Recurrent Neural Network Architecture Search for Geophysical Emulation

A Preprint

Romit Maulik
Argonne Leadership Computing Facility
Argonne National Laboratory
Lemont IL 60439, USA
rmaulik@anl.gov

Romain Egele
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont IL 60439, USA
romainegele@gmail.com

Bethany Lusch
Argonne Leadership Computing Facility
Argonne National Laboratory
Lemont IL 60439, USA
blusch@anl.gov

Prasanna Balaprakash
Mathematics and Computer Science Division
&
Argonne Leadership Computing Facility
Argonne National Laboratory
Lemont IL 60439, USA
pbalapra@anl.gov

April 24, 2020

Abstract

Developing surrogate geophysical models from data is a key research topic in atmospheric and oceanic modeling because of the large computational costs associated with numerical simulation methods. Researchers have started applying a wide range of machine learning models, in particular neural networks, to geophysical data for forecasting without these constraints. However, constructing neural networks for forecasting such data is nontrivial and often requires trial and error. To that end, we focus on developing proper-orthogonal-decomposition-based long short-term memory networks (POD-LSTMs). We develop a scalable neural architecture search for generating stacked LSTMs to forecast temperature in the NOAA Optimum Interpolation Sea-Surface Temperature data set. Our approach identifies POD-LSTMs that are superior to manually designed variants and baseline time-series prediction methods. We also assess the scalability of different architecture search strategies on up to 512 Intel Knights Landing nodes of the Theta supercomputer at the Argonne Leadership Computing Facility.

Keywords LSTMs · AutoML · Emulators · Geophysics

1 Introduction

Geophysical forecasting remains an active area of research because of its profound implications for economic planning, disaster management, and adaptation to climate change. Traditionally, forecasts for geophysical applications have relied on the confluence of experimental observations, statistical analyses, and high-performance computing. However, the high-performance computing aspect of forecasting has traditionally been limited to ensemble partial differential equation (PDE)-based forecasts of different weather models (for example, [39, 34, 45]). Recently, there has been an abundance of publicly available weather data through modern techniques for remote sensing, experimental observations, and data assimilation within numerical weather predictions. Consequently, many researchers have attempted to utilize data for more effective forecasts of geophysical processes [37, 11, 6, 31, 23]. For example, researchers have started building data-driven forecasts by emulating the evolution of the weather *nonintrusively* [9, 8, 10]. In this approach, forecasts are based on data-driven models by eschewing numerical equations. One rationale for these types of predictions is that equation-based forecasts are inherently limited since they do not capture all the relevant physical processes of the atmosphere or the oceans. More important, the data-driven models are attractive because they promise the possibility of overcoming the traditional limitations of equation-based models based on numerical stability and time to solution. Indeed, nonintrusive surrogate models for PDE-based systems have found popularity in many engineering applications [41, 42, 18, 29] because they have been successful in reducing computational simulation campaigns for product design or in complex systems control [28, 27, 22, 35].

We focus on a particularly promising approach for nonintrusive modeling (or forecasting) involving the use of linear dimensionality reduction followed by recurrent neural network time evolution [21, 25]. This forecast technique compresses the spatiotemporal field into its dominant principal components using proper orthogonal decomposition (POD) (also known as principal components analysis) [15, 5]. Following this, the coefficients of each component are evolved by using a time series method. In recent literature, long short-term memory networks (LSTMs), a variant of recurrent neural networks, have been used extensively for modeling temporally varying POD coefficients [30, 19]. The construction of an LSTM architecture for this purpose is generally based on trial and error, requires human expertise, and consumes significant development time. To that end, we devise an automated way of developing POD-LSTM using a neural architecture search (NAS) for a real-world geophysical data set, the NOAA Optimum Interpolation Sea-Surface Temperature (SST) data set, which represents a contribution over past POD-LSTM studies that have studied academic data sets alone. In particular, we leverage the NAS framework of DeepHyper [4] to automate the discovery of stacked LSTM architectures that evolve in time POD coefficients for spatiotemporal data sets. DeepHyper is a scalable open-source hyperparameter and NAS package that was previously assessed for automated discovery of fully connected neural networks on tabular data. In this study, we extend DeepHyper's capabilities for discovering stacked LSTM architectures by parameterizing the space of stacked LSTM architectures as a directed acyclic graph. We adopt the scalable infrastructure of DeepHyper using different search methods for POD-LSTM development. A schematic that describes our overall approach is shown in Figure 1. The main contributions of this work are as follows.

- We develop an automated NAS approach to generate stacked LSTM architectures for POD-LSTM to forecast the global sea-surface temperature on the NOAA Optimum Interpolation Sea-Surface Temperature (SST) data set.
- We improve the scalability of the NAS within DeepHyper by implementing aging evolution, an asynchronous evolutionary algorithm and demonstrate its efficacy for developing POD-LSTM.
- We compare aging evolution with reinforcement learning and random search methods at scale on up to 512 nodes of the Theta supercomputer and show that the proposed approach has a faster time to solution and better node utilization.
- We show that automatically obtained POD-LSTM compare favorably to manually designed variants and baseline machine learning forecast tools.

2 Data set and preprocessing

In this section we describe the data set and the POD technique we used for data compression.

2.1 NOAA Optimum Interpolation Sea Surface Temperature data set

For our geophysical emulation we utilize the open-source NOAA Optimum Interpolation SST V2 data set.[1] Seasonal fluctuations in this data set cause strong periodic structure, although complex ocean dynamics still lead to rich

[1]Available at https://www.esrl.noaa.gov/psd/

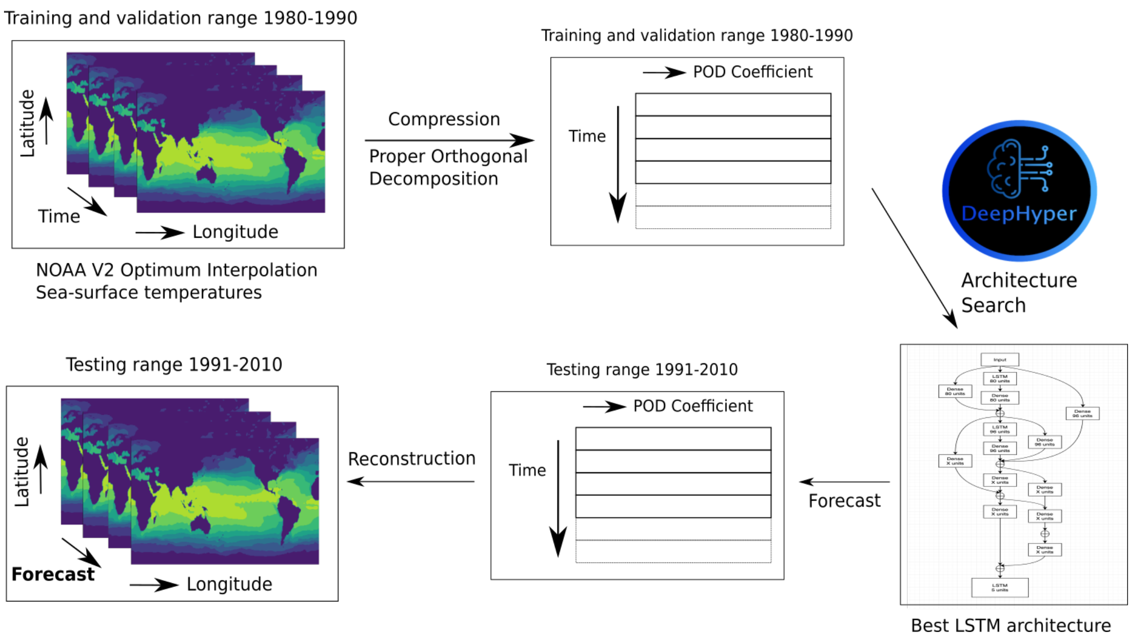


Figure 1: Our proposed NAS approach for automated POD-LSTM development. Snapshots of spatiotemporally varying training data are compressed by using proper orthogonal decomposition to generate reduced representations that vary with time. These representations (or coefficients) are used to train stacked LSTMs that can forecast on test data. The POD basis vectors obtained from the training data are retained for reconstruction using the forecast coefficients.

phenomena. Temperature snapshot data is available on a weekly basis on a one-degree grid. This data set has previously been used in data-driven forecasting and analysis tasks (for instance, see [16, 7]) particularly from the point of view of identifying seasonal and long-term trends for ocean temperatures by latitude and longitude. Each "snapshot" of data comprises a temperature field in an array of size 180×360 (i.e., the latitudes and longitudes on a one-degree resolution grid), which corresponds to the average sea-surface temperature magnitude for that week. Prior to its utilization for forecasting, a mask is used to remove missing locations in the array that correspond to the land area. Following the application of the mask, the nonzero data points are flattened to obtain an $\mathbb{R}^Z$-dimensional vector as our final snapshot for a week.

This data is available from 1980 to 2010 (i.e., 1,907 snapshots). We utilize the time period of 1980–1990 for training and validation (427 snapshots) and 1990 to 2010 for testing (1,487 snapshots). The training data is utilized to obtain data points given by a window of inputs and a window of outputs corresponding to the desired task of forecasting the future, given observations of the past sea surface temperatures. Further specifics of the forecasting (i.e., the window of history interpreted as input and the length of the forecast) will be discussed later in Section 2.2. These data points are then split into training and validation sets. We note that this forecast is performed non-autoregressively—that is, the data-driven method *is not* utilized for predictions beyond the desired window size. Since this data set is produced by combining local and satellite temperature observations, it represents an attractive forecasting task for data-driven methods.

2.2 Compression and forecasting

Here, we first review the POD technique for the construction of a reduced basis [15, 5] for data compression (see [41] for POD and its relationship with other dimension-reduction techniques). The POD procedure is tasked with identifying a space that approximates snapshots of a signal optimally with respect to the L^2-norm. The process of orthogonal basis ($\boldsymbol{\vartheta}$) generation commences with the collection of snapshots in the *snapshot matrix*

$$\mathbf{S} = \left[\begin{array}{c|c|c|c} \hat{\mathbf{q}}^1_h & \hat{\mathbf{q}}^2_h & \cdots & \hat{\mathbf{q}}^{N_s}_h \end{array} \right] \in \mathbb{R}^{N_h \times N_s}, \tag{1}$$

where N_s is the number of snapshots and $\hat{\mathbf{q}}^i_h \in \mathbb{R}^{N_h}$ corresponds to an individual $\mathbb{R}^{N_h}$ degree-of-freedom snapshot in time of the discrete solution domain with the mean value removed, namely,

$$\hat{\mathbf{q}}_h^i = \mathbf{q}_h^i - \bar{\mathbf{q}}_h, \quad \bar{\mathbf{q}}_h = \frac{1}{N_s}\sum_{i=1}^{N_s}\mathbf{q}_h^i, \tag{2}$$

where $\overline{\mathbf{q}}_h \in \mathbb{R}^{N_h}$ is the time-averaged solution field. We note that $\mathbf{q}_h^i$ may be assumed to be any multidimensional signal that is subsequently flattened. Within the context of our geophysical data sets, these correspond to the flattened land or sea-surface temperature snapshots. Our POD bases can then be extracted efficiently through the method of snapshots where we solve the eigenvalue problem on the correlation matrix $\mathbf{C} = \mathbf{S}^T\mathbf{S} \in \mathbb{R}^{N_s \times N_s}$. Then

$$\mathbf{C}\mathbf{W} = \mathbf{W}\Lambda, \tag{3}$$

where $\Lambda = \operatorname{diag}\{\lambda_1, \lambda_2, \cdots, \lambda_{N_s}\} \in \mathbb{R}^{N_s \times N_s}$ is the diagonal matrix of eigenvalues and $\mathbf{W} \in \mathbb{R}^{N_s \times N_s}$ is the eigenvector matrix. Our POD basis matrix can then be obtained by

$$\boldsymbol{\vartheta} = \mathbf{S}\mathbf{W} \in \mathbb{R}^{N_h \times N_s}. \tag{4}$$

In practice, a reduced basis $\boldsymbol{\psi} \in \mathbb{R}^{N_h \times N_r}$ is built by choosing the first N_r columns of $\boldsymbol{\vartheta}$ for the purpose of efficient ROMs, where $N_r \ll N_s$. This reduced basis spans a space given by

$$\mathbf{X}^r = \operatorname{span}\left\{\boldsymbol{\psi}^1, \ldots, \boldsymbol{\psi}^{N_r}\right\}. \tag{5}$$

The coefficients of this reduced basis (which capture the underlying temporal effects) may be extracted as

$$\mathbf{A} = \boldsymbol{\psi}^T\mathbf{S} \in \mathbb{R}^{N_r \times N_s}. \tag{6}$$

The POD approximation of our solution is then obtained via

$$\hat{\mathbf{S}} = [\ \tilde{\mathbf{q}}_h^1 \mid \tilde{\mathbf{q}}_h^2 \mid \cdots \mid \tilde{\mathbf{q}}_h^{N_s}\ ] \approx \boldsymbol{\psi}\mathbf{A} \in \mathbb{R}^{N_h \times N_s}, \tag{7}$$

where $\tilde{\mathbf{q}}_h^i \in \mathbb{R}^{N_h}$ corresponds to the POD approximation to $\hat{\mathbf{q}}_h^i$. The optimal nature of reconstruction may be understood by defining the relative projection error,

$$\frac{\sum_{i=1}^{N_s}\left\|\hat{\mathbf{q}}_h^i - \tilde{\mathbf{q}}_h^i\right\|_{\mathbb{R}^{N_h}}^2}{\sum_{i=1}^{N_s}\left\|\hat{\mathbf{q}}_h^i\right\|_{\mathbb{R}^{N_h}}^2} = \frac{\sum_{i=N_r+1}^{N_s}\lambda_i^2}{\sum_{i=1}^{N_s}\lambda_i^2}, \tag{8}$$

which shows that with increasing retention of POD bases, increasing reconstruction accuracy may be obtained.

Following compression, the overall forecast task may be formulated after recognizing that the rows of $\mathbf{A}$ contain information about the different POD modes and the columns correspond to their varying information in time. Therefore, one approach for forecasting is to predict the evolution of the N_r coefficients in time. Once a forecast has been performed, the first N_r bases may be used to reconstruct the snapshot (in the future). A popular approach for this forecasting task is to use data-driven time series methods. This is motivated by the fact that equations for the evolution of coefficients are nontrivial in the reduced basis. We can then generate training data by extracting the coefficients in $\mathbf{A}$ in a windowed-input and windowed-output form to completely define our forecast task. The state at a given time is represented by an N_r-dimensional column vector of POD coefficients. Given N_s snapshots of data, we choose every sub-interval of width $2K$ as an example, where K snapshots are the input and K snapshots are the output. We utilize a randomly sampled 80% of examples for training and utilize the remaining 20% for validation. For our NOAA SST data set, we have $N_s = 427$ snapshots, and we choose $K = 8$, resulting in 1,111 examples. We note that we avoid any potential data leakage by testing on data points generated in entirely different years of our data (no overlap between training and testing data).

3 NAS using DeepHyper

In this section, we describe the stacked LSTM search space for POD-LSTM and the NAS methods employed to explore it.

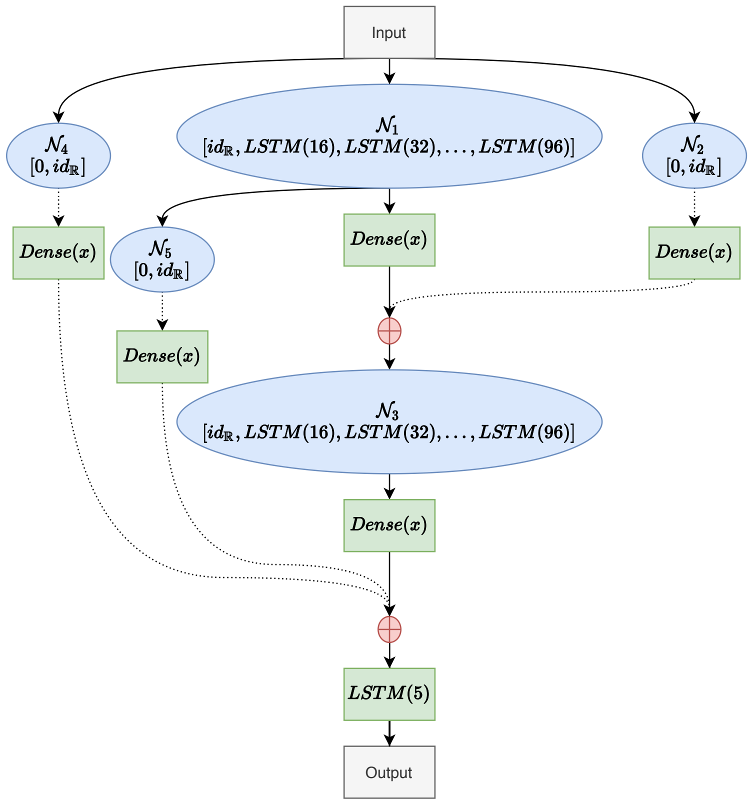

Input
$\mathcal{N}_4$ $[0, id_{\mathbb{R}}]$
$\mathcal{N}_1$ $[id_{\mathbb{R}}, LSTM(16), LSTM(32), \ldots, LSTM(96)]$
$\mathcal{N}_2$ $[0, id_{\mathbb{R}}]$
$\mathcal{N}_5$ $[0, id_{\mathbb{R}}]$
$Dense(x)$
$\mathcal{N}_3$ $[id_{\mathbb{R}}, LSTM(16), LSTM(32), \ldots, LSTM(96)]$
$LSTM(5)$
Output

Figure 2: An example stacked LSTM search space for POD-LSTM with two variable LSTM nodes in blue, $\mathcal{N}_1$ and $\mathcal{N}_3$. The skip-connection variable nodes are $\mathcal{N}_2$, $\mathcal{N}_4$, and $\mathcal{N}_5$. Dotted lines represent possible skip connections. The last layer is a constant LSTM(5) node to match the output dimension of five.

3.1 Stacked LSTM search space

In DeepHyper, the search space of the neural architecture is represented as a directed acyclic graph. The nodes representing inputs and outputs of the deep neural network are fixed and respectively denoted as $\mathcal{I}_i$ and $\mathcal{O}_j$. All other nodes $\mathcal{N}_k$ are called intermediate nodes, each with a list of operations (choices). Each intermediate node is either a constant, a variable, or a skip connection node. A constant node's list contains only one operation, while a variable node's list contains multiple options. The formation of skip connections between the variable nodes is enabled by a skip-connection variable node. Given three nodes $\mathcal{N}_{k-1}, \mathcal{N}_k$ and $\mathcal{N}_{k+1}$, the skip connection node allows for the possible construction of a direct connection between $\mathcal{N}_{k-1}$ and $\mathcal{N}_{k+1}$. This skip connection node will have two operations, zero for no skip connection and identity for skip connection. In a skip connection, the tensor output from $\mathcal{N}_{k-1}$ is passed through a dense layer and a sum operator. Since the skip connections can be formed between variable nodes that have a different number of neurons, the dense layer is used to project the incoming tensor to the right shape when the skip connection is formed. The sum operator adds the two tensors from $\mathcal{N}_{k-1}$ and $\mathcal{N}_k$ and passes the resulting tensor to $\mathcal{N}_{k+1}$. Without loss of generality, the same process can be followed for any number of nodes. For example, in the case of three nodes, two skip-connection nodes will be inserted before an incumbent node. For the stacked LSTM discovery, we define m variable LSTM nodes, where each node is an LSTM layer with a list of numbers of neurons as possible operations. Figure 2 shows an example LSTM search space with two variable nodes.

We note that the input and output nodes are determined by the shape of our training data and are immutable. We also note that the second dimension of a tensor that is transformed from input to output is kept constant for all experiments. This aligns with the temporal dimension of an LSTM and is not perturbed.

3.2 Algorithms for architecture discovery

We use search methods in DeepHyper to choose from a set of possible integer values at each variable node. At the LSTM variable nodes, this choice decides the number of hidden layer neurons. At the skip connection variable nodes, this choice decides connections to previous layers of the architecture. For intelligently searching this space, we have implemented a recently introduced completely asynchronous evolutionary algorithm called aging evolution (AE) [33] within DeepHyper. In addition, DeepHyper supports two search methods for NAS: a parallel version of reinforcement learning (RL) [4, 46] based on the proximal policy optimization [38], and a random search (RS).

3.2.1 Aging evolution

AE searches for new architectures by performing mutations without crossovers on existing architectures within a population. At the start of the search, a population of p architectures is initialized randomly, and the fitness metric (for validation accuracy) is recorded. Following this initialization, samples of size s are drawn randomly without replacement. A mutation is performed on the architecture with the highest accuracy within each sample (the parent) to obtain a new (child) architecture. A mutation corresponds to choosing a different operation for one variable node in the search space. This is achieved by first randomly sampling a variable node and then choosing (again at random) a value for that node excluding the current value. The validation accuracy of the child architecture is recorded. Following this, the child is added to the population by replacing the oldest member of the population. For the purpose of mutation, an architecture is interpreted to be a sequence of integers, and certain probabilistic perturbations to this sequence are performed to obtain new architectures. Over multiple cycles, better architectures are obtained through repeated sampling and mutation. The sampling and mutation operations are inexpensive and can be performed quickly. When AE completes an evaluation, another architecture configuration for training is obtained by performing a mutation of the previously evaluated architectures (stored for the duration of the experiment in memory) and does not require any communication with other compute nodes.

3.2.2 Distributed RL method

RL is a framework where an agent interacts (or multiple agents interact) with an environment by performing actions and collecting rewards and observations from this same environment. In our case, actions correspond to operation choices for variable nodes in the NAS search space. The reward is the accuracy computed on the validation set. The RL method in DeepHyper uses proximal policy optimization [38] with a loss function of the form

$$J_t(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t], \tag{9}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the ratio of action probabilities under the new and old policies; the clip/median operator ensures that the ratio is in the interval $[1-\epsilon, 1+\epsilon]$; and $\epsilon \in (0,1)$ is a hyperparameter (typically set to 0.1 or 0.2). The clipping operation helps stabilize gradient updates. The method adopts the multimaster-multiworker paradigm for parallelization. Each master runs a copy of a policy and value neural network, termed an agent. It generates b architectures and evaluates them in parallel using multiple worker nodes. The b validation metrics are then collected by each agent to do a distributed computation of gradients. The agents perform an all-reduce with the mean operator on gradients and use that to update the policy and value neural network. This procedure was chosen instead of asynchronous update because it has been empirically shown to perform better (see, e.g., [12]).

3.2.3 Random search method

For comparison of all of our methods, we also describe results from a random search algorithm that explores the search space of architectures by randomly assigning operations at each node. This search is embarrassingly parallel: it does not need any internode communication. However, the lack of an intelligent search leads to architectures that do not improve over time or scale. Such results are demonstrated in our experiments as well.

4 Experiments

We used Theta, a 4,392-node, 11.69-petaflop Cray XC40–based supercomputer at the Argonne Leadership Computing Facility. Each node of Theta is a 64-core Intel Knights Landing processor with 16 GB of in-package memory, 192 GB of DDR4 memory, and a 128 GB SSD. The compute nodes are interconnected by an Aries fabric with a file system capacity of 10 PB. The software environment that we used consists of Python 3.6.6, TensorFlow 1.14 [1] and DeepHyper 0.1.7. The NAS API within DeepHyper utilized Keras 2.3.1.

For the stacked LSTM search space, we used 5 LSTM variable nodes ($m = 5$). This resulted in the creation of 9 skip connection variable nodes. The possible operations at each of the LSTM variable node were set to [Identity, LSTM(16), LSTM(32), LSTM(64), LSTM(80), LSTM(96)] representing different layer types: Identity layer, LSTM layer with 16, 32, 64, 80, and 96 units. The dense layers for projection did not have any activation function. After each add operation, the ReLU activation function was applied to the tensor. For this search space, the total number of architectures is 8,605,184.

As a default, we evaluated the three search methods on 128 nodes. For scaling experiments, we used different node counts: 33, 64, 256, and 512. We used 33 (instead of 32) nodes because in RL we set the number of agents to 11 and adapt the number of workers per agent based on the node count as prescribed in [4]. This implies that given any number of compute nodes, 11 are reserved solely for the agents whereas the rest function as workers that are equally distributed to each agent. With 33 nodes, each agent is assigned 2 workers, for a total of 33 compute nodes being utilized. When 64 nodes are used, each agent is allocated 4 workers, for a total of 55 used nodes and 9 unused nodes. Similarly, for 128 compute nodes, each agent is allocated 10 workers, for a total of 121 utilized and 7 unused nodes. For 256 and 512 compute nodes, each agent is provided 22 and 45 workers, resulting in 3 and 6 unused nodes, respectively. The equal division of workers among agents is implemented for simplicity within DeepHyper. For each node count, each search method was run for 3 hours of wall time.

Each evaluation in all three search methods involved training a generated network and returning the validation metric to the search method. The evaluation used only a single node (no multi-node data-parallel training). The mean squared error was used for training the network, and the coefficient of determination (R^2) was used as a metric on the validation data set. The AE and RL methods were tasked with optimizing the R^2 metric by searching a large space of LSTM architectures. The RS method sampled configurations at random without any feedback. The training hyperparameters were kept the same in all of our experiments: batch size of 64, learning rate of 0.001, and 20 epochs of training with the ADAM optimizer [14]. We set the maximum depth of the stacked LSTM network (an upper bound held constant in all our searches) to 5.

To assess the efficiency of the search, we tracked the averaged reward (i.e., the validation accuracy of our architecture) with respect to wall-clock time. To assess scaling, we recorded the averaged node utilization for each search; to assess the overall benefit, we selected the best architecture found during the search and assessed it on the test data set. For this, we performed post-training, where the best-found architecture was retrained from scratch for a longer duration and tested on a completely unseen test data set. We note that our metrics (given by the reward and node utilization) were computed by using a moving window average of window size 100.

4.1 Comparison of different methods

Here, we compare the three search methods and show that AE outperforms RL and RS by achieving a higher validation accuracy in a shorter wall-clock time.

We ran AE, RL, and RS on 128 nodes on Theta. For the asynchronous algorithms of AE and RS, all nodes were considered workers because they are able to evaluate architectures independently of any master. For AE, we used a population size of 100 and a sample size of 10 to execute the mutation of architectures asynchronously.

For RS, random configurations are sampled independently of any other nodes. In contrast, RL relies on a synchronous update where the compute nodes are divided into agents and workers. Each agent is given an equal number of workers to evaluate architectures and is provided rewards before the agent neural networks are updated by using the policy gradients. We fixed the number of agents for this and all subsequent experiments to 11. For a 128-compute node experiment with 11 agents, we had 10 workers per agent node, for a total of 110 workers and 11 agents; 7 nodes remained idle.

Figure 3 shows the search trajectory of validation R^2 of our three search strategies with respect to wall-clock time. We observe that AE reaches a validation R^2 value of 0.96 within 50 minutes. On the other hand, RL exhibits strong exploration in the beginning of the search and reaches a R^2 value comparable to that of AE at 160 minutes. The RS without any feedback mechanism finds architectures with R^2 values between 0.93 and 0.94. The results of RS show the importance of having feedback-based search such as AE and RL.

The superior performance of AE can be attributed to its effective aging mechanism and the resulting regularization, as discussed in [33]. In AE, the individuals in the population die faster; an architecture can stay alive for a long time only through inheritance from parent to child for a number of generations. When that number is passed, the architecture undergoes retraining; and if the retraining accuracy is not high, the architecture is removed from the population. An architecture can remain in the population for a long time only when its retraining accuracy is high for multiple generations. Consequently, the aging mechanism helps navigate the training noise in the search process and provides a regularization mechanism. RL lacks such a regularization mechanism, and the slower convergence can be attributed to the synchronous gradient update mechanism at the inter- and intra-agent levels. Figure 4 shows the average node utilization of the three methods over time. We observe that the node utilization of AE and RS are close to 1, meaning that all the nodes are occupied most of the time. However, the RL node utilization is between 0.5 and 0.6. This can be attributed to two factors: the gradient averaging across agents is synchronous, and each agent needs to wait until all its workers finish their evaluations before computing the agent-specific gradient. In the beginning, each agent will generate a wide range of architectures, and each can have different training time. The worker nodes for a given

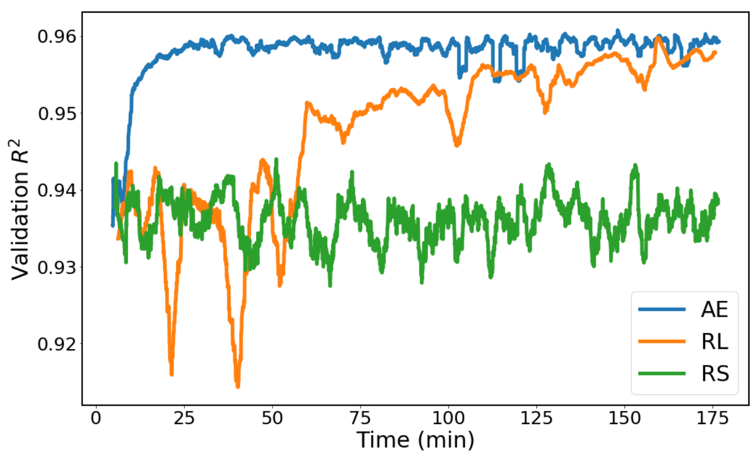


Figure 3: Comparison of search trajectories for AE, RL, and RS for 128 compute nodes on Theta. Each search was run for 3 hours of wall time. AE obtains optimal architectures in a much shorter duration. Both RL and AE are an improvement over random search methods.

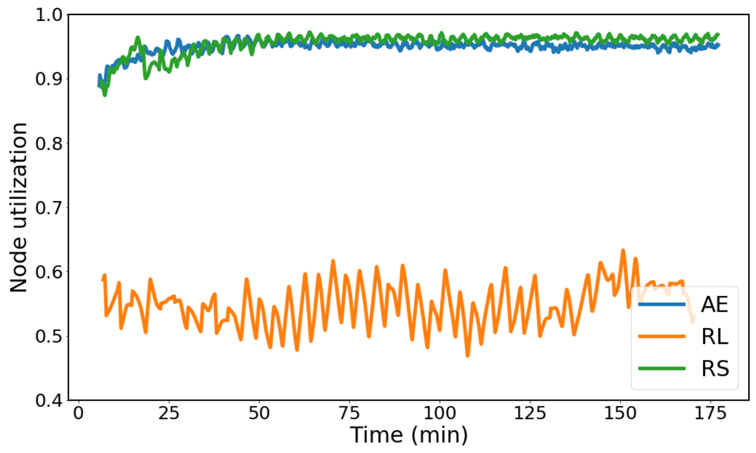


Figure 4: Comparison of node utilization for AE, RL, and RS for 128 compute nodes on Theta. Each search was run for 3 hours of wall time. AE achieves optimal node utilization (almost matching RS) whereas RL is less efficient because of synchronization during the reinforcement learning gradient update.

agent cannot proceed to the next batch of evaluations, and they become idle because one or more worker nodes require more time to finish the training. This situation has previously been observed in [4] within DeepHyper. Note that AE and RS do not have such utilization bottlenecks. Indeed, AE and RS can execute more evaluations (8,806 and 7,267, respectively) compared with RL (4,740 evaluations) for the duration of the search.

4.2 Post-training and science results

To ensure efficient NAS, one commonly solves a smaller problem during the search itself before utilizing the best architecture discovered for the larger training task. This approach helps us explore the large space of neural architectures more efficiently. It also helps us deal with larger data sets if needed. In this study, we utilized fewer epochs of training (20 epochs) during the search and retrained from scratch using a greater number of epochs during post-training (100 epochs). Here, we show that retraining the best-stacked LSTM architecture obtained from AE results in significant improvement.

Figure 5 shows the best architecture found by AE with 128 nodes. One can observe the unusual nature of our network as evidenced by multiple skip connections. We utilized the best architecture found by AE (in terms of validation R^2) for post-training and scientific assessments.

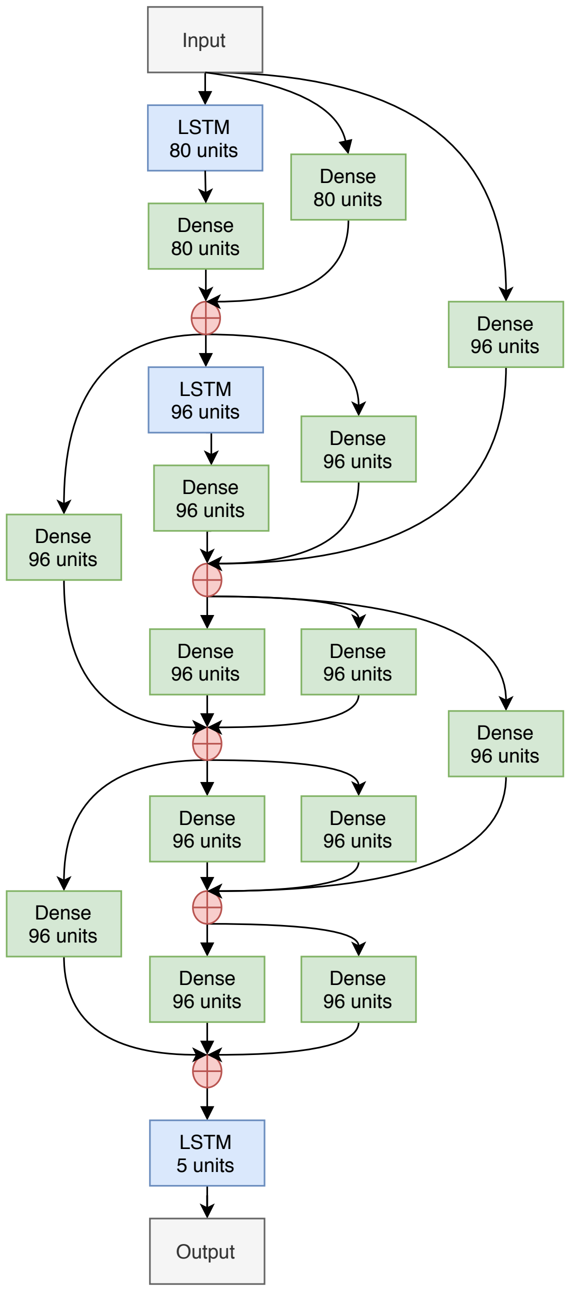


Figure 5: Best-found LSTM architecture for the NOAA SST data set using the aging evolution search strategy on 128 compute nodes of Theta for 3 hours of wall time.

For post-training, we used the same hyperparameters as specified in the NAS with the exception of a longer training duration of 100 epochs (instead of 20 for the search). In addition, our architecture search as well as our post-training utilized a sequence-to-sequence learning task where the historical temperatures (in a sequence) were used to predict a forecast sequence of the same length (i.e., measurements of 8 weeks of sea-surface temperature data were utilized to predict 8 weeks of the same in the future). This may also be seen in the output space of the best-found architectures where the second dimension of the output tensor is the same as the one used for the input.

The results from the post-training are shown in Figure 6, where one can see that the best architecture improves on its validation R^2 metric when training is extended beyond 20 epochs. A final validation R^2 value of 0.985 is observed; this trained architecture is used for our science assessments.

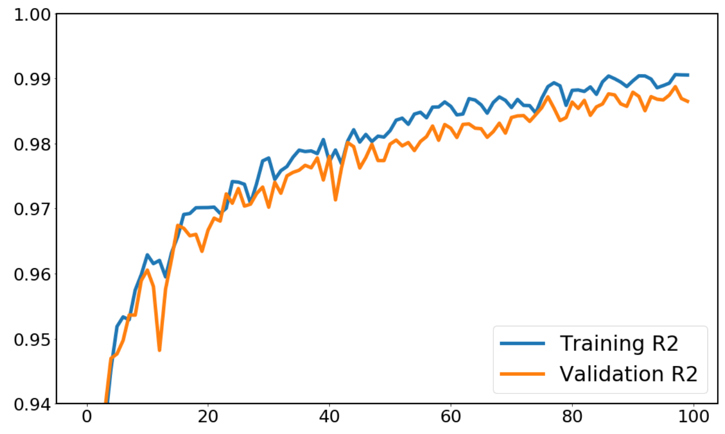


Figure 6: Post-training using the optimal architecture found by DeepHyper for geophysical emulation. The post-training utilized 100 epochs for a complete training of the best architecture found during LSTM search.

Figure 7 shows the forecasts for the POD coefficients on both the training and the testing data sets. While the training data (representing temperatures between 1980 and 1990) predicts very well, the test data (from 1990 to 2010) shows a gradual increase in errors, with later data points far away from the training regime. In this figure, the modes refer to the linear coefficients that premultiply the POD basis vectors prior to reconstruction in the physical space. They may be interpreted to be the contribution of the different basis vectors (and correspondingly the different spatial frequency contributions) in the evolving flow. One can also observe that the amount of stochasticity increases significantly as the modal number increases. This increase may be explained by the fact that while seasonal dynamics are cyclical and lead to repeating patterns at the global scale (mode 1, 2, and 3), small-scale fluctuations may be exceedingly stochastic (mode 4 and beyond). We note that there are no external inputs to our data set and that the networks are utilized for forecasting under the assumption that the past information is always 100% accurate. Simply speaking, the outputs of the LSTM forecast are not reused as inputs for future forecasts. The past is always known *a priori*.

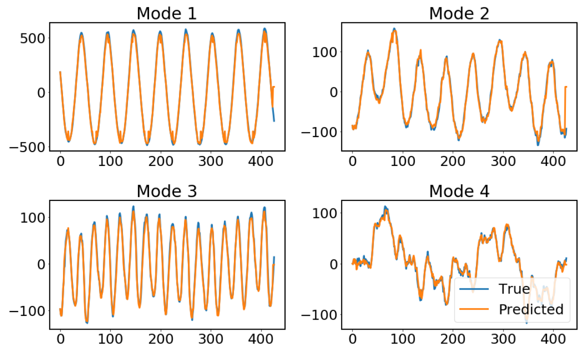
An example forecast (week 248 of the testing data range) is shown in Figure 8, where the larger structures in the temperature field are captured effectively. We note here that the POD-LSTM framework is fundamentally limited by the fact that the spectral content of the predictions can at best match the spectral support of the number of POD modes retained. Therefore, we also include the true fields projected onto the five retained POD modes to indicate how close the LSTM prediction is to the training data itself. Thus, we may interpret this to be training and forecasting on a filtered version of the true data set, where the truncation of the POD components leads to limited recovery of high-frequency information. This effect of filtering is noticeable in Figure 9 where the use of the POD-LSTM method leads to good capture of the seasonal phase in three different probed locations but results in some mismatch in amplitude prediction.

4.3 Comparisons with baseline models

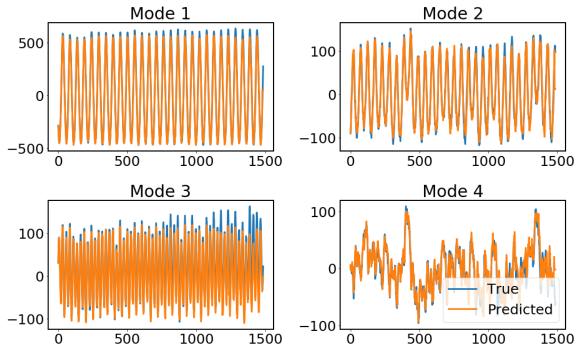
Here, we compare our automatically generated POD-LSTM network with manually generated POD-LSTM networks and with other classical forecasting methods and show the efficacy of our approach.

The classical forecasting assessments use linear, XGBoost, and random forest models within the non-autoregressive framework with no exogenous inputs; in other words, models are fit between an input space corresponding to a historical sequence of temperature to forecast the next sequence. These methods are all deployed within the `fireTS`[2] package; details may be found in [43]. Briefly, if our target is given by $\mathbf{a}(t+1), \mathbf{a}(t+2), \ldots, \mathbf{a}(t+K)$, we fit a data-driven regressor using information from $\mathbf{a}(t-1), \mathbf{a}(t-2), \ldots, \mathbf{a}(t-K)$, where K is now interpreted to be the autoregression order. A prediction is made by using the underlying regressor while assuming that the past information is always coming from the true measurements. We note that the linear model corresponds to the well-known ARIMA framework. Note that all our baseline regression frameworks were implemented through the Scikit-learn package [26] with default configurations. For our manually designed LSTMs, we developed simple stacked architectures and scanned across the number of hidden layer neurons for each cell. Our LSTMs utilized both one and five hidden layers and demonstrated the challenge of manual model selection. These LSTMs also utilized 100 epochs for training.

[2]https://github.com/jxx123/fireTS



(a) NOAA SST training data forecast



(b) NOAA SST testing data forecast

Figure 7: Post-training results with progress to convergence for the optimal architecture showing improved performance for longer training durations (top row). Training and testing forecasts for the NOAA SST data set (bottom row) display the *a posteriori* performance after surrogate training.

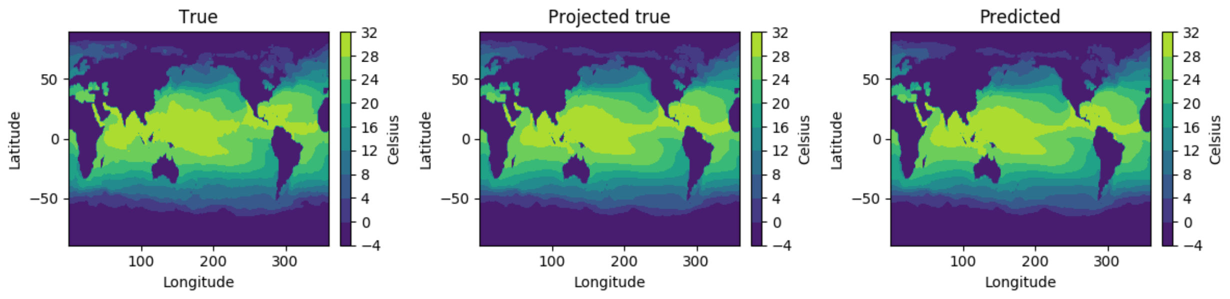
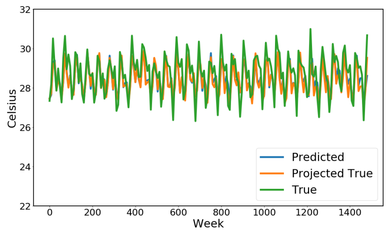
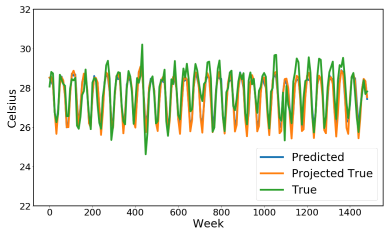


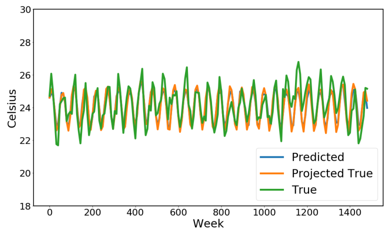
Figure 8: Sample forecast in the testing regime for the NOAA SST data set showing sea surface temperatures in degrees Celsius using the best architecture found by DeepHyper. This snapshot corresponds to week 248 from the start of the testing data range.



(a) 40 ° latitude, 80 ° longitude



(b) 50 ° latitude, 80 ° longitude



(c) 55 ° latitude, 80 ° longitude

Figure 9: Temporal probes for the temperature at three discrete locations on the global grid. While the seasonal phase of the temperature is reconstructed accurately, the POD-LSTM implementation causes reduced accuracy in capturing some peaks and troughs of the probe temperature.

Table 1: Coefficients of determination (R^2) of different forecasting methods on the NOAA SST data set. Here, training data and validation data are obtained from 1980 to 1990, whereas testing data is obtained from 1991 to 2010. Note that the LSTM metrics are expressed in one-layered/five-layered configuration.

Model	**NAS-POD-LSTM**	**Linear**	**XGBoost**	**Random Forest**	**LSTM-40**	**LSTM-80**	**LSTM-120**	**LSTM-200**
1980-1990	**0.985**	0.801	0.966	0.823	0.916/0.944	0.931/0.948	0.922/0.956	0.902/0.963
1991-2010	**0.876**	0.172	-0.056	0.002	0.742/0.687	0.734/0.687	0.746/0.711	0.739/0.724

Table 1 shows the results. The R^2 metrics show that the best architecture found by DeepHyper (denoted NAS-POD-LSTM) outperforms the manually designed LSTMs and classical time-series prediction methods, with the highest R^2 value of 0.876. In particular, the benefit of using LSTMs over their counterparts is easily observed where LSTMs are seen to be more accurate ($R^2 > 0.73$) than the linear model ($R^2 \approx 0.1$), XGBoost ($R^2 \approx -0.1$) and the random forest regressor ($R^2 \approx 0.0$).

4.4 Scaling

Here, we study the scaling behavior of the three search methods. We show that AE scales better than RL and outperforms RS with respect to accuracy. In addition to 128 nodes, we utilized 33, 64, 256, and 512 nodes for the scaling experiments.

Table 2 shows the average node utilization of the search methods for different node counts. The metric used in this table is computed by a ratio for the observed effective node utilization of an experiment against the ideal node utilization (for AE, RS, and RL). The observed effective node utilization over 3 hours of wall time is computed by using an area-under-the-curve (AUC) calculation. Subsequently, this area is divided by the ideal node utilization AUC. Therefore, a metric closer to 1 implies optimal utilization of all compute nodes. We note that the trapezoidal integration rule is utilized for the AUC calculation. We observe that node utilization of AE and RS are similar and are above 0.9 for up to 256 nodes; for 512 nodes, the utilization drops below 0.87. The node utilization for RL is poor, hovering around 0.5 for the different compute node experiments.

In terms of the number of architectures evaluated, we observe that AE consistently is able to evaluate more architecture evaluations than RS and RL can for a fixed wall time (see Table 2). The advantage over RL may be explained by the fact that RL requires a synchronization before obtaining rewards for a batch of architectures (also evidenced by lower node utilization). The improvement over RS may be attributed to the AE algorithm prioritizing architectures with fewer trainable parameters—a fact that has been demonstrated for asynchronous algorithms previously [4]. When 33 compute nodes are used, the AE strategy completes 2,093 evaluations compared with only 1,066 by RL and 1,780 by RS. For 64 compute nodes, the total number of evaluations for AE increases significantly, with 4,201 evaluations performed successfully, whereas RL and RS lead to 2,100 and 3,630 evaluations in the same duration, respectively. As mentioned previously, for 128 nodes, AE performs 8,068 evaluations whereas RL and RS evaluate 4,740 and 7,267 architectures, respectively. For 256 nodes, AE, RL, and RS evaluate 18,39, 9,680, and 15,221 architectures, respectively. Similar trends are seen for 512 nodes, with 33,748, 16335, and 26,559 architectures evaluated by AE, RL, and RS, respectively. We note that the AE strategy evaluates roughly double the number of architectures for all compute node experiments in comparison with RL.

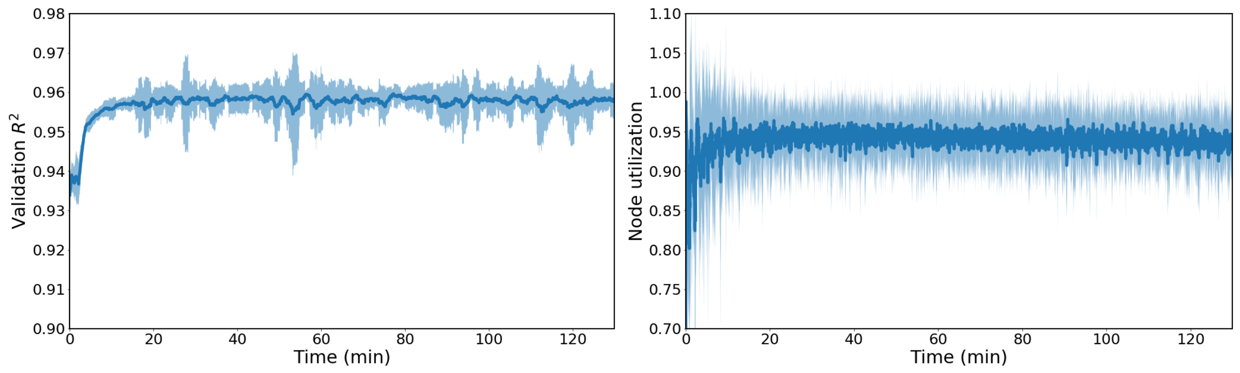
In terms of time to solution, which we define as the time taken (in minutes) to obtain the best validation R^2, all search strategies obtain high-performing architectures even at 33 nodes. These results imply that DeepHyper may be utilized with smaller compute resources to obtain accurate geophysical emulators within the POD-LSTM framework for similar data sets. As expected, we observe that RL and AE obtain higher-performing architectures than RS can.

We can observe a trend in AE, where the time to solution decreases with an increase in the node count. This can be attributed to the large number of architecture evaluations, which increases the chances of finding the high performing architecture in short wall clock time. We did not observe such a trend for RL, which is limited by the scaling bottlenecks. On the other hand, RS shows a trend similar to AE when we moved from 33 to 64; after that we do not have conclusive evidence for strong scaling.

We note that, since AR, RL, RS are randomized search methods, ideally, one would need multiple runs with different random seeds to exactly assess the variability with respect to time to solution. However, running 10 repetitions requires 450 hours (=10 runs $\times$ 3 methods $\times$ 5 node counts $\times$ 3 hours) if we run each job sequentially. Therefore, we analyzed the general trend instead of exact numbers.

Table 2: Tabulation of total number of evaluations, time to solution (in minutes), and accuracy (in terms of best validation R^2) scaling for different search strategies on varying numbers of compute nodes of Theta.

	Node utilization			**Number of evaluations**			**Time to solution** (min)			**Best-model accuracy**		
No. of nodes	**AE**	**RL**	**RS**	**AE**	**RL**	**RS**	**AE**	**RL**	**RS**	**AE**	**RL**	**RS**
33	0.905	0.592	0.913	2093	1066	1780	166	152	134	0.959	0.954	0.942
64	0.920	0.482	0.927	4201	2100	3630	113	165	93	0.960	0.957	0.943
128	0.918	0.527	0.921	8068	4740	7267	148	159	51	0.961	0.960	0.944
256	0.911	0.509	0.936	18039	9680	15221	51	178	63	0.961	0.962	0.945
512	0.962	0.541	0.869	33748	16335	26559	77	179	72	0.961	0.959	0.945



(a) Averaged reward (b) Node utilization

Figure 10: Mean and two standard-deviation-based confidence intervals for 10 NAS evaluations for the evolutionary search process on 128 nodes. The low variability indicates that the optimal performance of this search algorithm was not fortuitous.

4.5 Variability analysis

Our previous set of experiments shows that AE strikes the right balance between optimal node utilization and reward-driven search by avoiding the synchronization requirements of RL. To ensure that the behavior of AE is repeatable, we ran ten runs of AE, each with different random seeds, on 128 nodes. The results are shown in Figure 10. The averaged reward and node utilization curves are shown by their mean value and by shading of two standard deviations obtained from 10 runs. Overall, the values reflect the performance observed in the previous run that used AE on 128 nodes in Figures 3 and 4.

5 Related work

POD-based compression coupled with LSTM-based forecasting has been utilized for multiple physical phenomena that suffer from numerical instability and imprecise knowledge of physics. For instance, the POD-LSTM method has been used for building implicit closure models for high-dimensional systems [19, 30], for turbulent flow control [21], and for exogenous corrections to dynamical systems [2]. Our work is the first deployment of the POD-LSTM method for real-world geophysical data forecasting. There have been recent results from the use of convolutional LSTMs (i.e., using a nonlinear generalization of POD) for forecasting on computational fluid dynamics data [44, 20]. However, we have performed this study using POD because of the spectrally relevant information associated with each coefficient of the forecast. More importantly, previous POD-LSTMs were manually designed, whereas we have demonstrated NAS for automated POD-LSTM development, a significant improvement in the state of the art.

Several studies of automated recurrent network architecture search have been conducted for a diverse set of applications. One of the earliest examples that is similar to our approach was GNARL [3], which utilized evolutionary programming to search for the connections and weights of a neural network with applications to natural language processing. The approach was deployed serially, however, and relied on a synchronous update strategy where 50% of the top-performing architectures are retained while the rest are discarded at each cycle. More recently, there has been research into the use of differentiable architecture search [17], where a continuous relaxation of the discrete search space was used to instantiate a hyper-neural network including all possible models. Based on that a bi-level stochastic gradient based optimization problem was formulated. Nevertheless, this method does not show stable behavior when using skip connections because of unbalanced optimization between different possible operations at variable nodes. Evolutionary NAS has also been utilized to discover the internal configurations of an LSTM cell itself; for instance, in [32] a tree-based encoding of the LSTM cell components was explored for natural language and music modeling tasks. Other studies have looked into optimizing parameters that control the number of loops between layers of a network during the stochastic-gradient-based learning process [36] for image classification applications. This approach leads to recurrent neural architecture discovery during the learning process itself. In contrast, in our study, architectures are generated and evaluated separately at scale.

Evolutionary-algorithm-based architecture searches have also been deployed at scale, for example in [24], where hybridizations of LSTM cells as well as simpler nodes were studied by using neuroevolution [40]. Notably, this work was deployed at scale and also dealt with forecasting of real-world data obtained from the aviation and power sectors. Our approach is also able to integrate hybridizations of fully connected, skip, and identity layers in its search space at scale. A recent investigation in [13] also poses the recurrent NAS problem as the determination of weighting parameters for different networks that act as a mixture of experts applied to language modeling tasks. Our work differs from a majority of the investigations described here by combining the use of scale for architecture discovery for forecasting on a real-world data set for geophysical forecasting tasks.

6 Conclusion and future work

We introduced a scalable neural architecture for the automated development of proper-orthogonal-decomposition-based long-short-term memory networks (POD-LSTM) to forecast the NOAA Optimum Interpolation Sea-Surface Temperature data set. We implemented aging evolution (AE), an asynchronous evolutionary algorithm within DeepHyper, an open-source automated machine learning package, to significantly improve its scalability on Theta, a leadership-class HPC system at Argonne's Leadership Computing Facility. We compared AE to the two search methods already within DeepHyper, a distributed reinforcement learning and random search method, and showed that AE outperforms the distributed reinforcement learning method with respect to node utilization and consequently the time to find high performing architectures. In addition, AE achieves architectures with better accuracy in shorter wall clock time and matches the the node utilization and scalability of the completely asynchronous random search. We compared the best architecture obtained from AE that was retrained for a longer number of epochs to manually designed POD-LSTM variants and linear and nonlinear forecasting methods. We showed that the automatically designed architecture outperformed all of the baselines with respect to the accuracy on the test data, obtaining a R^2 value of 0.876. The automated data-driven POD-LSTM method that we developed has the potential to provide fast emulation of geophysical phenomena and can be leveraged within ensemble forecast problems as well as for real-time data assimilation tasks.

Keeping geophysical emulators as our focus, our future work will seek to overcome the limitations of the POD by hybridizing compression and time evolution in one end-to-end architecture search. In addition, we aim to deploy such emulation discovery strategies for larger and more finely resolved data sets for precipitation and temperature forecasting on shorter spatial and temporal scales. The results from this current investigation also prove promising for continued investigation into AutoML-enhanced data-driven surrogate discovery for scientific machine learning.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research was funded in part and used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. RM acknowledges support from the Margaret Butler Fellowship at the Argonne Leadership Computing Facility. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. DOE or the United States Government. Declaration of Interests - None.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Shady E Ahmed, Omer San, Adil Rasheed, and Traian Iliescu. A long short-term memory embedding for hybrid uplifted reduced order models. *arXiv preprint arXiv:1912.06756*, 2019.

[3] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1994.

[4] Prasanna Balaprakash, Romain Egele, Misha Salim, Stefan Wild, Venkatram Vishwanath, Fangfang Xia, Tom Brettin, and Rick Stevens. Scalable reinforcement-learning-based neural architecture search for cancer deep learning research. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–33, 2019.

[5] Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25(1):539–575, 1993.

[6] Noah D Brenowitz and Christopher S Bretherton. Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, 45(12):6289–6298, 2018.

[7] Jared L Callaham, Kazuki Maeda, and Steven L Brunton. Robust flow reconstruction from limited measurements via sparse representation. *Physical Review Fluids*, 4(10):103907, 2019.

[8] Ashesh Chattopadhyay, Pedram Hassanzadeh, Krishna Palem, and Devika Subramanian. Data-driven prediction of a multi-scale lorenz 96 chaotic system using a hierarchy of deep learning methods: Reservoir computing, ANN, and RNN-LSTM. *arXiv preprint arXiv:1906.08829*, 2019.

[9] Ashesh Chattopadhyay, Pedram Hassanzadeh, and Saba Pasha. Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data. *Scientific Reports*, 10(1):1–13, 2020.

[10] Ashesh Chattopadhyay, Ebrahim Nabizadeh, and Pedram Hassanzadeh. Analog forecasting of extreme-causing weather patterns using deep learning. *arXiv preprint arXiv:1907.11617*, 2019.

[11] Pierre Gentine, Mike Pritchard, Stephan Rasp, Gael Reinaudi, and Galen Yacalis. Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, 45(11):5742–5751, 2018.

[12] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[13] Zhiheng Huang and Bing Xiang. WeNet: Weighted networks for recurrent network architecture search. *arXiv preprint arXiv:1904.03819*, 2019.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] DD Kosambi. Statistics in function space. In *DD Kosambi*, pages 115–123. Springer, 2016.

[16] J Nathan Kutz, Xing Fu, and Steven L Brunton. Multiresolution dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 15(2):713–735, 2016.

[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[18] Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *arXiv preprint arXiv:2002.00470*, 2020.

[19] Romit Maulik, Arvind Mohan, Bethany Lusch, Sandeep Madireddy, Prasanna Balaprakash, and Daniel Livescu. Time-series learning of latent-space dynamics for reduced-order model closure. *Physica D: Nonlinear Phenomena*, 405:132368, 2020.

[20] Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed convolutional LSTM: An efficient deep learning framework to model high fidelity 3d turbulence. *arXiv preprint arXiv:1903.00033*, 2019.

[21] Arvind T Mohan and Datta V Gaitonde. A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks. *arXiv preprint arXiv:1804.09269*, 2018.

[22] Bernd R Noack, Marek Morzynski, and Gilead Tadmor. *Reduced-order modelling for flow control*, volume 528. Springer Science & Business Media, 2011.

[23] Paul A O'Gorman and John G Dwyer. Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *Journal of Advances in Modeling Earth Systems*, 10(10):2548–2563, 2018.

[24] Alexander Ororbia, AbdElRahman ElSaid, and Travis Desell. Investigating recurrent neural network memory structures using neuro-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 446–455, 2019.

[25] Suraj Pawar, SM Rahman, H Vaddireddy, Omer San, Adil Rasheed, and Prakash Vedula. A deep learning enabler for nonintrusive reduced order modeling of fluid flows. *Physics of Fluids*, 31(8):085101, 2019.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Sebastian Peitz, Sina Ober-Blöbaum, and Michael Dellnitz. Multiobjective optimal control methods for the Navier-Stokes equations using reduced order modeling. *Acta Applicandae Mathematicae*, 161(1):171–199, 2019.

[28] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[29] Elizabeth Qian, Boris Kramer, Benjamin Peherstorfer, and Karen Willcox. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406:132401, 2020.

[30] Sk M Rahman, Suraj Pawar, Omer San, Adil Rasheed, and Traian Iliescu. Nonintrusive reduced order modeling framework for quasigeostrophic turbulence. *Physical Review E*, 100(5):053306, 2019.

[31] Stephan Rasp, Michael S Pritchard, and Pierre Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018.

[32] Aditya Rawal and Risto Miikkulainen. From nodes to networks: Evolving recurrent neural networks. *arXiv preprint arXiv:1803.04439*, 2018.

[33] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[34] Eric Rogers, G DiMego, T Black, M Ek, B Ferrier, G Gayno, Z Janjic, Y Lin, M Pyle, V Wong, et al. The NCEP North American mesoscale modeling system: Recent changes and future plans. In *Preprints, 23rd Conference on Weather Analysis and Forecasting/19th Conference on Numerical Weather Prediction*, 2009.

[35] Clarence W Rowley and Scott TM Dawson. Model reduction for flow analysis and control. *Annual Review of Fluid Mechanics*, 49:387–417, 2017.

[36] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

[37] Tapio Schneider, Shiwei Lan, Andrew Stuart, and Joao Teixeira. Earth system modeling 2.0: A blueprint for models that learn from observations and targeted high-resolution simulations. *Geophysical Research Letters*, 44(24):12–396, 2017.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] William C Skamarock, Joseph B Klemp, Jimy Dudhia, David O Gill, Dale M Barker, Wei Wang, and Jordan G Powers. A description of the advanced research WRF version 3. NCAR technical note-475+ STR. 2008.

[40] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[41] Kunihiko Taira, Steven L Brunton, Scott TM Dawson, Clarence W Rowley, Tim Colonius, Beverley J McKeon, Oliver T Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S Ukeiley. Modal analysis of fluid flows: An overview. *AIAA Journal*, pages 4013–4041, 2017.

[42] Qian Wang, Jan S Hesthaven, and Deep Ray. Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem. *Journal of Computational Physics*, 384:289–307, 2019.

[43] Jinyu Xie and Qian Wang. Benchmark machine learning approaches with classical time series approaches on the blood glucose level prediction challenge. In *KHD@ IJCAI*, pages 97–102, 2018.

[44] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[45] Hyelim Yoo, Zhanqing Li, Yu-Tai Hou, Steve Lord, Fuzhong Weng, and Howard W Barker. Diagnosis and testing of low-level cloud parameterizations for the NCEP/GFS model using satellite and ground-based measurements. *Climate dynamics*, 41(5-6):1595–1613, 2013.

[46] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.