Probabilistic Parallel Time Networks

Anonymous submission

Abstract

Time series forecasting remains a complex challenge, especially in intricate domains such as chaotic systems. This paper introduces "Probabilistic Parallel Time Networks", a novel deep neural network hypermodel designed to generalize across diverse domains in complex, probabilistic time series forecasting. The learned multi-modal model takes three distinct inputs: static data, agent history, and forecasts, and subsequently predict probability distributions over time. This innovative approach offers a more interpretable and unified framework for complex forecasting tasks. The utilization of a genetic hyperparameter optimization algorithm intertwined with the Differentiable Architecture Search (DARTS) neural architecture search, and the careful minimization of the neural architecture space, ensures a tractable and efficient solution. Through empirical studies in weather forecasting for microclimates and the chaotic Lorenz system, we demonstrate the model's applicability, robustness, and potential. The Probabilistic Parallel Time Network model offers a promising advancement in time series forecasting, with potential implications for research and practical applications in diverse areas, including healthcare, finance, and environmental sciences.

Introduction

In an era where data-driven decision-making is paramount, forecasting complex, temporal phenomena is crucial across various domains. From healthcare and financial markets to transportation and energy management, accurate forecasting is the cornerstone of strategic planning and risk mitigation. The nonlinearity and complexity of many real-world systems necessitate models that capture intricate temporal dynamics, offer robust generalizability, and interpretability. Furthermore, with the large successes found in meta-learning, we strive to move past the proposals of neural architectures and propose novel model spaces or hypermodels. In this work, we introduce the Probabilistic Parallel Time Networks (PTNs), a novel hypermodel designed to address these challenges.

PTN is a hypermodel yielding a multi-modal neural network with three inputs: static, history, and forecasts. The static input gives context about the agent or state, the history gives temporal context, and the forecasts are given by an arbitrary deterministic forecaster. Being a hypermodel, PTNs generalize across various use cases, as the algorithm learns the network's architecture to the given task.

The appeal of PTNs extends to diverse use cases. We go in depth into the paramount, real-life application of weather forecasting for farms. Many farmers in the western United States operate in unique microclimates that are not accurately represented in public weather forecasts. This discrepancy can lead to significant differences between forecasted and actual weather conditions experienced on the farms. This bias creates hardships for farmers who rely on forecasts for preventative planning.

Additionally, we forecast one of the most difficult deterministic systems, the chaotic Lorenz system. This application serves as a pivotal demonstration of PTN's capabilities. With the Lorenz system, we showcase generalizability to capture complex temporal dynamics, explore chaos theory with approximated probability distributions showing possible trajectories, and highlight interdisciplinary relevance across various scientific disciplines.

Our contributions in this work are as follows:

- We introduce Probabilistic Parallel Time Networks (PTNs) for the purpose of time series forecasting.
- We integrate DARTS (Differentiable Architecture Search) (Liu, Simonyan, and Yang 2019) with genetic-based hyperparameter optimization as a pivotal element of our approach.
- We apply our PTN approach for forecasting in two real-world scenarios: weather forecasting and the Lorenz system analysis.

By addressing these challenges in complex forecasting, we aim to improve various societal aspects, from agriculture to economic preparations.

Related Work

We discuss related works related to PTNs that we have built upon, including meta-learning techniques such as neural architecture search algorithms (NAS) and hyperparameter optimizations (HPO), multi-modal networks, and model stacking.

Over the past several years, meta-learning has shifted the machine learning paradigm (Verma, Brahma, and Rai 2020; White, Neiswanger, and Savani 2021; Bischl et al. 2023). We find that there are broadly eight categories: grid search, random search, evolutionary algorithms, gradient-based meth-

ods, Bayesian optimization, reinforcement learning, one-shot NAS, and hybrid methods.

Grid searches are exhaustive and iterate throughout the entire model space. Bergstram finds that random searches, which randomly and naïvely traverse the model space often outperforms grid searches (Bergstra and Bengio 2012). Evolutionary NAS algorithms test each model in a generation with a fitness function. The model space is then traversed by using crossovers and mutations while considering the fitness score(Liu et al. 2020). Gradient-based methods shift the paradigm from testing discrete model subspaces by relaxing it into a continuous space. DARTS is such an algorithm (Liu, Simonyan, and Yang 2019). Bayesian optimization builds a probabilistic model of the objective function to predict the most promising architectures (Zhou et al. 2019). Next, Reinforcement-learning-based NAS algorithms use a controller with a learned policy to predict model candidates (Zoph and Le 2016). Another approach is the one-shot NAS algorithm which trains a single large network (supernet) that includes all possible architectures as subnetworks. The supernet then estimates each architecture's performance (Bender et al. 2018). Finally, Hybrid methods combine two or more approaches. Our PTN is a hybrid method combining evolutionary (genetic) hyperparameter optimization with a gradient-based NAS.

Multi-modal networks allow the network to reason from different representations. For example, GPT-4 allows image and text inputs which allows for greater flexibility of communication (OpenAI 2023). Stacking models has been shown to improve forecasting results (Pavlyshenko 2018). Our forecasts input is derived from any given forecaster. While other stacking methods only include the most recent forecast, we introduce parallel time by including the sequence of forecasts in a two dimensional framework with channels: (forecast number, time within the forecast, channels).

Regarding weather forecasting, the PTN excels with accommodating a growing weather network across multiple regions, with varying lengths of data ranging from weeks to years. Our static data helps this as the model can infer and specialize predictions for new sites by interpolating the learned static variable influences from other sites. Finally, our history input gives observations of events prior.

In weather forecasting, most deep learning techniques for weather forecasting employ spatiotemporal methods (Yuan et al. 2022; Bojesomo, Al-Marzouqi, and Liatsis 2021; Diehl et al. 2015), these techniques also assume the presence of numerous neighboring stations. Our PTN incorporates this by stacking the National Blend of Model's (NBM) forecasts (Craven, Rudack, and Shafer 2020). However, the challenge comes from the farm terrain being substantially different from most of the forecasted grid.

Data Preparation and Processing

In this section, we discuss our data wrangling techniques.

Weather Data

For weather forecasting, our learned model input includes static information about farms (such as elevation), temporal weather history sampled within the farms, and NBM's forecasts. The static data includes the latitude, longitude, elevation, slope, aspect, and exposure of the farms. Slope is the angle of the terrain, aspect is the orientation or direction the terrain faces, and exposure measures how sheltered an area is.

The farm history includes observations of air temperature, atmospheric pressure, vapor pressure, wind speed, wind direction, wind gusts, solar radiation, and dew point. Each measurement is taken from two meters above the ground every fifteen minutes. Solar radiation is the intensity of the sun upon the given terrain. The dew point is the temperature at which the water vapor will begin to condense. We calculate dew point using MetPy (May et al. 2022).

Our hourly NBM forecasts include temperature, dewpoint, solar radiation, wind direction, wind speed, and total cloud cover for 36 hours. We forecast up to 40 hours out with fifteen minute intervals.

Except for the time features, we normalize all features between zero and one. For time, we properly capture its cyclical measure before scaling. We convert each date into a numerical day of year (DoY) and each time of day into "seconds from midnight" (SFM). Then, we normalize DoY and SFM to $[-2\pi, 2\pi)$ before casting both time features into two pairs of sine and cosine. After the trigonometry conversions, we normalize the time data between zero and one.

Due to privacy concerns and policy compliance, we cannot release the farms' static or temporal data. However, the NBM forecasts are publicly available (Craven, Rudack, and Shafer 2020). We do not know of any publicly available datasets that sample from farm microclimates to the degree of this dataset.

The Chaotic Lorenz System

Chaotic systems are deterministic dynamical systems that exhibit highly sensitive dependence on initial conditions. The sensitivity is severe enough that it appears random and unpredictable—a minute change from an input of (10,10,10) to (10.00000000001,10,10) yields a substantial change over time in the output. This is known as the "butterfly effect". Additionally, the systems have "topological mixing" where points in the system's phase space eventually become close to each other. Furthermore, chaotic systems have dense periodic orbits, where every point in the space is approached arbitrarily close by periodic orbits. We focus on the Lorenz system (Lorenz 1963).

The Lorenz system is a simplified mathematical model for atmospheric convections. This chaotic system's phase space is a 3D Cartesian coordinate system with three dependent variables: (x, y, z) and three parameters (σ, ρ, β). Formally, we name it $\mathcal{L}(x, y, z; \sigma, \rho, \beta)$ or $\mathcal{L}$ for short. $\mathcal{L}$ is defined by an ordinary differential equation for each of its three axes:

$$\frac{dx}{dt} = \sigma(y - x), \tag{1}$$

$$\frac{dy}{dt} = x(\rho - z) - y, \tag{2}$$

$$\frac{dz}{dt} = xy - \beta z. \tag{3}$$

For our experiments, we sample our initial dependent variables from uniform distributions:

$$(x_0, y_0, z_0) \sim \big(U(-20, 20), U(-20, 20), U(0, 40)\big) \quad (4)$$

$\mathcal{L}$ is parameterized by σ, ρ, and β. The typical values are references from the original work (Lorenz 1963).

- σ: The Prandtl number represents the ratio of momentum diffusivity (how momentum is transferred through a fluid due to its viscosity) to thermal diffusivity (how quickly heat is transferred through a material). Typically, $\sigma = 10$ and straying too far from 10 may have uninteresting dynamics. Our experiments sample σ from a uniform distribution bounded by 8 and 12: $\sigma \sim U(8, 12)$.
- ρ: The Rayleigh number represents the temperature difference between the top and bottom of the fluid layer. Typically, $\rho = 28$ and increasing ρ often leads to more chaos. For our experiments, $\rho \sim U(27.9, 28.1)$.
- β: A geometry factor that is related to the aspect ratio of the physical system, specifically the ratio of the height to the width of the convective rolls. Large deviations from $8/3$ might lead to unphysical behaviors. Typically, $\beta = 8/3$. We use $\beta \sim U(7.9/3, 8.1/3)$ in our experiments.

$\mathcal{L}$ partly achieved such notoriety due to its strange attractor as visualized in Figure 1. This is a subset of the phase space with a fractal structure where trajectories from a wide region are attracted to. Once trajectories enter the attractor, they stay there, wandering chaotically with no repetition. Due to $\mathcal{L}'s$ significant numerical instability, machine learning is an attractive method to predict several steps in the future when exact parameters or coordinates are unknown. Furthermore, an approximate probability density function is further appealing since we can see possible paths and their likelihoods. This is as opposed to the deterministic forecast, which may take the center of two trajectories in times of high uncertainty even if the center is impossible. For example, if a bit is randomly assigned 0 or 1 with a uniform distribution, its optimal for a deterministic forecast to predict 0.5. However, a distribution can correctly predict 0 or 1, both with 50% probability.

In our Lorenz system case study, our three inputs follow: the static data is the parameters (σ, ρ, β), the temporal history are the coordinates (x, y, z) over sixteen time steps, and ten iterations of forecasts with sixteen time steps each are derived from our trained deterministic forecaster, a multilayer perceptron based on the nonlinear vector autoregressive model (NVAR). We choose this structure since (Shahi, Fenton, and Cherry 2022) found the NVAR model to outperform other machine learning techniques. We then forecast the x coordinate's approximate probability density function for sixteen time steps using 111 quantiles. All of our inputs are normalized between zero and one.

We use several advanced and scientific Python libraries for our models, experiments, and analyses. We utilize SciPy (Virtanen et al. 2020) to compute the differential equations of $\mathcal{L}$ and PyTorch (Paszke et al. 2019) for neural networks. For computational, qualitative, and quantitative analyses, we also use Pandas (pandas development team 2020), NumPy (Harris et al. 2020), seaborn (Waskom 2021), and Matplotlib

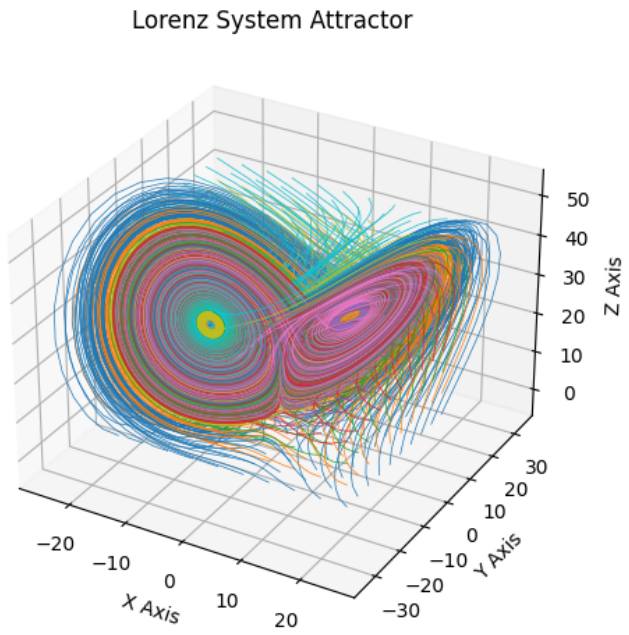

Figure 1: The Lorenz System's strange attractor. Every trajectory is chaotic but once they are near enough to the attractor space, they never leave nor visit the same position again.

(Hunter 2007). For computing dewpoint, we use MetPy (May et al. 2022). Finally, we also utilize scikit-learn (Pedregosa et al. 2011).

Probabilistic Parallel Time Networks

PTN is a hypermodel and is generalizable to any time series applications such as weather, video game actions, finances, and epidemics. The PTN hypermodel takes three inputs: agent history, agent static data, and a series of forecasts (the parallel time aspect). Then, the PTN outputs a multistep probabilistic forecast by using n quantiles. Notably, we are able to produce probabilistic forecasts given deterministic targets. To illustrate the effectiveness and robustness of our framework, we present two use cases: weather forecasting and the chaotic Lorenz system. We begin by describing the model space.

Model Space

To make the combination of genetic hyperparameter optimization and DARTS tractable, we restrict our model space. The DARTS algorithm contains two optimizers: one for the architecture and the other for the other parameters. The architecture parameters are the weights assigned to each neural layer and pass through a softmax function to learn the layer's importance for later pruning. The other parameters are trained with one data set and the architecture parameters are trained a separate data set. The architecture parameters are trained for five steps in each epoch with batch sizes of 64. The other parameters are trained for ten steps per epoch. We test each model candidate for 30 epochs each throughout our hyperparameter and architecture search.

We use a separate AdamW optimizer (Loshchilov and Hutter 2017) for both sets of parameters. Both optimizers have their own "reduce on plateau" learning rate scheduler with a patience of one. A patience of p reduces the learning rate by ten times if the loss is greater than the smallest seen loss for $p + 1$ consecutive epochs.

Each hypermodel subspace mentioned is followed by a set of architecture parameters with one parameter for each layer that signal the given layer's importance. This is part of the DARTS algorithm (Liu, Simonyan, and Yang 2019).

Architecture Space We embed each of our three inputs (static, history, and forecasts) into a $d \times d$ space. Static data is a vector of c_s real valued features. History is a matrix with a temporal τ_h and channel c_h dimensions. Finally, forecasts is represented by a $\mathbb{R}^{f_f \times \tau_f \times c_f}$ tensor where f_f is the number of the most recent forecasts, τ_f is the number of time steps predicted by each forecast, and c_f is the quantity of channels forecasted. Recall that our forecast input is given by an arbitrary, deterministic forecaster. The hyperparameters are drawn from the hyperparameter space in Section . For example, the Leaky ReLU's (LReLU's) negative slope is learned from the range $[0.0, 0.3]$. All of our 1D-CNN layers use causal padding.

The static data's embedding block begins with Gaussian noise regularization, a fully connected layer transforming the dimensions $c_s \mapsto d$, batch normalization over the channel dimension, and a LReLU activation. Next, the d dimensions are repeated along the temporal dimension ($1 \times d \mapsto d \times d$), followed by the first dropout layer, a dense layer, and the LReLU activation. We choose to implement Gaussian noise in the static data to allow generalizability for new systems and mitigate memorization.

The history embedding block is a dense layer ($\tau_h \times c_h \mapsto \tau_h \times d$) followed by an LReLU activation and a dropout layer. Next, the temporal dimension is changed by using a 1D-CNN layer over the temporal dimension ($\tau_h \times d \mapsto d \times d$) followed by the LReLU activation. Next, the data travels through three layers with learned architecture parameters: multihead attention, 1D-CNN, and a bi-directional GRU RNN. These three layers are part of the hypermodel subspace and pruned if the parameters find them unsubstantial.

The forecasts embedding block begins with a hypermodel subspace with two possible layers: a 2D-CNN (named Conv2D Collapse) and a dense layer. Both of which transform the data from $f_f \times \tau_f \times c_f$ to $f_f \times \tau_f$. Next, the forecasts are cast to $d \times d$ space in the same manner as the history, by a dense layer, then an LReLU activation, which is followed by a 1D-CNN over the temporal dimension. Following the forecasts embedding, we use another hypermodel subspace of three layers: a multihead attention layer, a 1D-CNN, and a bi-directional GRU RNN.

To help with interpretibility, we also multiply all three embedding spaces with a unique learnable parameter before their fusion by summation. Although, we do not prune any of them.

After fusion, we apply the learned normalization layer, use a 1D-CNN over the temporal dimension changing the space from $d \times d$ to $\tau_t \times d$, implement layer normalization and an LReLU activation, implement a 1D-CNN over the channel dimensions with an LReLU ($\tau_t \times c_t$), then our output layer is a dense layer with no activation. This output is now c_t quantiles over τ_t timesteps. The unbounded output allows the quantiles to go beyond the range of the trained data.

Hyperparameter Space Our hyperparameters have two types: choice and range. If choices are unordered, the crossover elicits one of the parents' current choices with probability proportional to the parents' fitnesses. Otherwise, the child receives the interpolated choice, weighted by the parents' fitnesses. Similar to ordered choices, the range types elicit weighted interpolated values in their crossovers. If the range is logarithmic, the interpolation happens in the natural log space.

In Table 1, we provide PTN's hyperparameter space. The hidden size d is sampled from the set of all integers between 72 and 255 that are divisible by the least common multiple of the number of MHA choices (24). This is due to the restrictions of the multihead attention block. We omit the attention head count of one since this value often performs worse (Vaswani et al. 2017). Additionally, we cap the hidden size to 255 since we find an increase to 256 doubles the training time in our environment and often gives less favorable results. The first layer dropout rate is learned separately from the dropout rate in other layers in case too much information would be lost on the first dropout. The kernel sizes are odd to prevent the copying overhead of even kernel sizes as implemented in PyTorch.

Genetic Hyperparameter Optimization with DARTS

We simultaneously traverse the neural architecture and hyperparameter search spaces by implementing DARTS within the genetic HPO.

Our evolutionary algorithm is explained in Algorithm 1. This algorithm implements state-of-the-art techniques such as proportional crossovers, a dynamic mutation rate, elite rounds, the DARTS NAS algorithm, and a patience threshold for the population's evolution. We then select the x best agents by the smallest sum of squares for the fitness and fitness validation scores. Our data is partitioned five ways: neural training and neural validation for the DARTS training, genetic training and genetic validation for measuring the fitness and fitness validation scores, and a test data set.

We define variable names for Algorithm 1 below.

- (f, f_v): the fitness and validation fitness scores.
- e_f: the frequency of elitism rounds.
- n_e: the number of elite agents injected into the population.
- $sort(S; x)$: sort S by x in descending order.
- p: the patience generation count.
- n: the number of agents within a living population.

Loss and Fitness Functions

To minimize the loss when predicting the approximate probability distribution from deterministic target data, we minimize the approximate continuous ranked probability score

Name	Values	Type	Is Logarithmic	Is Ordered
Hidden Size d	$[72 + 24n \mid n = 0, 2, \ldots, 7]$	Choice	False	True
Dropout Rate	$[0.0, 0.5]$	Range	False	N/A
First Layer Dropout Rate	$[0.0, 0.5]$	Range	False	N/A
Leaky ReLU Negative Slope	$[0.0, 0.3]$	Range	False	N/A
History MHA Heads	$[2, 3, 4, 6, 8, 12]$	Choice	False	True
Forecasts MHA Heads	$[2, 3, 4, 6, 8, 12]$	Choice	False	True
Gaussian Noise's Standard Deviation	$[0.0, 0.25]$	Range	False	N/A
Architecture Parameters' AdamW's LR	$[0.00001, 0.001]$	Range	True	N/A
Architecture Parameters' AdamW's β_1	$[0.5, 0.99]$	Range	False	N/A
Architecture Parameters' AdamW's β_2	$[0.95, 0.999]$	Range	False	N/A
Other Parameters' AdamW's LR	$[0.00001, 0.001]$	Range	True	N/A
Other Parameters' AdamW's β_1	$[0.5, 0.99]$	Range	False	N/A
Other Parameters' AdamW's β_2	$[0.95, 0.999]$	Range	False	N/A
Normalization Layer	[layer norm, batch norm, none]	Choice	N/A	False
History Conv1D Embed Kernel Size	$[1, 3, 5, \ldots, 15]$	Choice	False	True
History Conv1D Kernel Size	$[1, 3, 5, \ldots, 15]$	Choice	False	True
Forecasts Conv2D Collapse Kernel Width	$[1, 3, 5, \ldots, 15]$	Choice	False	True
Forecasts Conv2D Collapse Kernel Height	$[1, 3, 5, \ldots, 15]$	Choice	False	True
Forecasts Conv1D Embed Kernel Size	$[1, 3, 5, 7, 9]$	Choice	False	True
Forecasts Conv1D Kernel Size	$[1, 3, 5, \ldots, 15]$	Choice	False	True
Output Kernel Temporal Size	$[1, 3, 5, \ldots, 15]$	Choice	False	True
Output Kernel Features Size	$[1, 3, 5, \ldots, 15]$	Choice	False	True

Table 1: The hyperparameter space for PTN.

(CRPS) using an estimated cumulative density function during run time (see Equation 7). Our approximation uses summations as opposed to the integrals used for probability density functions since our predicted quantiles are discrete. The fitness functions and evaluation metrics use CRPS as described in Equation 7 while our neural network optimizers use CRPS plus the ordering penalty as shown in 9.

Our notations follow:

- $\hat{y} \in \tau_t \times c_t$: the predicted quantiles over time.
- $y \in \tau_t \times 1$: the deterministic target over time.
- $t \in [1, 2, \ldots, \tau_t]$: the time step.
- $T_1 \in \tau_t \times 1$: the MAE for each quantile by subtracting each corresponding deterministic target in time, elementwise. We formalize T_1 in Equation 5.
- $T_2 \in \tau_t \times 1$: measures the spread of $\hat{y}$ by calculating the distance between each pair and giving a summarized scalar. We formalize T_2 in Equation 6.

$$T_1 = \frac{1}{\tau_t} \sum_{i=1}^{\tau_t} \left(\frac{1}{c_t} \sum_{j=1}^{c_t} |\hat{y}_{i,j} - y_i| \right) \tag{5}$$

$$T_2 = \frac{1}{\tau_t} \sum_{i=1}^{\tau_t} \left(\frac{1}{c_t^2} \sum_{j=1}^{c_t} \sum_{k=1}^{c_t} |\hat{y}_{i,j} - \hat{y}_{i,k}| \right) \tag{6}$$

$$\text{CRPS} = T_1 - 0.5T_2 \tag{7}$$

Following the results from (Takeuchi et al. 2006), we add an ordering penalty $\mathcal{O}$ to the quantiles. Recall our predictions are in the space $\tau_t \times c_t$ for c_t quantiles over τ_t time steps. Our ordering penalty loss occurs iff the values are not monotonically non-decreasing. We formalize this penalty in Equation 8.

$$\mathcal{O} = \frac{1}{\tau_t \cdot (c_t - 1)} \sum_{i=1}^{\tau_t} \sum_{j=1}^{c_t - 1} \max(0, \hat{y}_{i,j} - \hat{y}_{i,j+1}) \tag{8}$$

Combining the ordering penalty with weight ω, we describe our loss function, ℓ in Equation 9.

$$\ell = \text{CRPS} + \omega\mathcal{O} \tag{9}$$

We use $\omega = 0.0075$.

Experiments and Results

Here, we review our experiments and results. We show that, overall, PTN outperforms the deterministic models and is more informative. We run our experiments on a local desktop with an RTX 4090 GPU, i9-13900K CPU, 96 GB of DDR5 RAM, and the Windows 11 operating system.

For weather forecasting, we predict 111 quantiles over 40 hours with 15 minute resolution (160 time steps). In Figure 2, we see that the target data is entirely within the bounds of our quantiles. Furthermore, the median is often closer to the target than NBM.

To calculate skill scores (as defined in Equation 10), we transform the NBM forecasts to a Gaussian distribution by finding NBM's mean absolute error (MAE) from the target in the training dataset. NBM's predicted value is then the

Algorithm 1: Probabilistic Parallel Time Network

Randomly initialize the current population, A_c.
Track all individuals with the set A.
for $generation = 1$ to $num\_generations$ **do**
  for all $a \in A_c$ **do**
    Encode the neural model.
    Train the model and implement NAS by DARTS.
    Update the fitness and validation fitness of a.
  end for
  $A \leftarrow A \cup A_c$
  if The minimum $f_v \in A$ has not improved for p generations **then**
    break
  end if
  if $generation$ mod $e_f = 0$ **then**
    $A_c \leftarrow sort(A_c; f_v)[1, 2, \ldots, n - n_e]$.
    $A_c \leftarrow A_c \cup sort(A \setminus A_c; f_v)[1, 2, \ldots, n_e]$.
  end if
  Select n pairs of parents by the tournament selection strategy, with replacement and using the validation fitness.
  $A_c \leftarrow \emptyset$
  for all Parent pairs **do**
    Use proportional crossover to create a child c.
    Compute the expected fitness as the average of both parent's f_v.
    Mutate c with probability in range $[0.05, 0.6]$, proportional to the expected fitness.
    $A_c \leftarrow A_c \cup c$
  end for
end for
Best x agents $\leftarrow sort(A; f_v^2 + f^2)[1, 2, \ldots, x]$.

mean and the MAE is the standard deviation. We then extract the corresponding 111 quantiles from the approximated distribution. To compute a skill score for MSE, we compare PTN's median quantiles with NBM's predicted values.

$$\text{Skill score} = 1 - \frac{\text{performance of model}}{\text{performance of baseline}} \in (-\infty, 1] \tag{10}$$

We present the skill scores in Table 2. We give depth to these results by calculating the skill score for every time window instance, then we give the quartile scores. Furthermore, we find the percentage of forecasts improved upon by locating the percentile of a zero score (where the forecasts are equally accurate) then subtract it from one. For example, a zero score located at the percentile of 6 means that 6% of the scores are below zero so 94% are greater than or equal to zero. In both skill scores, PTN dominates the NBM forecast. A skill score of 1 is a perfect forecast for the PTN, a score of zero is when the PTN performs equal to the baseline, and a negative score means the PTN performed worse than the NBM baseline. For the chaos system, our PTN performed similarly in Table 3.

For weather forecasting, PTN takes about 8.5 hours. For the chaotic system, the PTN takes about 6.5 hours. This discrepancy is likely due to the large amount of data (about 60 GB) that we use in the weather forecasting opposed to the couple of gigabytes in the Lorenz system. Training the PTN's learned neural network takes between thirty and sixty seconds. For our analyses, we train the network on the data used in the DARTS training and validation data, then implement a warm start by training on a fraction of the left out data before testing on the rest of the left out data. For the weather data, we partition data by sites. For $\mathcal{L}$, we generate 50 systems with randomly initialized parameters and 125 different, random initial conditions for each data partition. For each system and instance, $\mathcal{L}$ has 1000 evenly spaced times between 0 and 100 at which the solution to the initial value problem is evaluated.

Figure 3 illustrates a confident forecast (one with little spread) and Figure 4 illustrates a trimodal probability distribution between time steps two and six. We see that our deterministic forecast is consistent with one of three clusters of quantiles, but is substantially off from the target data. Our median is in the center cluster, and our third cluster contains the target data. This illustrates the importance of probabilistic forecasts in understandability and risk assessment.

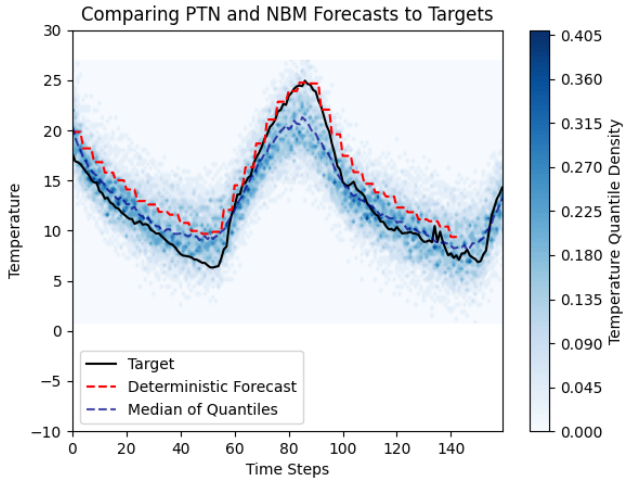

Figure 2: This plot compares the PTN predicted probability distributions for temperature over time to the NBM forecast and target data.

Metric	CRPS Skill Score	MSE Skill Score
75th Percentile	0.70	0.91
Median	0.59	0.83
25th Percentile	0.42	0.65
Percentage of forecasts improved	96.4%	95.8%

Table 2: Summary statistics for the PTN CRPS and MSE skill scores against NBM. A score above zero means PTN improves over NBM; a score of one is a perfect forecast.

Metric	CRPS Skill Score	MSE Skill Score
75th Percentile	0.68	0.89
Median	0.57	0.79
25th Percentile	0.40	0.59
Percentage of forecasts improved	94.3%	90.8%

Table 3: Summary statistics for the CRPS and MSE Skill scores for PTN compared to the deterministic forecaster. A score above zero means PTN improves over the deterministic forecaster; a score of one is a perfect forecast.

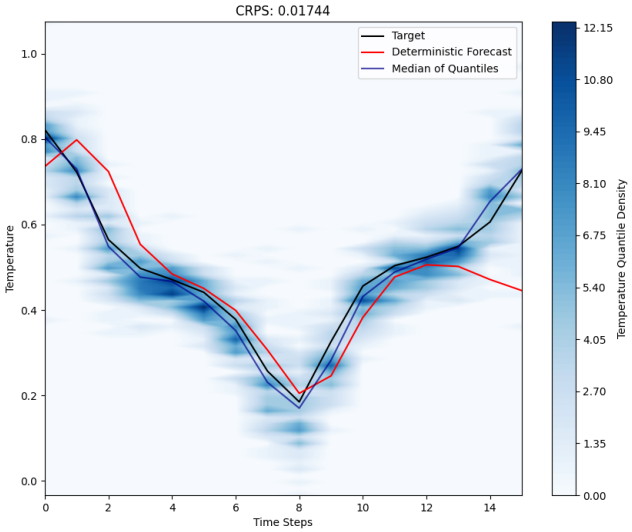

Figure 3: A confident prediction by PTN for the chaotic Lorenz system. A confident prediction has less spread in its quantiles.

Should we have a deterministic forecast that takes the middle, we may falsely that assume the distribution is nearly Gaussian and be surprised at the frequency of large errors in certain states. Note that Figure 4 is not representative of the deterministic forecast; its used as an example for our former statement.

Conclusions

In conclusion, we successfully present Probabilistic Parallel Time Networks (PTNs), a hypermodel that approximates a probability distribution for each of multiple time steps. By searching an architecture and a hyperparameter space, PTNs are designed to be effective, transferrable, and robust. PTNs are designed to handle multi-modal input for additional context. The inputs are static data, historical temporal data, and a series of forecasts from a deterministic forecasting model (the parallel time aspect).

We illustrate how the PTN's probabilistic forecasts are more interpretable and informative than deterministic forecasts in our case studies of weather forecasting in microclimates and difficult systems such as the chaotic Lorenz system. In real life applications such as economic preparations and weather forecasting, this understanding model uncertainty is critical for the quality of human life. Additionally, these use cases demonstrate PTN's effectiveness and robustness.

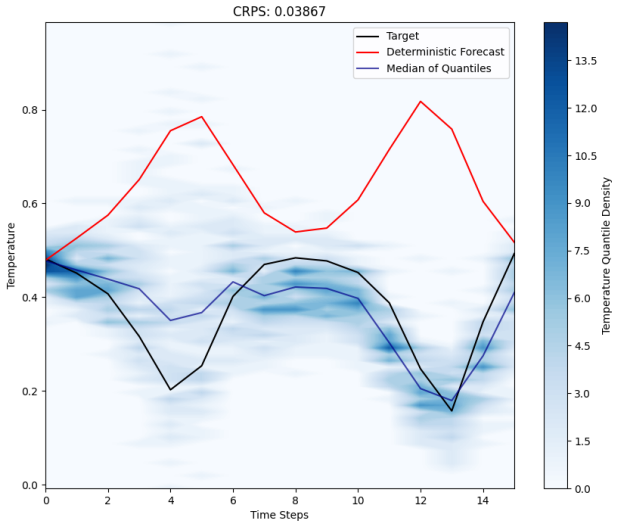

Figure 4: Another forecast for the Lorenz system. We can see that the model is less confident in this forecast than in Figure 3 but predicts that the chaos will take one of three routes between time steps two and six.

Future research may explore the application of PTNs to other complex systems such as forecasting a double pendulum forecasting, electricity demand forecasting, epidemic predictions, full general relativity, and other environmental sciences. Furthermore, forecasts that often go to one extreme or another greatly benefit from probabilistic forecasts, such as volatile financial markets, green energy reliant on wind speed, flooding in certain terrains, and election forecasting in polarized political climates. Additionally, other combinations of NAS and hyperparameter optimizations may be explored to optimize the hypermodel exploration.

This work contributes to the ongoing advancement of forecasting methodologies, offering a novel approach of a hypermodel that learns multimodal models tailored to the given application. The case study of the Lorenz system especially shows promising applications for the practical applicability of complex systems.

References

Bender, G. M.; jan Kindermans, P.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and Simplifying One-Shot Architecture Search.

Bergstra, J.; and Bengio, Y. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

Bischl, B.; Binder, M.; Lang, M.; Pielok, T.; Richter, J.; Coors, S.; Thomas, J.; Ullmann, T.; Becker, M.; Boulesteix, A.-L.; et al. 2023. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2): e1484.

Bojesomo, A.; Al-Marzouqi, H.; and Liatsis, P. 2021. Spatiotemporal Vision Transformer for Short Time Weather Forecasting. In *2021 IEEE International Conference on Big Data (Big Data)*, 5741–5746.

Craven, J. P.; Rudack, D. E.; and Shafer, P. E. 2020. National Blend of Models: a statistically post-processed multi-model ensemble. *Journal of Operational Meteorology*, 8(1).

Diehl, A.; Pelorosso, L.; Delrieux, C.; Saulo, C.; Ruiz, J.; Gröller, M. E.; and Bruckner, S. 2015. Visual analysis of spatio-temporal data: Applications in weather forecasting. In *Computer Graphics Forum*, volume 34, 381–390. Wiley Online Library.

Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; and Oliphant, T. E. 2020. Array programming with NumPy. *Nature*, 585(7825): 357–362.

Hunter, J. D. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3): 90–95.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. arXiv:1806.09055.

Liu, Y.; Sun, Y.; Xue, B.; Zhang, M.; and Yen, G. G. 2020. A Survey on Evolutionary Neural Architecture Search. *CoRR*, abs/2008.10937.

Lorenz, E. N. 1963. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2): 130–141.

Loshchilov, I.; and Hutter, F. 2017. Fixing Weight Decay Regularization in Adam. *CoRR*, abs/1711.05101.

May, R. M.; Goebbert, K. H.; Thielen, J. E.; Leeman, J. R.; Camron, M. D.; Bruick, Z.; Bruning, E. C.; Manser, R. P.; Arms, S. C.; and Marsh, P. T. 2022. MetPy: A Meteorological Python Library for Data Analysis and Visualization. *Bulletin of the American Meteorological Society*, 103(10): E2273 – E2284.

OpenAI. 2023. GPT-4: Generative Pre-trained Transformer 4. Technical report, OpenAI.

pandas development team, T. 2020. pandas-dev/pandas: Pandas.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Pavlyshenko, B. 2018. Using Stacking Approaches for Machine Learning Models. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, 255–258.

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

Shahi, S.; Fenton, F. H.; and Cherry, E. M. 2022. Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study. *Machine Learning with Applications*, 8: 100300.

Takeuchi, I.; Le, Q.; Sears, T.; and Smola, A. 2006. Nonparametric Quantile Estimation. *Journal of Machine Learning Research*, 7: 1231–1264.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need.

Verma, V. K.; Brahma, D.; and Rai, P. 2020. Meta-learning for generalized zero-shot learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 6062–6069.

Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272.

Waskom, M. L. 2021. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60): 3021.

White, C.; Neiswanger, W.; and Savani, Y. 2021. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 10293–10301.

Yuan, T.; Zhu, J.; Ren, K.; Wang, W.; Wang, X.; and Li, X. 2022. Neural Network Driven by Space-time Partial Differential Equation for Predicting Sea Surface Temperature. In *2022 IEEE International Conference on Data Mining (ICDM)*, 656–665.

Zhou, H.; Yang, M.; Wang, J.; and Pan, W. 2019. Bayesnas: A bayesian approach for neural architecture search. In *International conference on machine learning*, 7603–7613. PMLR.

Zoph, B.; and Le, Q. V. 2016. Neural Architecture Search with Reinforcement Learning. *CoRR*, abs/1611.01578.