 16-core Intel Haswell processors and 128 GB DDR4 2133 MHz memory, and 9,688 nodes each containing a single 68-core 1.4 GHz Intel Xeon Phi 7250 (Knights Landing) processor and 96 GB DDR4 2400 GHz memory. Cori also provides 18 GPU nodes, where each node contains two sockets of 20-core Intel Xeon Gold 6148 2.40 GHz, 384 GB DDR4 memory and 8 NVIDIA V100 GPUs (each with 16 GB HBM2 memory). We use the Haswell processor and the GPU nodes for the results presented in this section.
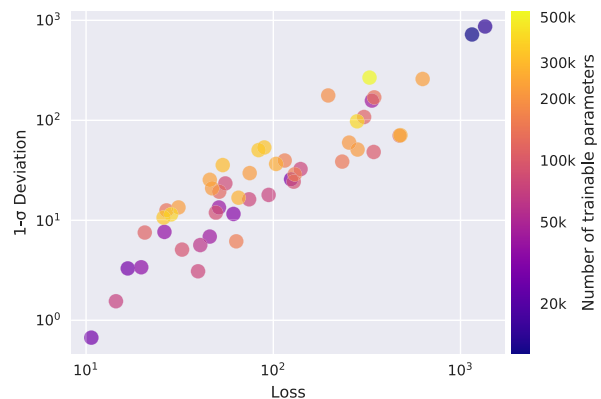

Fig. 9: Scatter plot of median loss value versus median absolute deviation. The loss and median absolute deviation are computed over 50 training trials for each of 50 hyperparameter evaluations. In the bottom left region, a simple architecture (less than 50k trainable parameters) can be found which provides an accurate (low loss) and efficient (low variability) solution.

Complete Sinogram

Sparse Sinogram

Reference Reconstruction

MSE: 1.39e-4, SSIM: 0.765, PSNR: 27.2

0   **1**

Inpainted Sinogram, Optimized Parameters

Inpainted Sinogram, Sub-Optimal Parameters

MSE: 2.12e-5, SSIM: 0.984, PSNR: 35.3
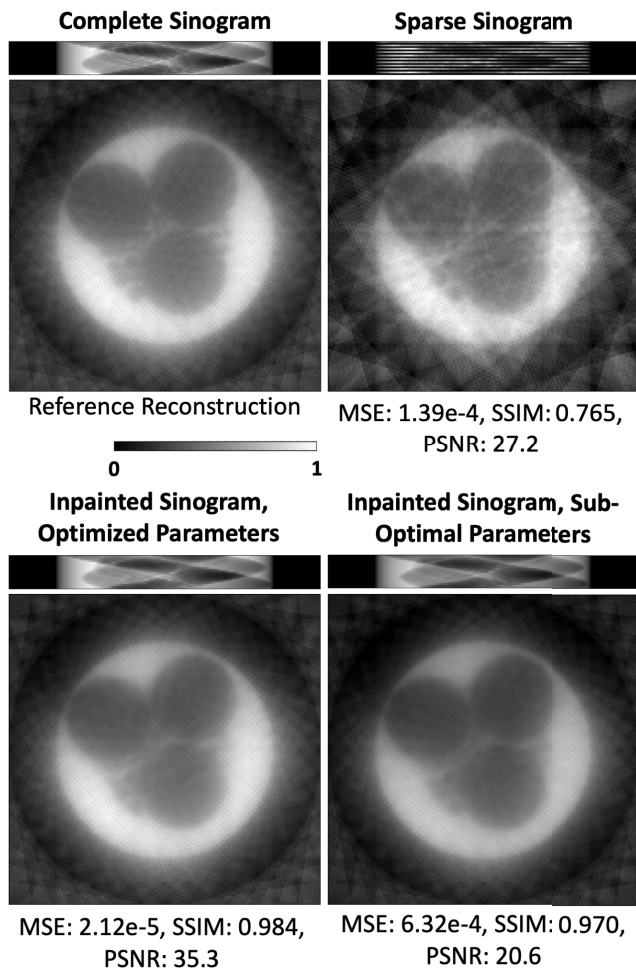
MSE: 6.32e-4, SSIM: 0.970, PSNR: 20.6

Fig. 10: Comparison of the complete sinogram with the sparse and inpainted (neural network output) sinograms for a test example. The reconstruction for each sinogram is performed using the SIRT algorithm, described in Section V-A. Error metrics of per-pixel mean squared error (MSE, lower is better), structural similarity (SSIM, larger is better), and peak signal-to-noise ratio (PSNR, larger is better) compared to the reference reconstruction, showing the improvement of using the inpainted sinogram with optimized network hyperparameters over the sparse sinogram and inpainted sinogram with sub-optimal network hyperparameters.

The first experiment consists of running 50 different hyperparameter evaluations with five trials for each evaluation. Fig. 8 reports the maximum speedup resulting from this experiment across each SLURM step and SLURM task. The neural network is trained for 300 iterations using the entire training dataset of 13500 images. We observe an improvement of two orders of magnitude in speedup between the combination of 1 SLURM task/1 SLURM step and 6 SLURM tasks/16 SLURM steps.

Fig. 9 shows the outer loss function value plotted against the

median absolute deviation. Here, we evaluated each hyperparameter set 50 times to compute the median outer loss function value and standard deviation. An outer loss value of 24.81 is achieved within four iterations when performing Gaussian process surrogate modeling, which may not be guaranteed by random sampling alone, emphasizing the importance of HPO in this case study.

Sparse Sinogram

Inpainted Sinogram, Optimized Parameters

Inpainted Sinogram, Sub-Optimal Parameters

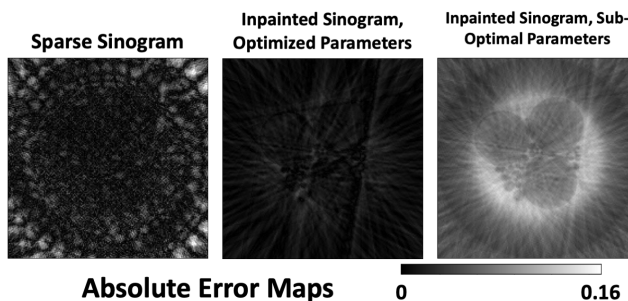**Absolute Error Maps**   **0**   **0.16**

Fig. 11: The absolute error maps for each example in Fig. 10. The baseline comparison for each error map is the complete sinogram.

We also assess the performance of the U-Net after running HPO, comparing the quality of the reconstruction obtained with the neural network output to the original (complete) sinogram. The optimal network hyperparameters found with HPO show improved reconstructions compared to other hyperparameters sampled during HPO (see Figs. 10 and 11, and Table I) demonstrating the importance of HPO to this case study. We also show the results of training the network for values at the lower and upper bounds (set by the user) of all eight hyperparameters, which define the hyperparameter search space. For the results in Figs. 10 and 11, and Table I, each neural network architecture was trained for 100000 iterations. We use the structural similarity index measure (SSIM) to quantify how similar the sparse and inpainted sinogram reconstructions are to the complete sinogram reconstruction. SSIM values range between -1 and 1, with values closer to 1 indicating very similar structures between the two images being compared. SSIM, unlike MSE and PSNR, does not measure absolute error, allowing structural differences to be characterized.

VI. Discussions

In the previous sections we demonstrated different features of our HYPPO software for finding optimal hyperparameters of neural nets. Although our results are promising on different applications, two aspects of our HYPPO method require further development in order to improve its efficiency. First, a sensitivity analysis (SA) of the model's performance regarding the hyperparameters is needed. If we could identify the subset of hyperparameters that most impact the model's performance, we could significantly reduce the number of hyperparameter sets that need to be tried during the optimization because the search space would be smaller. Thus, additional savings in optimization time could be achieved. Second, our current

initial experimental design is created by randomly sampling integer values from the hyperparameter space. Using a type of space-filling design (e.g., a low-discrepancy sequence) instead would be preferable, and thus our initial surrogate models could be improved. However, there are obstacles for both space-filling designs and sensitivity analysis when parameters have integer constraints, as in our problem. Off-the-shelf SA methods such as the ones implemented in the SALib open-source library in Python [45] only work for continuous parameters. Low discrepancy sampling methods such as Sobol's sequences [46] generate evenly distributed points across the sample space, avoiding large clusters and gaps between the points. However, these methods are not easily modified for integer constraints and must be developed further. Simply rounding or truncating continuous values to obtain integers does not deliver the required sample characteristics for SA and sample designs to be maximally effective. However, it is possible to formulate and solve an integer optimization problem to achieve the desired sample distribution. There is also an opportunity to modify the computation of the sample points in Sobol's sequences. These aspects will be a future feature in our software.

Another aspect in our HYPPO software that requires further analysis is the parameters, including initial experimental design sizes, the weights w_T and w_D used to balance the importance of expected performance and uncertainty, the number of times a given hyperparameter set should be trained, and how many hyperparameters should be tried in total. The weights w_T and w_D are user-defined and should reflect the user's averseness to model variability. The number of hyperparameters to be tried in total usually depends on the amount of available compute budget. The size of the initial experimental design influences how well the initial surrogate model approximates the objective function. Typically, larger initial experimental designs give better surrogate model performance at first, but it also means that fewer adaptively selected hyperparameters will be evaluated due to the compute budget. Another potentially impactful parameter is the dropout rate. Currently, a default value is used, but in the future we will include it in our hyperparameters to be optimized. Lastly, in this study we did not take noise in the data into account. In the future we will analyze how small variations in the training data propagate through the network and impact the predictive performance and reliability of the DL models.

VII. Conclusions

In this paper, we demonstrated a new method for conducting hyperparameter optimization of deep neural networks while taking into account the prediction uncertainty that arises from using stochastic optimizers for training the models. Computationally cheap surrogate models are exploited to reduce the number of expensive model trainings that have to be done. We showed the quality of the solutions found on a variety of datasets and how asynchronous nested parallelism can be exploited to significantly accelerate the time-to-solution. The HYPPO software comes with a number of features, allowing the user to conduct simple HPO, UQ, and HPO under uncertainty. Model variability that arises from using stochastic optimizers when training models is rarely addressed in ML literature, but it can have a considerable effect, particularly in scientific applications for which dataset sizes are often limited. High variability of the model performance can have a significant impact on decisions being made, and these decisions should be made with confidence by using reliable models. Our software is a first step towards providing these much needed reliable and robust models. Finally, HYPPO was developed as an open-source software and will be made publicly available in the future via the `pip` Python package manager. More information about the software, including a detailed documentation can be found at https://hpo-uq.gitlab.io/.

Reproducibility

As we strive to make this research fully transparent, step-by-step instructions are provided in the HYPPO online documentation (see link in the previous section) to allow complete reproduction of this work.

Acknowledgement

References

[1] J. Pathak, M. Mustafa, K. Kashinath, E. Motheau, T. Kurth, and M. Day, "Using machine learning to augment coarse-grid computational fluid dynamics simulations," 2020.

[2] J. Müller, J. Park, R. Sahu, C. Varadharajan, B. Arora, B. Faybishenko, and D. Agarwal, "Surrogate optimization of deep neural networks for groundwater predictions," *Journal of Global Optimization*, May 2020. [Online]. Available: https://doi.org/10.1007/s10898-020-00912-0

[3] P. Krawczuk, G. Papadimitriou, S. Nagarkar, M. Kiran, A. Mandal, and E. Deelman, "Anomaly detection in scientific workflows using end-to-end execution gantt charts and convolutional neural networks," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3437359.3465597

[4] Y. F. Cheng, M. Strachan, Z. Weiss, M. Deb, D. Carone, and V. Ganapati, "Illumination pattern design with deep learning for single-shot Fourier ptychographic microscopy," *Optics Express*, vol. 27, no. 2, pp. 644–656, Jan. 2019. [Online]. Available: https://doi.org/10.1364/OE.27.000644

[5] A. Robey and V. Ganapati, "Optimal physical preprocessing for example-based super-resolution," *Optics Express*, vol. 26, no. 24, pp. 31 333–31 350, Nov. 2018. [Online]. Available: https://doi.org/10.1364/OE.26.031333

[6] S. Lapointe, S. Mondal, and R. A. Whitesides, "Data-driven selection of stiff chemistry ode solver in operator-splitting schemes," *Combustion and Flame*, vol. 220, pp. 133–143, 2020.

[7] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017.

[8] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014–. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[9] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 2951–2959.

[10] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[11] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, p. I–115–I–123.

[12] S. Y. Mudugandla, "10 hyperparameter optimization frameworks." https://towardsdatascience.com/10-hyperparameter-optimization-frameworks-8bc87bc8b7e3, year=2020.

[13] P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild, "Deephyper: Asynchronous hyperparameter search for deep neural networks," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, 2018, pp. 42–51.

[14] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2834892.2834896

[15] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm," 2020.

[16] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3053–3062. [Online]. Available: http://proceedings.mlr.press/v80/liang18b.html

[17] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6765–6816, Jan. 2017.

[18] M. Powell, *Advances in Numerical Analysis, vol. 2: wavelets, subdivision algorithms and radial basis functions. Oxford University Press, Oxford, pp. 105-210*. Oxford University Press, London, 1992, ch. The Theory of Radial Basis Function Approximation in 1990.

[19] G. Matheron, "Principles of geostatistics," *Economic Geology*, vol. 58, pp. 1246–1266, 1963.

[20] D. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of Global Optimization*, vol. 21, pp. 345–383, 2001.

[21] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1050–1059. [Online]. Available: http://proceedings.mlr.press/v48/gal16.html

[22] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.

[24] A. Labach, H. Salehinejad, and S. Valaee, "Survey of dropout methods for deep neural networks," *ArXiv*, vol. abs/1904.13310, 2019.

[25] R. Regis and C. Shoemaker, "A stochastic radial basis function method for the global optimization of expensive functions," *INFORMS Journal on Computing*, vol. 19, pp. 497–509, 2007.

[26] D. Jones, M. Schonlau, and W. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998.

[27] O. Tange, *GNU Parallel 2018*. Ole Tange, Apr. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1146014

[28] L. D. Wise, C. T. Winkelmann, B. Dogdas, and A. Bagchi, "Micro-computed tomography imaging and analysis in developmental biology and toxicology," *Birth Defects Research Part C: Embryo Today: Reviews*, vol. 99, no. 2, pp. 71–82, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bdrc.21033

[29] G. D. Rubin, "Computed Tomography: Revolutionizing the Practice of Medicine for 40 Years," *Radiology*, vol. 273, no. 2S, pp. S45–S74, Nov. 2014. [Online]. Available: http://pubs.rsna.org/doi/10.1148/radiol.14141356

[30] S. Garcea, Y. Wang, and P. Withers, "X-ray computed tomography of polymer composites," *Composites Science and Technology*, vol. 156, pp. 305–319, Mar. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0266353817312460

[31] V. Cnudde and M. N. Boone, "High-resolution X-ray computed tomography in geosciences: A review of the current technology and applications," *Earth-Science Reviews*, vol. 123, pp. 1–17, Aug. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S001282521300069X

[32] D. Y. Parkinson, D. M. Pelt, T. Perciano, D. Ushizima, H. Krishnan, H. S. Barnard, A. A. MacDowell, and J. Sethian, "Machine learning for micro-tomography," in *Developments in X-Ray Tomography XI*, vol. 10391. International Society for Optics and Photonics, Sep. 2017, p. 103910J. [Online]. Available: https://doi.org/10.1117/12.2274731

[33] D. Pelt, J. Batenburg, and J. Sethian, "Improving tomographic reconstruction from limited data using Mixed-Scale Dense

convolutional neural networks," *Journal of Imaging*, vol. 4, no. 11, pp. 128–128, Oct. 2018. [Online]. Available: https://ir.cwi.nl/pub/28279

[34] D. Ayyagari, N. Ramesh, D. Yatsenko, T. Tasdizen, and C. Atria, "Image reconstruction using priors from deep learning," in *Medical Imaging 2018: Image Processing*, vol. 10574. International Society for Optics and Photonics, Mar. 2018, p. 105740H. [Online]. Available: https://doi.org/10.1117/12.2293766

[35] Y. Huang, T. Würfl, K. Breininger, L. Liu, G. Lauritsch, and A. Maier, "Some investigations on robustness of deep learning in limited angle tomography," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, A. F. Frangi, J. A. Schnabel, C. Davatzikos, C. Alberola-López, and G. Fichtinger, Eds. Cham: Springer International Publishing, 2018, pp. 145–153.

[36] S. Bazrafkan, V. Van Nieuwenhove, J. Soons, J. De Beenhouwer, and J. Sijbers, "Deep Learning Based Computed Tomography Whys and Wherefores," *arXiv:1904.03908 [cs, eess]*, Apr. 2019. [Online]. Available: http://arxiv.org/abs/1904.03908

[37] L. Fu and B. D. Man, "A hierarchical approach to deep learning and its application to tomographic reconstruction," in *15th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, vol. 11072. International Society for Optics and Photonics, May 2019, p. 1107202. [Online]. Available: https://doi.org/10.1117/12.2534615

[38] J. He and J. Ma, "Radon inversion via deep learning," in *Medical Imaging 2019: Physics of Medical Imaging*, vol. 10948. International Society for Optics and Photonics, Mar. 2019, p. 1094810. [Online]. Available: https://doi.org/10.1117/12.2511643

[39] H. Lee, J. Lee, and S. Cho, "View-interpolation of sparsely sampled sinogram using convolutional neural network," in *Medical Imaging 2017: Image Processing*, vol. 10133. International Society for Optics and Photonics, Feb. 2017, p. 1013328.

[40] Z. Li, W. Zhang, L. Wang, A. Cai, N. Liang, B. Yan, and L. Li, "A sinogram inpainting method based on generative adversarial network for limited-angle computed tomography," in *15th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, vol. 11072. International Society for Optics and Photonics, May 2019, p. 1107220.

[41] D. J. Ching and D. Gürsoy, "Xdesign: an open-source software package for designing x-ray imaging phantoms and experiments," *Journal of Synchrotron Radiation*, vol. 24, no. 2, pp. 537–544, 2017.

[42] D. Gürsoy, F. De Carlo, X. Xiao, and C. Jacobsen, "Tomopy: a framework for the analysis of synchrotron tomographic data," *Journal of synchrotron radiation*, vol. 21, no. 5, pp. 1188–1193, 2014.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[44] P. Gilbert, "Iterative methods for the three-dimensional reconstruction of an object from projections," *Journal of Theoretical Biology*, vol. 36, no. 1, pp. 105–117, 1972. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022519372901804

[45] J. Herman and W. Usher, "SALib: An open-source python library for sensitivity analysis," *The Journal of Open Source Software*, vol. 2, no. 9, jan 2017. [Online]. Available: https://doi.org/10.21105/joss.00097

[46] I. M. Sobol, "Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates," *Mathematics and computers in simulation*, vol. 55, no. 1-3, pp. 271–280, 2001.