DJEnsemble: a Cost-Based Selection and Allocation of a Disjoint Ensemble of Spatio-temporal Models

Rafael S. Pereira
LNCC-DEXL
Petropolis, Brazil
rpereira@lncc.br

Yania Molina Souto
LNCC-DEXL
Petropolis, Brazil
yaniams@lncc.br

Anderson Silva
LNCC-DEXL
Petropolis, Brazil
achaves@lncc.br

Rocio Zorrilla
LNCC-DEXL
Petropolis, Brazil
romizc@lncc.br

Brian Tsan
UC Merced
Merced, USA
btsan@ucmerced.edu

Florin Rusu
UC Merced
Merced, USA
frusu@ucmerced.edu

Eduardo Ogasawara
Cefet-RJ
Rio de Janeiro, Brazil
eogasawara@ieee.org

Arthur Ziviani
LNCC-DEXL
Petropolis, Brazil
ziviani@lncc.br

Fabio Porto
LNCC-DEXL
Petropolis, Brazil
fporto@lncc.br

ABSTRACT

Consider a set of black-box models – each of them independently trained on a different dataset – answering the same predictive spatio-temporal query. Being built in isolation, each model traverses its own life-cycle until it is deployed to production, learning data patterns from different datasets and facing independent hyper-parameter tuning. In order to answer the query, the set of black-box predictors has to be ensembled and allocated to the spatio-temporal query region. However, computing an optimal ensemble is a complex task that involves selecting the appropriate models and defining an effective allocation strategy that maps the models to the query region. In this paper we present DJEnsemble, a cost-based strategy for the automatic selection and allocation of a disjoint ensemble of black-box predictors to answer predictive spatio-temporal queries. We conduct a set of extensive experiments that evaluate DJEnsemble and highlight its efficiency, selecting model ensembles that are almost as efficient as the optimal solution. When compared against the traditional ensemble approach, DJEnsemble achieves up to 4X improvement in execution time and almost 9X improvement in prediction accuracy.

CCS CONCEPTS

• **Information systems → Query planning**; **Spatial-temporal systems**.

KEYWORDS

Ensembles, Spatio-temporal, Deep Learning, Query Processing

SSDBM 2021, July 6–7, 2021, Tampa, FL, USA

ACM ISBN 978-1-4503-8413-1/21/07…$15.00
https://doi.org/10.1145/3468791.3468806

ACM Reference Format:
Rafael S. Pereira, Yania Molina Souto, Anderson Silva, Rocio Zorrilla, Brian Tsan, Florin Rusu, Eduardo Ogasawara, Arthur Ziviani, and Fabio Porto. 2021. DJEnsemble: a Cost-Based Selection and Allocation of a Disjoint Ensemble of Spatio-temporal Models. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021), July 6–7, 2021, Tampa, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3468791.3468806

1 INTRODUCTION

As AI expands into wide economical and societal activities, an increasing number of AI models are developed and embedded in diverse applications in many different domains. A particular class of models such as in [22] aims to predict spatio-temporal phenomena by capturing data patterns that are both spatially and temporally correlated to the prediction. Typically, in order to improve prediction accuracy different models are considered. The assumption is that models have been trained and validated independently – as in a traditional machine learning life-cycle – and are integrated into applications as black-box functions, i.e., without taking into account its architecture particularities.

In this paradigm, challenges emerge in the model selection process to answer a given prediction query. Black-box models exhibit different performance due to variation in the predictor's architecture, hyper-parameters' configuration, and the data samples observed during training [13]. Thus, for a given predictive query covering areas of varying spatio-temporal data correlation, a combination of spatio-temporal predictor models (STP), i.e. an STP Ensemble, may lead to an improved prediction accuracy, when compared to a single model prediction. However, (a) selecting the models to compose an ensemble and (b) defining their spatial allocation over a query region is a hard problem. In (a), the challenge involves predicting an STP model performance in a spatial region it may not have seen during training, while in (b) one must take into account the model varying performance within the query region.

In this paper, we propose DJEnsemble, an approach for the selection and allocation of models to compose a disjoint ensemble for

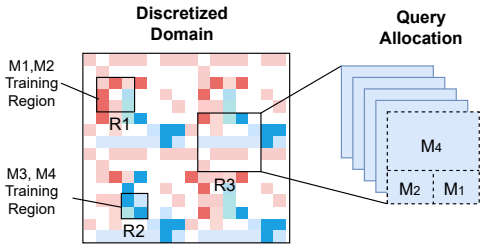

Figure 1: STPs M_1 and M_2 trained on a different region than two other STPs M_3 and M_4. On the right, the optimal allocation of models to answer a query over a different region

spatio-temporal predictions. This approach is orthogonal to the design and exploration of specific learning strategies and algorithms. It can be applied to a set of spatial models implementing algorithms varying from time-series predictors – such as ARIMA – to spatio-temporal deep learning models with different architectures and hyperparameters [5, 16].

We perform extensive experiments that evaluate our approach on two real datasets using six queries and a set of 36 STPs. Firstly, we show that the learning curve correctly approximates the prediction error as a function of data distribution distances. Next, we show that the optimization procedure implemented by DJEnsemble is capable of selecting a good ensemble plan out of a large number of possible predictor allocations. The approach is resilient to different scenarios involving the predictors' architecture and training datasets. Finally, we compare the results obtained by DJEnsemble against 5 other ensemble approaches, including both a traditional and a stacking ensemble. DJEnsemble achieves an accuracy improvement of up to $9X$ and it reduces query prediction time by a factor of up to $4X$.

We summarize our contributions as follows:

- We propose a data distribution based methodology to select spatio-temporal models for building ensembles
- We also propose a multi-variate cost model and allocation approach to support the DJEnsemble methodology
- We present a comprehensive experimental evaluation over two real datasets of meteorological observations, 6 queries, and 36 models.

2 PRELIMINARIES

In this section, we describe in more detail the analyzed problem, the spatio-temporal predictors considered and the Generalized Lambda Distribution (GLD), a probability density function used in our approach.

2.1 Spatio-Temporal Queries Optimization

Assume that we have access to multiple black-box STP models that can answer predictive spatio-temporal queries. The question we want to answer is: "How to select and spatially allocate an STP combination that gives the best possible performance, by measuring performance as a function of multiple parameters, including accuracy, execution time, and resource utilization?". This task can be formulated as the optimization problem of finding the optimal STP ensemble of black-box predictors that minimizes a multivariate cost function. We refer to this problem as the optimization of spatio-temporal predictive queries (OSTEMPQ) problem. The solution specifies an allocation of the selected predictor's spatial frames to the query region that forms a disjoint ensemble allocation. Figure 1 illustrates this scenario. On the left-hand side, it shows a spatio-temporal domain, where models M_1 and M_2 are trained in region R_1 and models M_3 and M_4 are trained in region R_2. R_3 corresponds to a spatio-temporal region of interest to a query. Finally, on the right-hand side, a solution to the query is given by the allocation of models M_1, M_2 and M_4. However, identifying such an ensemble is a hard problem as it involves: (i) estimating the prediction accuracy of each black-box predictor at the query region; (ii) defining a black-box model ensembling approach; (iii) finding the ensemble plan that minimizes the cost function; and (iv) planning for the execution.

We consider spatio-temporal deep learning models to solve regression problems, each models input and output being lists of fixed-size frames. Both input and output frames have the same size, and the number of frames that the model receives determines the rate of the input temporal signal. In this paper, we adopt the ConvLSTM architecture as our baseline. Given a particular convolutional model with a fixed-size frame and a spatial area where predictions are to be computed, we assume that multiple invocations of the model may be necessary to cover the entire area.

2.2 Modeling Data Distributions with GLDs

Tipically during training, a learner captures data patterns in an input dataset. Different approaches to learn data distributions and extract meta-features from the training data are proposed in the literature [1, 6]. We adopt the Generalized Lambda Distribution (GLD) probability density function (pdf) [19] because it can model an entire family of data distributions, such as Gaussian, Logarithm, and Exponential [4]. GLD encodes different data distributions through the specification of the lambda parameter representing statistical moments, where λ_1 and λ_2 determine the location (i.e., mean) and scale (i.e., standard deviation) parameters, while λ_3 and λ_4 determine the skew and kurtosis of the distribution, respectively. Then, a GLD is represented as $GLD(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$, known as the *RS* parametrization [19]. We fit GLDs to the distribution in the time-series at each spatial position and time seasonality interval [14]. Then, we use the λ parameters to identify regions sharing similar distributions (Section 3.1.1).

3 THE DJENSEMBLE APPROACH

We propose the DJEnsemble approach to solve the OSTEMPQ problem. DJEnsemble has an offline phase, in which the data and the candidate models are prepared, and an online phase, in which a cost function is used to select the model ensemble that answers the given query. DJEmsemble is implemented as an extension of the SAVIME multidimensional array database system [15].

3.1 Offline: Preprocessing

The offline phase consists of three different stages: Clustering, in which data is preprocessed and clustered into regions with similar distributions, tiling, in which we repartition the domain data into

nonaligned tiles and a last stage in which we estimate STP models performance on a query region.

3.1.1 Clustering. Consider a spatial domain $D(D, V)$, where $D = \{p_1, p_2, ..., p_n\}$ is its discretization into a set of localized 2D-points $p_i(x_i, y_i)$, with x_i and y_i being spatial coordinates. At every point $p_i \in D$, observations are recorded as a time series V. We perform clustering in order to group time-series with similar data distribution, adopting a feature-based approach [1]. The GLD function is used as a mechanism to compute the time-series features—exposed through its lambda parameters. We fit a GLD function to every time-series V_i from domain D and associate the four λ parameters (i.e., time-series features) to it. If V_i exhibits seasonality, we fit a separate GLD function to each season. Thus, domain D is represented as $D_t(p, V, \lambda_1, \lambda_2, \lambda_3, \lambda_4)$. After determining D_t, we cluster the time-series that have similar λ values together, using a *k-means* algorithm and *silhouette analysis* to guide the choice of k. Therefore, dataset D_t is transformed into $D_c(p, V, \lambda_1, \lambda_2, \lambda_3, \lambda_4, cid)$, where cid identifies the cluster each point belongs.

3.1.2 Tiling. Tiling has been introduced in the context of multidimensional array database systems. In [9], different tiling approaches are discussed. Among them, nonaligned tiling divides a multidimensional array into disjoint tiles, where the vertices of a tile do not intersect with those of neighboring tiles. In DJEnsemble, we adopt this tiling strategy in order to partition the domain into tiles of time-series sharing the same cluster id and, as a consequence, exhibiting time-series with similar data distribution. We start from an arbitrary point $p_i(\langle x_i, y_i \rangle, V_i)$ and aggregate neighboring points to form a hyper-rectangle—as long as they belong to the same cluster as p_i. We repeat this process until every point in D has formed a tile *Tile (id,*$\langle$*coordinates*$\rangle$*, centroid). coordinates* corresponds to the 3D spatial region covered by the tile, while *centroid* is the closest time-series to all the other series in the tile. We use *centroid* as a representative of a tile's data distribution in order to simplify the computation of the distance between two tiles.

3.1.3 Estimating model performance on query region. In addition to preprocessing the domain data, we need to be able to estimate the performance of every candidate model on any region of the domain. For this, we define a data distribution-based error prediction function as follows:

$$E_{gnr} = F_\epsilon(dist_{ij} + e_i) \quad (1)$$

$F_\epsilon(dist_{i,j} + e_i)$ is a monotonic non-decreasing function for every candidate model. $dist_{i,j}$ is the distance between the distribution in the model's training data d_i and a prediction region d_j. e_i denotes the model's generalization error, obtained on the testing dataset the model is evaluated on. We assume that the training and the testing datasets have similar distributions. Additionally, we adopt the shape-based *Dynamic Time Warping* (DTW) function [17] for computing the distance between the time-series representing the model training dataset centroid and the tile centroid.

For a given model, we compute its corresponding error function $F_\epsilon(dist_{i,j} + e_i)$ by fitting a polynomial model to a series of pairs $(dist, error)$. It has been shown that learning curves that estimate a model loss as a function of an distribution distance follow a power-law [8]. Thus, our goal is to find a polynomial function that maps the data distribution distance to prediction error. The fitting process works as follows: First we select the region from the test set. We modify these by sequentially adding a Gaussian noise with increasing variance. Distance between the original and modified dataset centroid series is calculated using the DTW function along with the error in applying the model. From the $dtw, error$ pair we fit a regression model of the form $error = F_\epsilon(dist_{i,j} + e_i)$ using cross validation. At the end of the offline preprocessing phase, the spatio-temporal series are partitioned into tiles of homogeneous data distributions—represented by their centroid spatio-time series.

3.2 Online: Query Processing

In the online phase of the DJEnsemble, the spatio-temporal predictive query Q is evaluated, and a set of STPs $M = \{m_1, m_2, \ldots m_s\}$ is considered to compute the predictions. This phase is split into three stages—planning, execution, and post-processing. In the planning stage, a set of black-box candidate predictors are selected and an allocation matrix is computed. In the execution stage, the selected models are evaluated according to the planned allocations. Lastly, post-processing actions are taken if necessary.

3.2.1 Model Ensembling Strategy. The single STP model approach to solve the OSTEMPQ problem is depicted in Figure 2 a) and b). In a), the model M_i completely covers the query region $Q.R$. Thus, a single instance of the model is sufficient to evaluate the query. In b), the area of the query region is larger than the model frame size. The traditional ensemble of STP models [21] – depicted in Figure 2 c) – evaluates query Q as follows. In the planning stage, it selects a subset $M' \subseteq M$ of models with testing accuracy higher than some threshold δ. An allocation matrix A is built for every model in M', such that the entire query region $Q.R$ is covered by every model. In the execution stage, the allocations are submitted to a prediction execution engine. Finally, in the post-processing stage, an aggregation operation computes a linear combination of the results.

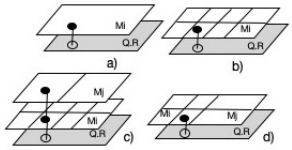

Figure 2: Strategies for answering spatio-temporal predictive queries: a) Single STP model, b) Single STP model with smaller frame size, c) Traditional Ensemble, and d) DJEnsemble.

The DJEnsemble approach, depicted in Figure 2d), extends the planning phase of the traditional ensemble approach. In addition to identifying M', it also computes the allocation matrix A using the cost function described in Equation 2. However, differently from the traditional ensemble, the allocation of each selected model may not cover the complete spatial query region. Instead, the allocation matrix associates each selected STP model to the spatial area in the query (i.e. a tile) it exhibits the best cost among the selected models. Thus, each tile in the spatial area covered by the query is associated to one and only one model in M'. The allocations in A are used

as input to an execution engine and the generated predictions are composed in the query result.

The main challenge of the DJEnsemble approach is to compute an assignment of models M to the query spatial area that minimizes the cost function. For each candidate model $m \in M$ with frame size m_{fs} ($m.frame\text{-}size$) – a fraction of the query frame size $q_{fs} = (R.size)$ – we can align its frame's top-left corner with any of the p_i spatial positions in $Q.R$. We can repeatedly apply this procedure until all the points in $Q.R$ are considered for prediction by a model in M. However, this exhaustive procedure hinders the ability to execute prediction queries efficiently due to the high overhead it incurs. Instead, we partition the query domain into *tiles*, as described in Section 3.1.2. Given that each tile covers a region with time-series having similar data distribution, we pick the model whose training data distribution resembles that of the tile's centroid the most. This procedure reduces the search space for selecting candidate models to every query tile, which are considerably fewer than the number of observation points.

Given a suggested allocation, the implication of a possible difference between $m.frame\text{-}size$ and the tile size is managed as follows. First, models are placed with the top-left corner matching that of the tile. Then, for models whose frame size is a fraction of the tile size, we place as many non-overlapping instances of the model so that the tile region is covered. Conversely, in case the model frame extends beyond the tile area, we only consider predictions on spatial points falling within the tile area.

3.2.2 Cost function. We design a cost function to optimize model allocation as the linear combination of a model's estimated generalization error and its estimated prediction execution time. The error estimate is computed by the error function, as described in Section 3.1.3. The estimate for the prediction execution time is obtained by averaging the model's previously recorded execution times, leading to a unitary cost (uc). Moreover, depending on the ratio between the tile's 2D size and the model frame size and a , a number $\lceil r \geq 1 \rceil$ of invocations of the model are required to cover all the points in the tile region. In this case, every candidate allocation $A(t_i, m_j)$ for tile t_i, model m_j, and a weighting parameter μ_e, is assigned a cost given by the formula:

$$Cost_{i,j} = (1 - \mu_e) \times F_\epsilon(dist_{i,j} + \epsilon_i) + \mu_e \times \lceil r_{i,j} \rceil \times uc \qquad (2)$$

This cost formula normalizes the generalization error and the prediction time to $[0, 1]$ intervals by dividing each value by the maximum value in the set of models, once outliers are eliminated from the set. The maximum values can be computed in a single pass over the predictions.

3.2.3 DJEnsemble algorithm. Algorithm 1 presents the DJEnsemble algorithm. It takes as input the query Q, the set of tiles T, the set of candidate models M, and a weight parameter μ_e. The *DJEnsemble* function returns the set of mappings $A(T_i, M_j)$ that satisfy the problem constraints and minimize the cost function detailed in Section 3.2.2.

Algorithm 1 DJEnsemble Algorithm

```
1:  function DJEnsemble(Q, T, M, μ_e)
2:      query_T ← queryTiles(Q, T)
3:      for q_t ∈ query_T in parallel do
          /* Extract the query tile centroid /*
4:          q_c ← q_t.getcentroid()
          /* Min priority queue /*
5:          pq_i ← ⊥
          /* Compute generalization error /*
6:          M.estimate_generalization_error(M, q_t)
          /* Remove models with estimated error outliers /*
7:          M' ← drop_outlier_model(M, q_t)
8:          for m ∈ M' do
          /* Compute prediction error estimate /*
9:              me ← m.generalization_error
10:             ex ← m.unitary_cost
11:             mf ← m.framesize
12:             qtf ← q_t.framesize
13:             c ← cost_function(me, mf, qtf, ex, μ_e)
14:             pq_i.push(c, < m, q_t >)
15:         end for
16:     end for
          /* Collect the best allocation */
17:     S ← ⋃_{i=1}^{|query_T|} pq_i.top()
18:     Return S
19: end function
```

4 EXPERIMENTS

In this section, we evaluate the assumptions considered in this work and the applicability of the DJEnsemble approach.

4.1 Setup

We introduce the experimental scenario and methodology, the computational environment, and the implementation of DJEnsemble in SAVIME.

4.1.1 Experimental scenario. The data used in the experiments is a subset of the Climate Forecast System Reanalysis (CFSR) dataset that contains air temperature observations from January 1979 to December 2015 covering the space between 8N-54S latitude and 80W-25W longitude (temperature dataset) [18]. Additionally, we also use a subset of the rainfall dataset from NASA's TRMM and GPM missions, with rainfall collected for the same spatial region as in the CSFR dataset over 22 years (rainfall dataset) [11].

All experiments used the CFSR dataset, except for the experiments in subsections 4.2.2 and 4.2.4, which used the rainfall dataset. The complete set of experiments uses thirty-six models adopting the ConvLSTM architecture. Twenty-one of them are trained in the temperature domain. Models SA_1 to SA_6 share the same architecture – filters, layers, etc. – and are trained in different regions. While models DA_1 to DA_7 are trained in different regions and with different architectures. Model SR_1 is considered a baseline model and is trained in the region where the predictions to answer the predictive query are computed. The last seven temperature models and the fifteen models trained on the rainfall dataset have all different architectures and are used to answer the five queries on real data (Figure 4). Our evaluation methodology considers root mean square error (RMSE) and composition execution time—as defined in Eq. 2.

4.1.2 Computational environment. The computing environment was kept constant throughout the experiments. It consists of a Dell PowerEdge R730 server with 2 Intel Xeon E5-2690 v3 @ 2.60GHz CPUs, 768GB of RAM, and running Linux CentOS 7.7.1908 kernel version 3.10.0-1062.4.3.e17.x86_64. The models were trained and tested on an NVIDIA Pascal P100 GPU with 16GB RAM.

4.2 Results

Next sections present the experiments.

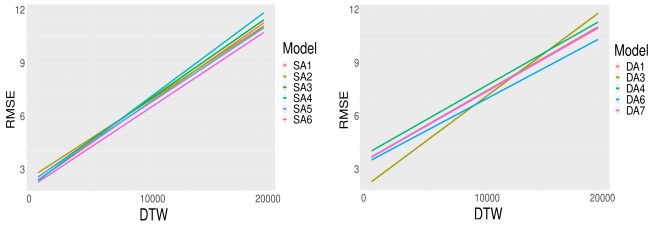

Figure 3: Generalization error on increasingly distant datasets for predictors with identical (left) and varying (right) hyper-parametrization.

4.2.1 Distance to error model fitting. An important assumption in this work is that we can predict the error of a black-box model on unseen data by a learning error function. In this experiment, we evaluate the $error\text{-}function$ predictor. Figure 3 depicts the generalization error curve for a set of black-box STP models $M = \{DA_1, \ldots, DA_7, SA_1, \ldots, SA_6\}$ obtained by applying the procedure presented in Section 3.1.3, on 50 datasets for each model. On the y axis, we plot the estimate for the generalization error, considering the RMSE in the spatio-temporal region d_i being predicted. Distances are computed between a base and a perturbed dataset using the DTW function and it is the measure on the x axis. One can observe that the error computed by the $error\text{-}function$ can be used to rank the models for a given STP prediction. Moreover, the results show that the $error\text{-}function$ reflects the generalization capacity of every model, as well as showing a strong correlation between distance and error—as the curves in both graphs are monotonically increasing.

4.2.2 DJEnsemble versus baseline approaches. We consider two baseline approaches to solve the OSTEMPQ problem. The first approach takes a single model trained on the same region as the query and uses it as a predictor (i.e., single model baseline). The second approach applies the traditional ensemble technique (i.e., ensemble baseline), as presented in Section 3.2.1.

Single model baseline. The single model baseline is constructed as follows. We use the SA_1 model architecture – which works best in most of the experiments – and train, validate, and test it on the same region in which query Q is specified, using a time interval from 1 to time-final (200). The remaining time slots are used for the predicting query.

Ensemble baselines. The second baseline includes multiple ensembles models. We build four ensembles and compare them against DJEnsemble and the single model baseline. The ensemble models are constructed using the seven models built with different architectures DA_1 to DA_7. The traditional ensemble is built considering all models in that list, combined as described in Section 3.2.1. The second ensemble selects models whose estimated error at all tiles is below 5 degrees. The model allocation follows the same strategy as in the traditional ensemble approach. The third ensemble extends the second ensemble approach by using tiling as a guide for model allocation, in addition to model filtering based on the error estimate at each tile. Finally, the fourth ensemble creates a stacking on top of the predictions of the seven base models. A multiple linear regression model (MLR) was trained and used for computing a linear combination of the ensemble predictions.To evaluate our approach, we built two DJEnsemble plans. Both implement the DJEnsemble approach, but ensemble (5) considers the choice of models to be allocated using the estimates for prediction error. The ensemble in (6) considers real errors in computing the cost function. The latter would correspond to an optimal solution.

Ensemble approaches	Error	Perf.	Exec.Time
1- Traditional ensemble	21.03	-838.83%	68.35 +–0.586
2- Ensemble-DTW distance	3.07	-37.05%	34.38 +–0.482
3- Ensemble-DTW and tiles	2.68	-19.64%	43.46 +–0.262
4- Stacking(MLR)-DTW distance	2.92	-23.28%	35.32 +–0.301
5- Single model baseline/SR_1	2.85	-21.00%	4.22 +–0.059
6- DJEnsemble (S)	2.35	-4.91%	14.06 +–0.193
7- **DJEnsemble (R)**	2.24	-	

Table 1: DJEnsemble vs ensemble baselines.

The results obtained by running the different ensemble approaches are included in Table 1. We can see that with respect to known ensemble approaches, DJEnsemble is more accurate and faster, outperforming the traditional ensemble approach accuracy in almost $9x$.

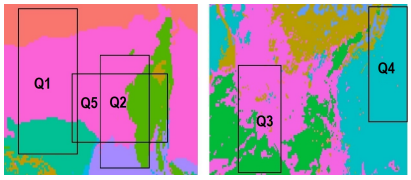

Figure 4: Queries over the temperature (left) and rainfall (right) domains.

4.2.3 Queries on temperature and rainfall datasets. We present the results obtained on five queries executed on data extracted from the temperature and rainfall datasets. Figure 4 depicts the regions corresponding to each query. Each color represents a region with different data distribution, associated to a different cluster. Table 2 summarizes the results and shows that DJEnsemble achieves the best rmse for all the queries.

Query	R[lat,lon]	Trad. ensemble	Stacking	DJEnsemble
Q1	[70:130,95:140]	25.01	7.28	3.35
Q2	[60:110,40:80]	27.88	5.52	4.29
Q3	[125:175,25:90]	14.28	13.96	5.34
Q4	[0:40,60:130]	8.80	12.94	6.04
Q5	[59:100,25:100]	29.10	5.04	3.18

Table 2: RMSE over the temperature and rainfall domains.

5 RELATED WORK

Meta-Feature Learning. In DJEnsemble, model selection follows a meta-learning approach [3]—datasets are used in learning a model that predicts deep learning models' performance, enabling model selection. This is similar to the approach developed in AutoGRD [6]. However, AutoGRD selects a single best performing model based on independent multi-variate prediction.

Ensembles. There are several approaches that apply deep learning techniques as model stacking strategies to improve the performance of base models [2], or that use a single deep learning architecture in which, while some layers of the architecture learn, others are used for second-level supervision acting as an ensemble [10]. Other strategies are ensemble-based black-box attacks to explore the vulnerability of the deep learning models—which is significant to choose effective substitute models for ensembles. The main difference between the approach we propose in the online phase and the existing literature is that our decisions are data-driven (i.e., data distributions) and address an auto-regressive problem.

AutoML and Model Serving Systems. DJEnsemble contributes to the broader topic of AutoML [12] with respect to model selection, an approach introduced in many state-of-the-art predictor serving systems, such as the framework Clipper [7] and the machine learning training and inference service Rafiki [20], just to name a few. In this sense, DJEnsemble considers a cost model that statically defines a prediction ensemble plan based on estimates for accuracy and prediction time. In fact, DJEnsemble can be implemented as a model selection solution if these systems provide a service for auto-regressive STP predictions.

6 CONCLUSIONS AND FUTURE WORK

We presented DJEnsemble, a disjoint ensemble approach to plan for the composition of black-box deep learning models to answer spatio-temporal auto-regressive predictive queries, and showed that it significantly outperforms traditional ensemble approaches in both accuracy and execution time. We believe there is plenty of future work to be explored. The offline phase can be further optimized, improving the identification of spatio-temporal patterns and reducing the pre-processing cost. Techniques to improve the error function accuracy – especially in higher dimensions – can also be investigated. The execution of the selected plans can take advantage of parallelism in the AI inference framework. Improvements in model design can also contribute to the overall prediction quality, especially considering the effect of different grid scales across datasets. Finally, multivariate predictions and how to adapt the data distribution distance-based approach to this scenario are topics to be explored.

ACKNOWLEDGMENTS

The authors would like to thank CAPES, FAPERJ, and CNPq for scholarships and research productivity fellowships. We also thank Petrobras, Gerência de Perfuração e Completação de Poços, and ANP for financing this work through contract 5850.0108 913.18.9. The UC Merced authors are supported by a US Department of Energy (DOE) Early Career Award. Finally, Fabio Porto thanks the National University of Singapore and Prof. Beng Chin Ooi for hosting him during the initial paper preparation.

REFERENCES

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. 2015. Time-Series Clustering—A Decade Review. *Information Systems* 53, C (2015).
[2] L. Ambrogioni, Y. Berezutskaya, U. Guclu, E.W.P. van den Borne, Y. Gucluturk, M.A.J. van Gerven, and E.G.G. Maris. 2017. Bayesian Model Ensembling Using Meta-trained Recurrent Neural Networks. In *Proceedings of 2017 NIPS Conference on Neural Information Processing Systems.*
[3] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. 2009. *Metalearning: Applications to Data Mining.* Springer.
[4] Y. Chalabi and W. Diethelm. 2012. Flexible Distribution Modeling with the Generalized Lambda Distribution. *ETH Econohysics Working and White Papers Series* (2012).
[5] Xingyi Cheng, Ruiqing Zhang, and Wei Xu. 2018. DeepTransport: Learning Spatial-Temporal Dependency for Traffic Condition Forecasting. In *Proceedings of 2018 IJCNN International Joint Conference on Neural Networks.* 1–8.
[6] Noy Cohen-Shapira, Lior Rokach, Bracha Shapira, Gilad Katz, and Roman Vainshtein. 2019. AutoGRD: Model Recommendation Through Graphical Dataset Representation. In *Proceedings of 2019 ACM CIKM International Conference on Information and Knowledge Management.* 821–830.
[7] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *Proceedings of 2017 NSDI USENIX Symposium on Networked Systems Design and Implementation.* 613–627.
[8] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In *Proceedings of 2015 IJCAI International Joint Conference on Artificial Intelligence.* 3460–3468.
[9] P. Furtado and P. Baumann. 1999. Storage of Multidimensional Arrays Based on Arbitrary Tiling. In *Proceedings of 1999 IEEE ICDE International Conference on Data Engineering.*
[10] Ping Hu, Dongqi Cai, Shandong Wang, Anbang Yao, and Yurong Chen. 2017. Learning Supervised Scoring Ensemble for Emotion Recognition in the Wild. In *Proceedings of 2017 ACM ICMI International Conference on Multimodal Interaction.*
[11] G. Huffman, D. Bolvin, D. Braithwaite, K. Hsu, R. Joyce, and P. Xie. 2014. NASA Global Precipitation Measurement (GPM) Integrated Multi-satellitE Retrievals for GPM (IMERG) v5.2. *NASA* (2014).
[12] F. Hutter, L. Kotthoff, and J. Vanschoren. 2019. *Automated Machine Learning: Methods, Systems, Challenges.* Springer.
[13] Daniel Kang, Raghavan Deepti, Peter Bailis, and Matei Zaharia. 2019. Model Assertion for Monitoring and Improving ML Models. In *Proceedings of 2019 SysML Conference.*
[14] Ji Liu, Noel Moreno Lemus, Esther Pacitti, Fábio Porto, and Patrick Valduriez. 2020. Parallel Computation of PDFs on Big Spatial Data Using Spark. *Distributed and Parallel Databases* 38, 1 (2020), 63–100.
[15] Hermano Lourenço Souza Lustosa, Anderson Chaves da Silva, Daniel Nascimento Ramos da Silva, Patrick Valduriez, and Fabio Porto. 2020. SAVIME: An Array DBMS for Simulation Analysis and ML Models Prediction. *Journal of Information Data Management* 11, 3 (2020).
[16] Yania Molina Souto, Fabio Porto, Ana Maria C. Moura, and E. Bezerra. 2018. A Spatiotemporal Ensemble Approach to Rainfall Forecasting. In *Proceedings of 2018 IJCNN International Joint Conference on Neural Networks.* 1–8.
[17] Minard Muller. 2007. *Information Retrieval for Music and Motion.* Springer.
[18] NCAR. 2010. NCEP Climate Forecast System Reanalysis (CFSR) 6-hourly Products, January 1979 to December 2010. https://doi.org/10.5065/D69K487
[19] John S. Ramberg and Bruce W. Schmeiser. 1974. An Approximate Method for Generating Asymmetric Random Variables. *Commun. ACM* 17, 2 (1974), 78–82.
[20] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: Machine Learning as an Analytics Service System. *Proc. VLDB Endow.* 12, 2 (2018), 128–140.
[21] Cha Zhang and Yunqian Ma. 2012. *Ensemble Machine Learning: Methods and Applications.* Springer.
[22] X. Zheng, J. Ye, Y. Chen, S. Wistar, J. Li, J. A. Piedra Fernández, M. A. Steinberg, and J. Z. Wang. 2019. Detecting Comma-Shaped Clouds for Severe Weather Forecasting Using Shape and Motion. *IEEE Transactions on Geoscience and Remote Sensing* 57, 6 (2019), 3788–3801.