# Technical Note: Data assimilation and autoregression for using near-real-time streamflow observations in long short-term memory networks

Grey S. Nearing[1], Daniel Klotz[2], Jonathan M. Frame[3], Martin Gauch[2], Oren Gilon[4], Frederik Kratzert[5], Alden Keefe Sampson[6], Guy Shalev[4], and Sella Nevo[4]

[1]Google Research, Mountain View, CA, United States
[2]LIT AI Lab & Institute for Machine Learning, Johannes Kepler University, Linz, Austria
[3]Department of Geological Sciences, University of Alabama, Tuscaloosa, AL, USA
[4]Google Research, Tel Aviv, Israel
[5]Google Research, Vienna, Austria
[6]Upstream Tech, Alameda, CA, USA

**Correspondence:** Grey Nearing (gsnearing@google.com)

**Abstract.** Ingesting near-real-time observation data is a critical component of many operational hydrological forecasting systems. In this paper we compare two strategies for ingesting near-real-time streamflow observations into Long Short-Term Memory (LSTM) rainfall-runoff models: autoregression (a forward method) and variational data assimilation. Autoregression is both more accurate and more computationally efficient than data assimilation. Autoregression is sensitive to missing data, however an appropriate (and simple) training strategy mitigates this problem. We introduce a data assimilation procedure for recurrent deep learning models that uses backpropagation to make the state updates.

## 1 Introduction

Long Short-Term Memory networks (LSTMs) are currently the most accurate and extrapolatable streamflow models available ~~from the hydrological science community (e.g., Kratzert et al., 2019c, b; Gauch et al., 2021a; Frame et al., 2021)~~(e.g., Kratzert et al., 2019 . Achieving the highest accuracy simulations possible in an operational setting requires the ability to leverage near-real-time streamflow observation data during prediction, wherever and whenever such data are available. There are two primary ways that rainfall-runoff models most often use near-real-time streamflow observation data: autoregression and data assimilation.

Autoregression (AR) has been a core component of statistical hydrology for decades (e.g., Matalas, 1967; Fernandez and Salas, 1986; Hsu et al., 1995; Abrahart and See, 2000; Wunsch et al., 2021). AR is also common in machine learning applications across many different types of domain applications (e.g., Uria et al., 2013; Vaswani et al., 2017; De Fauw et al., 2019; Child, 2020; Salinas et al., 2020; Dhariwal et al., 2020), including LSTM based approaches (e.g., Graves, 2013; Gregor et al., 2015; Van Oord et al., 2016). Most importantly for this discussion, ~~Feng et al. (2020); Moshe et al. (2020)~~ Feng et al. (2020) and Moshe et al. (2020) showed that AR improves streamflow predictions from LSTMs. AR modeling with LSTMs is complicated somewhat by the fact that LSTMs are sensitive to missing input data. In particular, a naive LSTM simply cannot run if any of its inputs are missing, and missing near-real-time ~~data is an issue for operational forecasting~~streamflow data is common, since in many parts of the world streamflow data are collected by hand or using sensors that are prone to malfunction, large measurement error, or breaks in communication with data loggers. It is possible to mitigate the problem of missing data in LSTM inputs by masking (e.g., Chollet, 2017, chapter 4), gap filling, or adversarial learning ~~to impute missing data~~ (Kim et al., 2020; Dong et al., 2021), however these strategies necessarily introduce some amount of bias in the inputs.

In contrast with statistical autoregressive models, conceptual and process-based rainfall-runoff models typically use data assimilation (DA) to ingest near-real-time streamflow observations. There are a number of different DA methods used in the Earth sciences (Reichle, 2008), ranging from direct insertion to full Markov Chain Monte Carlo approximations of nonlinear, non-Gaussian conditional probabilities (e.g., van Leeuwen, 2010). Most DA methods use filters or smoothers based on simplified probabilities – for example, variations of Kalman-type filters minimize variance (e.g., Evensen, 2003), particle filters maximize more general likelihoods (e.g., Del Moral, 1997) but run into challenges related to high dimensional sampling (Snyder et al., 2008), and variational filters numerically minimize specified loss functions (Rabier and Liu, 2003). All data assimilation methods fundamentally work by conditioning (changing) the states of a dynamical systems model so that information from observations persist in the model for some amount of time.

Like dynamical systems models, LSTMs have a recurrent state. This means that it is possible to use DA with LSTMs. This would allow ingesting near-real-time observation data without AR, making it possible to train LSTM models that are able to leverage near-real-time streamflow data where and when available. Further, LSTMs are trained with backpropagation, which means that there already exists a gradient chain through the model's tensor network that can be used for implementing certain types of inverse methods required for DA. Similar principles have been applied to update other features in deep learning models for a variety of purposes. For example, backpropagation to update inputs and specific layers has been used as an analytical tool (e.g., Olah et al., 2017; Dosovitskiy and Brox, 2016; Mahendran and Vedaldi, 2015) and to generate adversarial examples for training (Szegedy et al., 2013).

The major concern with statistical approaches (like AR) is that they often do not generalize to new locations or to situations that are dissimilar to the training data (e.g., Cameron et al., 2002; Gaume and Gosset, 2003). ~~However, rainfall-runoff~~ Rainfall-runoff models based on LSTMs generalize to ungauged basins ~~Kratzert et al. (2019b)~~ Kratzert et al. (2019b); Mai et al. (2022) and extreme events (Frame et al., 2021) better than both conceptual and process-based hydrology models~~. The performance of LSTM-based rainfall-runoff models can be improved significantly using autoregression, however LSTMs are sensitive to missing data, which means that ad hoc approaches must be employed for gap-filling. Additionally, autoregressive models cannot be applied to ungauged catchments where lagged streamflow data are not available. Missing streamflow data is a common problem in operational settings, since in many parts of the world streamflow data are collected by hand or using sensors that are prone to malfunction, large measurement error, or breaks in communication with data loggers. As an example, the Google flood forecasting model (Nevo et al., 2019) typically faces roughly 10% - 30% missing streamflow data during each monsoon season in different watersheds in India.~~ , however we do not know whether this will also be true for AR LSTMs. DA has at least a potential advantage over AR in that it is robust to missing data: whenever there is no observation data to assimilate, the original model continues to make predictions.

The purpose of this paper is to provide insight into trade-offs between DA and AR for leveraging potentially sparse near-real-time streamflow observation data. AR is easier to implement than DA (simply train a model with autoregressive inputs), and it is also more computationally efficient because it does not require any type of inverse procedure during prediction (e.g., variational optimization, ensembles for estimating conditional probabilities, high dimensional particle sampling, etc.). Inverse procedures used for DA not only require significant computational expense, but also are sensitive to ~~hyperparameters~~ (hyperpar)ameters related to things like error distributions, regularization coefficients, and resampling procedures (Nearing et al., 2018; Bannister, 2017; Snyder et al., 2008). On the other hand, AR suffers from ~~effects of missing data, which is a serious problem in an operational setting~~problems with missing input data. In this paper we compare two things: (i) a very simple procedure for dealing with missing data data in an AR LSTM model~~with~~ , and (ii) variational ~~data assimilation (also~~ DA applied to an LSTM~~model), and~~ . We show that the simple AR approach ~~works better~~is generally more accurate.

As a caveat, ~~it is important to point out that there are many strategies that probably could be developed or employed to deal with missing input data in LSTM streamflow models. Additionally, there are many different types of DA methods that could be used~~ DA is a large category of very diverse methods. We prefer to define *data assimilation* as any Bayesian or approximately Bayesian method for (probabilistically) conditioning the states of a dynamical systems model on observations. Most common DA methods fit this definition (e.g., ~~EnKF,~~ Kalman filters and smoothers, EnFK, ENKS, particle filters, variational filters and smoothers, etc.). The ~~variational DA strategy that we test here is one of the most common forms of DA, and one that is easily implemented in an ML setting because it directly leverages the deep learning tensor network. It is impossible for us to test every possible method for AR and~~ DA ~~that has been developed, however the goal of~~ method that we present here is novel – we use backpropagation through a tensor network to update the cell states of an LSTM (see Appendix C), – however we cannot claim that our results will hold for every type of DA. Similarly, the AR method we test here is extremely simple and it might be possible to improve on our methodology. However, our opinion is that both the DA and AR methods that we test are perhaps the most straightforward way to use either of these categories of approaches for ingesting near-real-time streamflow data, and our objective for this paper is ~~to provide guidance to forecasters and model developers about which of these approaches are worth pursuing in operational models. Our suggestion based on results presented within is that, at present, we suggest that it is likely more promising to develop AR-based approaches rather than DA-based approaches~~(aside from highlighting our new deep learning DA approach) is to provide some guidance on what might be most promising for operational modeling (e.g., Nevo et al., 2021).

## 2 Methods

### 2.1 Data

To allow for direct comparison with previous studies, we tested autoregression and backpropagation-based variational data assimilation using an open community hydrologic benchmark data set that is curated by the US National Center for Atmospheric Research (NCAR). This Catchment Attributes and Meteorological Large Sample data set (CAMELS; Newman et al., 2015; Addor et al., 2017) consists of daily meteorological and daily discharge data from 671 catchments in CONUS ranging in size from 4 $km^2$ to 25,000 $km^2$ that have largely natural flows and long streamflow gauge records (1980-2014). Again, to be consistent with previous studies (Kratzert et al., 2019c, 2021; Klotz et al., 2021; Gauch et al., 2021b; Newman et al., 2017; Frame et al., 2021), we used the 531 of 671 CAMELS catchments that were chosen for model benchmarking by Newman et al. (2017), who removed basins with (i) large discrepancies between different methods of calculating catchment area, and (ii) areas larger than 2,000 $km^2$.

CAMELS includes daily discharge data from the USGS Water Information System, which are used as training and evaluation target data. CAMELS also includes several daily meteorological forcing data sets (Daymet, NLDAS, Maurer) that are used as model inputs. Following Kratzert et al. (2021) we used all three data sets as inputs. CAMELS also includes several static catchment attributes related to soils, climate, vegetation, topography, and geology (Addor et al., 2017) that we used as input features – we used the same input features (meteorological forcings and static catchment attributes) that were listed in Table 1 by Kratzert et al. (2019c), and this table is ~~repordieed~~ reproduced in Appendix G.

### 2.2 Models

In total, we trained ~~61~~ forty-six (46) LSTM models. ~~One of these was a pure simulation model with no lagged streamflow data as inputs. This “simulation" model was used as a baseline for benchmarking performance of both DA and AR, and was also the base model used for data assimilation. 60 of the LSTMs were AR models: we tested 6~~ Twenty-six (26) of these models were trained and tested using a sample split in time (i.e., some years of data were used for training and some years for testing, but all CAMELS basins contributed training data to all models). Twenty (20) of these models were trained and tested using a cross-validation split in space (i.e., some basins were withheld from training and used only for testing). The latter mimics a situation where no streamflow data is available in a given location for training (i.e., an ungauged basin), but data becomes available at some point during inference. The purpose of these basin-split experiments is less to test a likely real-world scenario, as it is to highlight how the different approaches learn to generalize.

We trained two classes of models using both the space-split and basin-split approaches: simulation models and AR models. Simulation models do not receive lagged streamflow inputs and AR models do. Simulation models are used for baseline benchmarking and also for DA. One (1) simulation model was trained for the time-based train/test split, meaning that a single model was trained on all training data from all 531 basins. Ten (10) simulation models were trained for the basin-split – in that case we used a k-fold cross validation approach with $k = 10$. We used k-fold cross validation for the basin-split so that out-of-sample simulations are available for all 531 basins, to compare with models that used a time-split. The same time periods were used for training and testing in both the time-split and basin-split.

We trained time-split AR models at five different lag times~~for ingesting near-real-time streamflow data (~~, meaning that the autoregressive streamflow was lagged by 1, 2, ~~3, 5, 7, and~~ 4, 8, or 10 days~~), each associated with 10~~ , respectively. We also trained with different fractions of ~~missing data (~~the streamflow data record withheld (as inputs) during the training period (0%~~through 90%).~~ , 25%, 50% , 75%, 100%). This means that a total of twenty-five (25) AR models were trained on a time based train/test split. The reason for training AR models with different missing data fractions becomes apparent when we present results: models trained with some missing lagged streamflow inputs perform better when there is missing data during inference. Lagged streamflow data was withheld ~~randomly throughout the input time series, and the strategy that we used for handling missing data is described in Sect. 2.2.2.~~ at these fractions as random sequences of missing data with mean sequence length of five (5) days. For a full description about how data was withheld see Appendix A. We chose a mean sequence length of five days for withheld AR inputs because it at time lags greater than this, the AR models revert to accuracies that become similar to simulation (non-AR) models. The 100% missing data fractions test cases that are similar to having long periods of missing data. For each of these twenty-five (25) time split AR models, we performed inference with different amounts of missing data (the same fractions as used for training). This means that each trained time split AR model was used for inference five (5) times.

We trained and tested basin-split AR models only at lead time of one day, and only with a missing data fraction of 50%. This means we trained a total of ten (10) basin-split AR models.

We did not consider other types of missing data (i.e., meteorological forcings or basin attributes) because they are not central to the question at hand (how best to use lagged streamflow observations where those are available), and missing meteorological inputs ~~and missing basin attributes~~ are not common in operational models ~~(~~– most operational hydrology models require meteorological data at every timestep and most meteorological data sets are dense in time at the time resolution of the data set.

DA was performed on the trained simulation models – both the time-split and basin-split models. We performed DA on the one (1) time-split simulation model with the same missing data fractions as the time split AR models: 0%, 25%, 50% , and 75%, (100% missing data with DA is equivalent to the simulation model without DA). We also performed DA on the ten (10) ~~-~~ k-fold cross validation basin split simulation models with the same missing data fractions as the basin split AR models (50%).

### 2.2.1 Training

Daily meteorological forcing data and static catchment attributes were used as input features for all models, and daily streamflow records were used as training targets with a normalized squared-error loss function that does not depend on basin-specific mean discharge (i.e., to ensure that large and/or wet basins are not over-weighted in the loss function):

$$\text{NSE}^* = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_b}\sum_{n=1}^{\cancel{N}\,N_b}\frac{(\widehat{y}_n - y_n)^2}{(s(b)+\epsilon)^2}. \quad (1)$$

$B$ is the number of basins, ~~$N$~~ $N_b$ is the number of samples (days) per basin ~~$B$,~~ $b$, $\epsilon = 0.1$ is a constant designed to avoid divide-by-zero errors, $\widehat{y}_n$ is the prediction for sample $n$ (~~$1 \le n \le N$~~ $1 \le n \le N_b$), $y_n$ is the corresponding observation, and $s(b)$ is the standard deviation of the discharge in basin $b$ ($1 \le b \le B$), calculated from the training period (see, Kratzert et al., 2019c).

All models were trained using the training and test procedures outlined by Kratzert et al. (2019c). We trained for 30 epochs using sequence-to-one prediction to allow for randomized, small minibatches. We used a minibatch size of 256 and, due to sequence-to-one training, each minibatch contained (randomly selected) samples from multiple basins. We used 128 cell states and a 365-day sequence length. Input and target features were pre-normalized by removing bias and scaling by variance.

Gradients were clipped to a global norm (per minibatch) of 1. Heteroscedastic noise was added to training targets (resampled at each minibatch) with standard deviation of 0.005 times the value of each target datum. We used an ADAM optimizer with a fixed learning rate schedule with initial learning rate of 1e-3 that decreased to 5e-4 after 10 epochs and 1e-4 after 25 epochs. Biases of the LSTM forget gate were initialized to 3 so that gradient signals persisted through the sequence from early epochs (Gers et al., 2000). All models were trained on data from all 531 CAMELS catchments simultaneously. The training period was October 1, 1999 through September 30, 2008 and the test period was October 1, 1989 through September 30, 1999.

#### 2.2.2 Autoregression

The strategy that we used to deal with missing data in AR models was to replace missing lagged streamflow data with model-predicted streamflow data at the same lag time. This is related to a standard ML technique for training recursive models, discussed in Appendix B. Autoregression models were thus trained with two inputs in addition to the CAMELS data inputs described in Sect. 2.1: (i) streamflow lagged by some number of days (different lag times) and (ii) a binary input flag that represents whether any particular autoregressive input came from observation or from previous model predictions. The binary flag allows the model to differentiate between observed vs. simulated autoregressive inputs.

#### 2.2.3 Data Assimilation

The theory behind using backpropagation through tensor networks to perform variational data assimilation is given in Appendix C ~~-~~ – this is essentially a direct implementation of standard variational DA using tensor networks.

Data assimilation was performed during the test period on the “simulation" ~~LSTM outlined in Sect.~~ LSTMs outlined in Section 2.2. Unlike training an LSTM, where all training data from all catchments must be used together to train a single model ~~(Nearing et al., 2020; Gauch et al., 2021b), data assimilation~~ (Nearing et al., 2020; Gauch et al., 2021b), DA is independent between basins. As such, the loss function we used for data assimilation was a regularized Mean Squared Error (MSE) (for more details on the loss function used for data assimilation, see Appendix D)

We used the ADAM optimizer for data assimilation with a dynamic learning rate that started at ~~0.05~~ 0.1 and decreased by a factor of ~~0.1~~ 0.9 (90%) each time the update step loss failed to decrease. We used an assimilation window of five (5) timesteps (updating the cell state at timesteps $t-5$), ~~1000 update steps~~ 100 update steps (similar to epochs) with early stopping criteria if the learning rate decreased below $1e-6$, and we did not use any regularization. The search used to find these hyperparameter values for data assimilation is reported in Appendix F.

### 2.3 Testing & Evaluation

Following previous studies (cited in ~~Sect.~~ Section 2.1), we report a number of hydrologically relevant performance metrics, listed in Table 1. ~~We report all of these metrics for the simulation model (no lagged streamflow data as inputs), and also for data assimilation and autoregression models.~~ These metrics (and our evaluation procedure in general) were chosen to allow for benchmarking against previous studies (Kratzert et al., 2019c, b, 2021; Gauch et al., 2021a; Frame et al., 2020). Metrics in

**Table 1.** Overview of evaluation metrics

| Metric | Description |
|---|---|
| NSE$^{i}$ | Nash-Sutcliff efficiency |
| KGE$^{ii}$ | Kling-Gupta efficiency |
| Pearson-r | Pearson correlation between observed and simulat |
| $\alpha$-NSE$^{iii}$ | Ratio of standard deviations of observed and simu |
| $\beta$-NSE$^{iv}$ | Ratio of the means of observed and simulated flow |
| Peak-Timing-ErrorPeak-Timing$^{v}$ | Mean peak time lag (in days) between observed a |
| Peak-Timing-Abs-ErrorMissed-Peaks$^{vi}$ | Mean peak absolute time lag (in days) between ob |
| Peak-Abs-Bias$^{viii}$ Number of observed peaks without simulated peaks within 1 day Appendix ?? | |

$^{i}$: *Nash-Sutcliffe efficiency:* $(-\infty, 1]$*, values closer to one are desirable.*
$^{ii}$: *Kling-Gupta efficiency:* $(-\infty, 1]$*, values closer to one are desirable.*
$^{iii}$: $\alpha$*-NSE decomposition:* $(0, \infty)$*, values close to one are desirable.*
$^{iv}$: $\beta$*-NSE decomposition:* $(-\infty, \infty)$*, values close to zero are desirable.*
$^{v}$: *Peak-Timing:* $[0, \infty)$*, values close to zero are desirable.*
$^{vi}$: *Missed-Peaks:* $[0, 1]$*, values close to zero are desirable.*

this paper are reported for now-casting, meaning that we do not use meteorological forecast data and we do not predict beyond the end of the precipitation data.

AR models were trained with fractions of missing data (withheld randomly) between 0% and 90100% and tested on data with different fractions of missing data. This was done to understand what effect the training data fraction has on performance. After choosing an appropriate fraction of missing data for training AR models, these models were trained with streamflow inputs that had varying lag times (between 1 and 10 days). Both AR and DA models were tested with different fractions of (randomly) withheld lagged streamflow input data and different lag times, however all metrics were calculated on all streamflow observations within each basin during the entire test period, even when some of the lagged streamflow data were withheld as inputs.

## 3 Results

### 3.1 Training AR Models with Missing Data

Figure 1 compares median (over test periods in 531 basins) NSE values from AR models trained with 10 five (5) missing data fractions, each tested on 10 five (5) different missing data fractions (a total of twenty-five inference models). The primary signal in these results is that AR models lose accuracy as the fraction of missing lagged streamflow data in the test period increases. In general, training with fewer missing data is better if the fraction of missing data in the test period is also low. However, if the fraction of missing data in the test period is high, then it is better to train with more missing data. Mo matter how the AR

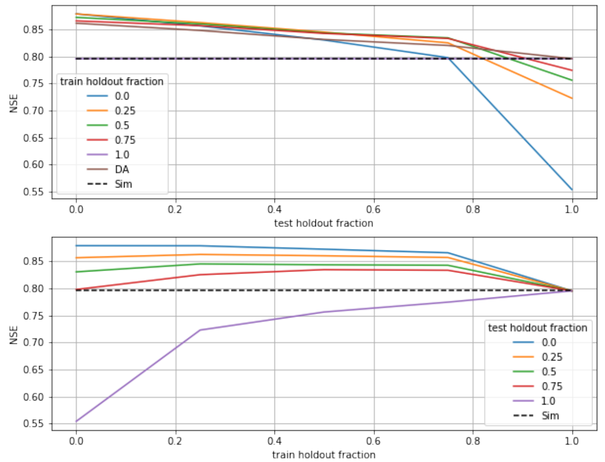


**Figure 1.** Median NSE scores of AR models trained and tested with different fractions of lagged streamflow data withheld. The two subplots show the same results, but organized by the amount of lagged streamflow data withheld during training vs. during testing.

model is trained, when all lagged streamflow data is withheld during inference, performance is similar to the simulation model.

For the remainder of our experiments (including basin-split experiments), we chose to benchmark AR models trained with 50% missing lagged streamflow inputs. This represents a compromise between training with too many or too few missing data that only degrades below the (median) accuracy of the pure simulation model with a missing data fraction of 90%.

### 3.2 ~~Benchmarking~~Time-Split Models

Table 2 lists the median (over 531 basins) performance metrics for all models with a lag of one day and no missing data. Figure 3 shows the distribution (over 531 basins) NSE scores for the same models. The major takeaways from these statistics are that both ~~variational assimilation and autoregression~~ DA and AR improved over the base LSTM model but ~~autoregression~~ AR was better. ~~Autoregression~~ AR (with no holdout) improved the median NSE (across test periods in 531 basins) by ∼10%, whereas ~~data assimilation~~ DA improved the median NSE by ∼8%. In general, ~~autoregression~~ AR with no missing data performed better than ~~data assimilation~~ DA across all metrics, and also across most basins (~~Fig.~~ Figure 2). Note that ~~autoregression~~ AR trained with and without any missing lagged streamflow data performed similarly ~~(Fig. 2).~~ with no missing data during inference.

As a point of comparison with previous work, (Feng et al., 2020) reported that autoregression improved LSTM median NSE by ∼19% (from 0.714 to 0.852), whereas we saw improvement to median NSE of ∼10% (from 0.796 to 0.879). The primary difference between that previous study and ours is that our baseline LSTMs were better (median NSE of ~~0.800~~ 0.796 vs. 0.714) due to the fact that the models reported in this study used multiple forcing data products (Kratzert et al., 2021).

**Table 2.** ~~Results for 1-day lag and no missing data~~Median performance metrics over 531 CAMELS basins.

| ~~Metric~~ | Simulation | AR ~~w/o~~ 0.0 holdout | AR ~~w/~~ 0.5 holdout | Assimilation |
|---|---|---|---|---|
| NSE | 0.796 | 0.879 | ~~0.876~~ 0.872 | ~~0.860~~ 0.862 |
| KGE | ~~0.791~~ 0.795 | ~~0.895~~ 0.896 | 0.896 | ~~0.841~~ 0.878 |
| Alpha-NSE | ~~0.875~~ 0.874 | ~~0.937~~ 0.942 | ~~0.953~~ 0.945 | ~~0.875~~ 0.913 |
| Pearson-r | 0.902 | ~~0.940~~ 0.939 | ~~0.933~~ 0.937 | 0.932 |
| Beta-NSE | ~~-0.030~~ -0.027 | ~~-0.012~~ -0.007 | ~~0.004~~ -0.002 | ~~-0.026~~ -0.014 |
| ~~Peak-Timing-Error~~ Peak-Timing | ~~0.000~~ 0.263 | ~~-0.115~~ 0.385 | ~~-0.129~~ 0.368 | ~~-0.206~~ 0.444 |
| ~~Peak-Timing-Abs-Error 0.206 0.304 0.300 0.360~~ Missed-Peaks | ~~0.342~~ 0.352 | ~~0.238~~ 0.250 | ~~0.265~~ 0.262 | 0.250 |
| ~~Peak-Abs-Bias 0.285 0.212 0.221 0.222~~ | | | | |

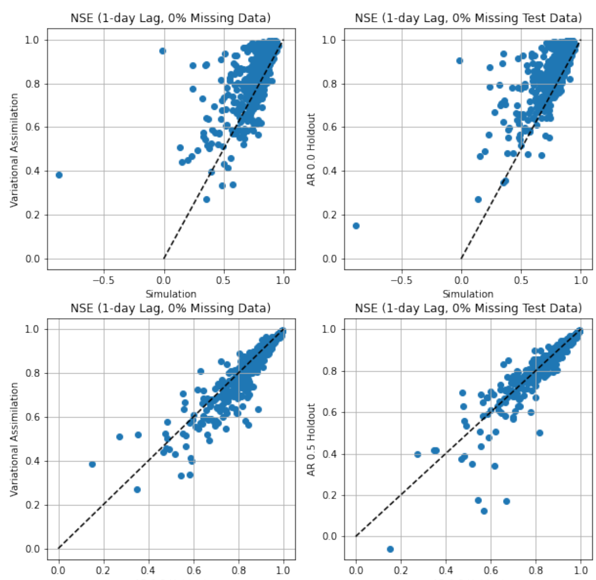


**Figure 2.** Comparison of per-basin NSEs with an observation lag of one day~~and~~: (top left) simulation vs. DA with no missing ~~streamflow input~~ data~~:~~, (~~left~~top right) ~~autoregression trained~~ simulation vs. AR with no ~~holdout~~ missing data, (bottom left) AR vs. ~~variational assimilation~~ DA with no missing data and (right) ~~autoregression~~ AR with no missing data vs. ~~without~~ AR with 50% ~~of lagged streamflow~~ missing data ~~withheld~~ during inference (no missing data during training to be comparable with other models shown in these plots).

Feng et al. (2020) also reported that autoregression was less informative in flashy basins, and there is a similar effect in our results (Appendix G reports similar efforts here to correlate differences between autoregression and data assimilation

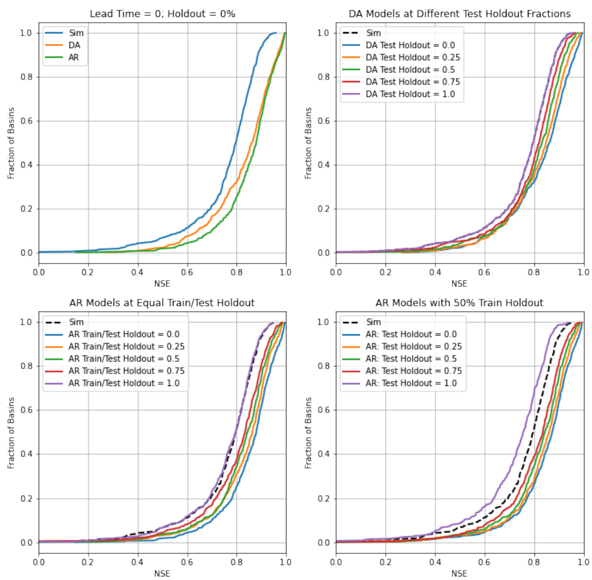


**Figure 3.** CDF plots of per-basin NSE scores. (Upper Left) Comparison between the main model types with no holdout. (Upper Right) Comparison of DA models with different holdout fractions during inference (testing). (Lower Left) Comparison of AR models with equal train and inference (test) holdout fractions. (Lower Right) Comparison of AR models with 50% train holdout and varying inference holdout. All results from lead times of 1 day.

with basin attributes related to climate, geology, soils, vegetation). While both autoregression and assimilation improved the average absolute peak timing error and also allowed the model to miss fewer peak-flow events, the peak timing error with both autoregression and data assimilation was always negative due to the fact that the model receives delayed information from lagged streamflow about any peak event that it might otherwise miss predicting from meteorological data alone. That is, we predict more peaks but the ones we pick up from using real-time inputs are lagged.

~~Fig.~~ Figure 4 compares the median NSE scores (over 531 basins) between the four models as a function of lag time in days and fraction of missing lagged streamflow data in the test period. ~~Autoregression~~ AR is always better than ~~data assimilation~~DA, but if the ~~autoregression~~ DA model is not trained with a fraction of missing data (here 50%), then performance decreases ~~if~~ whenever there is missing data in the test period. In this case, the autoregression model can perform worse than a simulation model with no lagged streamflow data. If the autoregression model is trained with an input flag to indicate whether a particular lagged streamflow value is from observation or simulation, then autoregression almost always improves on the baseline simulation model and is almost always better than data assimilation (up to large values of missing data). Similar figures for all metrics in Table 1 are given in Appendix H.

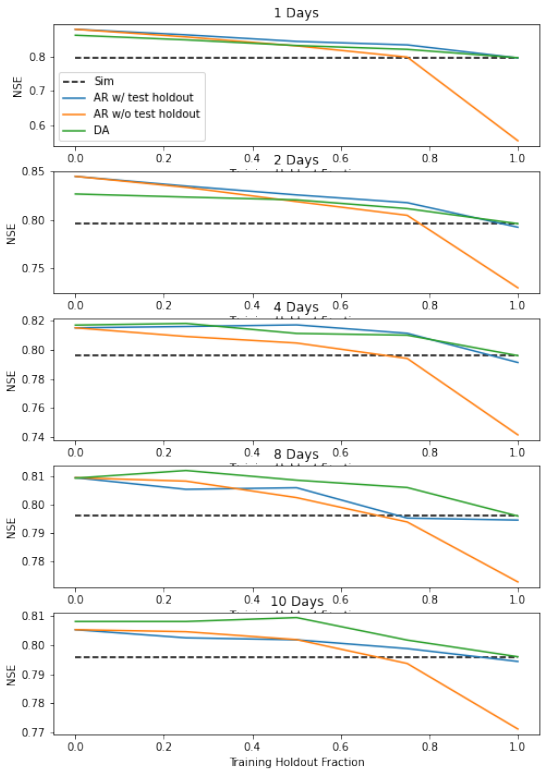


**Figure 4.** Median NSE over 531 basins of four models (simulation, ~~autoregression~~ AR trained with and without holdout data, and ~~data assimilation~~DA) as a function of lag time in days and fraction of missing lagged streamflow data in the test period. The AR and DA models here used 0% missing data during inference. Notice the different scales on the y-axis.

### 3.3 Basin-Split Models

Figure 5 compares the time-split and basin-split results at 1-day lead time with no training or test holdout. Previous studies have shown that LSTM models can often be used to predict in basins that did not supply training data, and the purpose of these basin-split experiments was to assess whether this ability to generalize holds when using observation data as inputs. This covers a class of use cases where data are not available for training in a given basin, but become available during inference, but more importantly, it helps us assess whether the model is learning general relationships between past and future streamflow.

Results in Figure 5 show that this is the case. The main thing to highlight in these results is that AR in basins that were "ungauged" for training but where data were available during inference (the basin-split AR model) generally performed

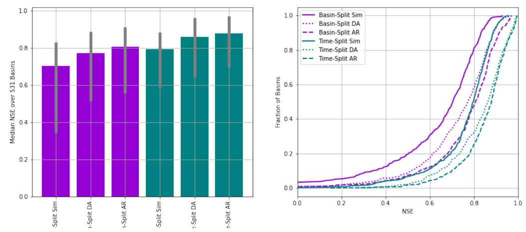


**Figure 5.** Comparison of NSE scores between Time-Split and Basin-Split models at 1-day lead time with no training or inference (test) holdout. The left subplot shows median scores (over 531 basins) and 80% interval (10th to 90th percentiles), and the right subplot shows distributions (over 531 basins)

better than basins where data was available for training but not for inference (the time-split simulation model). This indicates that adding AR to the model does not break extrapolatability – for example, the model learns a general representation of relationships between past streamflow and current inputs and can extrapolate those relationships to new basins. It does not simply learn to extrapolate streamflow in basins where it was trained.

## 4 Conclusions & Discussion

Data assimilation is necessary in order to use certain types of data to “drive" dynamical systems models. For example, if a model is based on some conceptual understanding of a physical system (like a conceptual process-based rainfall-runoff model), then the only way to use observations of system states ~~and outputs is via an~~ or outputs is through some type of inverse method. DA is a class of inverse methods that project information onto the states of a dynamical systems model. DA is often complicated to set up (e.g., choosing parameters to represent uncertainty distributions, sampling procedures, etc.), and often requires simplifying assumptions that cause significant information loss ~~(e.g., Nearing et al., 2018)~~(e.g., Nearing et al., 2018, 2013). ML models do not necessarily suffer from these same limitations – it is possible to simply train ~~the~~ models to use whatever data is available~~as inputs~~. This has several advantages~~over DA~~, including ease of use ~~and computational efficiency~~ (no additional hyperparameter tuning), computational efficiency (see below), and, apparently from our results, an increase in accuracy (less information loss).

It’s worth noting that in the experiments presented here, running DA for inference on the 10-year test period in 531 basins required approximately ∼2 hours of GPU time (NVIDIA Tesla v100) using the hyperparamters specified in Appendix F. Inference over the same basins and time period with an AR model required ∼30 minutes, and simulation required less than 5 minutes. The reason that AR is more expensive than simulation is because the AR LSTM is not CUDA-optimized, since the tensor network includes a gradient path from outputs to inputs. It might be possible to design an optimized version of this model (similar to the PyTorch optimized LSTM), however this is significantly outside the scope of our project.

DA has one advantage: it does not require that we choose how to withhold inputs to train the model. In cases where there is no target data during inference, AR models have potential to perform worse than a simulation model (see Figure 3), whereas DA models do not. DA reverts to a pure simulation model when there is no data to assimilate. This means that if the purpose of a model were to predict in a combination of gauged and ungauged basins, DA would allow you to use one trained model whereas AR would require separate models for gauged and ungauged basins. In principle, you would likely choose whether to use a simulation model or an AR model dynamically based on whether streamflow data were available at the time and place of each individual prediction. ~~The forward (ML) approach also allows the model to learn the best way to use the new input data streams directly, instead of through inverse methods. We've seen hints in previous work that the latter allows for more information extraction from data (Nearing et al., 2013).~~

To reiterate from the introduction, ~~it is impossible to test every AR or every DA method in a single study. We also~~ both DA and AR are broad classes of methods. We do not know of any benchmarking study in the hydrology literature that directly compares different DA methods ~~(typically a single method is applied and tested). Because of this, we cannot~~ over large, standardized, public data sets. We also do not make conclusive statements about ~~whether all DA methods might under-perform compared to AR in a ML context~~DA in general, however we have no particular reason to suspect that other DA methods might perform significantly better than variational DA ~~on large sample data sets. Given that DA systems generally require large investments of time and resources to develop and tune, the purpose of this current study is to give a sense of whether this type of investment might be worthwhile given that there is another option (AR) for ML-based models. The AR method we tested was exceptionally simple to implement and significantly out-performed the more complicated DA method. As such, the primary actionable conclusion of this study is that we do *not* recommend DA as a strategy for using~~ in this type of application. In principle, we would suspect that training a deep learning model to take all data as inputs (including near-real-time ~~streamflow data. We would instead recommend investing effort into better strategies for AR (e.g., different methods for dealing with missing data, using multiple data streams at potentially different spatiotemporal resolutions, etc. ).~~ lagged target data) will be more efficient than real-time inverse methods. The reason for this is that there is no a priori reason to suspect that a deep learning model (e.g., LSTMs) will be sub-optimal at extracting information from inputs data, especially in a way that other inverse methods (e.g., DA) might mitigate. Such cases might exist, but we do not see an a priori reason to suspect that this is the case. To summarize, our conclusion is that it is cheaper, easier, and generally more accurate to simply give your models all the data you have as inputs whenever possible.

*Code and data availability.* Plug-and-play code to reproduce the experiments reported in this paper is available at https://github.com/grey-nearing/lstm-data-assimilation.

odel code, including data assimilation and autoregression is available at https://github.com/grey-nearing/neuralhydrology. This is a fork of the NeuralHydrology codebase (https://neuralhydrology.github.io/). The fork contains a branch called 'assimilation' that contains code necessary for data assimilation.

CAMELS data is available at https://ral.ucar.edu/solutions/products/camels, with extensions to 2014 at https://www.hydroshare.org/resource/0a68bfd7ddf642a8be9041d60f40868c/ and https://www.hydroshare.org/resource/17c896843cf940339c3c3496d0c1c077/.

*Author contributions.* DK, AKS, and GN had the original idea for backpropagation-based data assimilation. All authors contributed to experimental design. GN wrote the data assimilation code, performed hyperparameter tuning, and conducted all experiments except correlations with basin attributes, which were done by JF. All authors contributed to experimental design. MG suggested to implement the input data flag for autoregression (which was transformative for AR skill). GN, MG, DK, and FK integrated the data assimilation code into the NeuralHydrology codebasde. GN wrote the paper with contributions from all authors.

*Competing interests.* The authors declare no competing interests

*Acknowledgements.* Frederik Kratzert FK was supported by a Google Faculty Research Award (PI: Sepp Hochreiter). Martin Gauch MG was supported by the Linz Institute of Technology DeepFlood project. Daniel Klotz DK was supported by Verbund AG.

## Appendix A: Sampling Missing Data

Streamflow input data was sampled at different missing data fractions for training and testing AR models and for DA. We masked continuous periods of missing data by using two Bernoulli samplers. We sampled ‘downshifts‘ and ‘upshifts‘ through a timeseries at different rates ($\rho_d$ and $\rho_u$, respectively). Moving sequentially through a timeseries, downshifts indicate that the missing data mask is turned on (a period of missing data begins), and upshifts indicate that the missing data mask turns off (a period of missing data ends). The $\rho_d$ and $\rho_u$ were chosen as to yield a mask with two properties: (i) an average masking sequence length of a given length (we used 5 timesteps), and (ii) a total masking density of a given percentage of the whole timeseries. These relationships are:

$$\rho_u = \frac{1}{mean\_missing\_length} \tag{A1}$$

$$\rho_d = \frac{\rho_u * missing\_fraction}{(1 - missing\_fraction)} \tag{A2}$$

(A3)

## Appendix B: Related ML Techniques for Autoregressive Modeling

The strategy we use for handling missing data in AR models is loosely related to a class of techniques used commonly to train recursive neural networks called *teacher-forcing* methods (Williams and Zipser, 1989). Teacher-forcing methods substitute model outputs from timestep $t$ as inputs into the network at timestep $t+1$ with observations during training (not during inference). This was originally used to avoid backpropagating through time, and the method is sensitive to differences between train and test samples which can result in divergent behavior when the model is run recursively during inference.

Our strategy for dealing with missing data in AR models does not replace the cell state (the recursive state of the LSTM)during training, but does run the risk of over-training to observed lagged streamflow inputs in cases where these data are sparse during inference (Figure 1). Our solution to this problem is to train with a combination of observed and simulated lagged streamflow inputs, which is a form of *scheduled sampling* (Bengio et al., 2015). This seems to work in this case (see Sect. 3.1). Another class of approaches to solving this problem are *professor-forcing* methods (Lamb et al., 2016), which uses adversarial learning to encourage a teacher-forcing network (i.e., trained with only observed inputs) to match the fully recursive model. This could be applied to the streamflow problem if AR models were to exhibit divergent behavior that cannot be solved with scheduled sampling.

## Appendix C: Variational Data Assimilation in LSTMs

The method of data assimilation that we will use in this paper is a type of variational assimilation. Variational assimilation works as follows (Rabier and Liu, 2003). We begin with a model that has time-dependent states, $\boldsymbol{c}[t]$ that determine a time-dependent observable, $\boldsymbol{y}[t]$, through a (potentially nonlinear) observation operator $\mathcal{H}$, up to random error $\boldsymbol{\varepsilon_y}[t]$:

$$\boldsymbol{y}[t] = \mathcal{H}(\boldsymbol{c}[t]) + \varepsilon_y[t] \tag{C1}$$

The model itself propagates through time according to some state transition function, $\mathcal{M}$, that operates on the state at the previous timestep and model inputs, $\boldsymbol{x}[t]$:

$$\boldsymbol{c_b}[t] = \mathcal{M}(\boldsymbol{c}[t-1], \boldsymbol{x}[t]) \tag{C2}$$

The *background state*, $\boldsymbol{c_b}[t]$, is the state estimated by model $\mathcal{M}$ at time $t$ without performing assimilation at time $t$ (although assimilation may have been performed at previous timesteps). The true (but unknown) state of the system is assumed to be equal to the background state up to random error $\varepsilon_c$:

$$\boldsymbol{c}[t] = \boldsymbol{c_b}[t] + \boldsymbol{\varepsilon_c}[t] \tag{C3}$$

Notating observations and states as drawn from distributions $p_y$ and $p_c$, we condition the model state on observations at time $t$ as:

$$p(\boldsymbol{c}[t]|\boldsymbol{y}[t], \boldsymbol{c_b}[t]) \propto p_y(\boldsymbol{y}[t]|\boldsymbol{c}[t]) p_c(\boldsymbol{c}[t]|\boldsymbol{c_b}[t]). \tag{C4}$$

The maximum likelihood estimate of the state vector is found by minimizing the negative log likelihood associated with Eq. C4. For example, if the state and observation errors ($\boldsymbol{\varepsilon_c}$ and $\boldsymbol{\varepsilon_y}$) are assumed to be normally distributed, the resulting loss function is:

$$\mathcal{J}(\boldsymbol{c}[t]) = (\boldsymbol{c}[t] - \boldsymbol{c_b}[t])^T \boldsymbol{B}^{-1} (\boldsymbol{c}[t] - \boldsymbol{c_b}[t]) + (\boldsymbol{y}[t] - \mathcal{H}(\boldsymbol{c}[t]))^T \boldsymbol{R}^{-1} (\boldsymbol{y}[t] - \mathcal{H}(\boldsymbol{c}[t])) \tag{C5}$$

where $\boldsymbol{B}$ and $\boldsymbol{R}$ are covariances of the state and observation errors, respectively. Analytical solutions are known for the special case when $\mathcal{H}(\cdot)$ is linear. Eq. C5 can be understood as a regularized loss function acting on target variables that is to be maximized with respect to the model states. If any component of this is not linear and Gaussian, then $\mathcal{J}(\cdot)$ must be minimized numerically.

The LSTM is described by the following equations:

$$\boldsymbol{i}[t] = \sigma(\boldsymbol{W_i}\boldsymbol{x}[t] + \boldsymbol{U_i}\boldsymbol{h}[t-1] + \boldsymbol{b_i}) \tag{C6}$$

$$\boldsymbol{f}[t] = \sigma(\boldsymbol{W_f}\boldsymbol{x}[t] + \boldsymbol{U_f}\boldsymbol{h}[t-1] + \boldsymbol{b_f}) \tag{C7}$$

$$\boldsymbol{g}[t] = \tanh(\boldsymbol{W_g}\boldsymbol{x}[t] + \boldsymbol{U_g}\boldsymbol{h}[t-1] + \boldsymbol{b_g}) \tag{C8}$$

$$\boldsymbol{o}[t] = \sigma(\boldsymbol{W_o}\boldsymbol{x}[t] + \boldsymbol{U_o}\boldsymbol{h}[t-1] + \boldsymbol{b_o}) \tag{C9}$$

$$\boldsymbol{c}[t] = \boldsymbol{f}[t] \odot \boldsymbol{c}[t-1] + \boldsymbol{i}[t] \odot \boldsymbol{g}[t] \tag{C10}$$

$$\boldsymbol{h}[t] = \boldsymbol{o}[t] \odot \tanh(\boldsymbol{c}[t]), \tag{C11}$$

$\boldsymbol{x}[t]$ are again the model inputs at time $t$, and $\boldsymbol{i}[t]$, $\boldsymbol{f}[t]$ and $\boldsymbol{o}[t]$ refer to the *input gate*, *forget gate*, and *output gate* of the LSTM, respectively. $\boldsymbol{g}[t]$ are the *cell inputs*, $\boldsymbol{h}[t-1]$ are the LSTM outputs, which are also called the *recurrent input* because these are used as inputs to all gates in the next timestep. $\boldsymbol{c}[t-1]$ are the *cell state* from the previous time step. Similar to dynamical systems models, the cell state, $\boldsymbol{c}[t]$ tracks the time-evolution of the system.

odel-predicted streamflow values comes from a *head layer*, and many LSTM studies in hydrology (e.g., Kratzert et al., 2018) have used a linear (dense) head layer:

$$y[t] = \boldsymbol{h}[t]\boldsymbol{w_h} + b_h \tag{C12}$$

Eqs. C11 and C12 effectively define the observation operator for data assimilation (Eq. C1). Notice that observations $\boldsymbol{y}[t]$ are not linear functions of the cell state (through the hyperbolic tangent operator in Equation C11), so no analytical solution to maximizing ~~eq~~ Eq. C4 or minimizing Eq. C5 exists. We therefore must minimize $\mathcal{J}(\boldsymbol{c}[t])$ numerically, which requires gradients with respect to the cell states.

In any deep learning model, the various weights, $\boldsymbol{W_*}$, and biases, $\boldsymbol{b_*}$, are trained by minimizing a training loss function $L(\cdot)$ using backpropagation along gradient chains like:

$$\frac{\delta L(\boldsymbol{x}[0:t],\boldsymbol{y}[0:t])}{\delta w_{*,j}} = \left(\frac{\delta L(\boldsymbol{x}[0:t],\boldsymbol{y}[0:t])}{\delta h_k[t]} \times (\frac{\delta h_k[t]}{\delta c_l[t]} \times \frac{\delta c_l[t]}{\delta *} \times \ldots \times \frac{\delta *}{\delta w_{*,j}}\right) + \ldots \tag{C13}$$

where the subscripts $j$ and $k$ (e.g., $w_{*,j}$, $h_k[t]$) indicate an arbitrary components of vectors or matrices (e.g., $\boldsymbol{W_*}$ like $\boldsymbol{W_i}$, $\boldsymbol{W_f}$, $\boldsymbol{W_o}$, or $\boldsymbol{W_g}$). $\boldsymbol{h}[t-1]$ are again the LSTM outputs (recurrent inputs), and the ellipses indicate that the network may have arbitrary depth. Eq. C13 is a simple derivative chain rule that any machine learning software library calculates automatically through the entirety of whatever tensor network defines a particular model. Almost all deep learning models are trained by backpropagating information through this type of gradient chain. Every time that the training loss function $L(\cdot)$ is calculated on a series of model inputs, $\boldsymbol{x}[0:t]$, and outputs, $\boldsymbol{y}[0:t]$, the values of all weights and biases in the model are updated based on perturbing in a direction that will decrease the loss according to these partial derivatives.

Notice that gradient chains like Eq. C13 necessarily include partial derivatives of the loss function with respect to features in the model that are not weights and biases. As an example, the partial derivative of loss $L$ with respect to weights in the input gate, $\boldsymbol{W_i}$, requires derivatives with respect to the cell states, $\boldsymbol{c}[t]$. To perform data assimilation, we can simply break the gradient chains to get partial derivatives of a loss function with respect to cell states like:

$$\frac{\delta L(\boldsymbol{x}[0:t],\boldsymbol{y}[0:t])}{\delta w_{i,j}} \frac{\delta L(\boldsymbol{x}[0:t],\boldsymbol{y}[0:t])}{\delta c_l t} = \left(\frac{\delta L(\boldsymbol{x}[0:t],\boldsymbol{y}[0:t])}{\delta h_k[t]} \times \frac{\delta h_k[t]}{\delta c_l[t]}\right) + \ldots \tag{C14}$$

Gradient chains like Equation C13 are used when training deep learning models. In this case, the loss function is calculated over a large number of historical data points (sometimes using minibatches). We want to be able to use streamflow observation data as it becomes available in near-real-time, which means that we want to use gradient chains like Equation C14 during inference rather than during training. These take the following form:

$$\frac{\delta \mathcal{L}(x[0:t],y[t-s:t])}{\delta c_l[t]} \frac{\delta \mathcal{L}(x[0:t],y[t-s:t])}{\delta c_l[t-s]} = \frac{\delta \mathcal{L}(x[0:t],y[0:t])}{\delta h_k[t]} \left(\frac{\delta \mathcal{L}(x[0:t],y[0:t-s:t])}{\delta h_k[t]} \times \frac{\delta h_k[t]}{\delta c_l[t]} \times \frac{\delta c_l[t]}{\delta *} \times \ldots \times \frac{\delta *}{\delta c_l[t-s]}\right) \tag{C15}$$

The primary difference between Equations C14 and C15 is that the loss function is calculated over observations a finite time period, $s$, into the past, $\delta L(x[0:t],y[t-s:t])$, and used to update cell states at the start of ~~the current~~ that observation period. We call $s$ the *assimilation window*. After the model is fully trained, and while it is running in forward mode to make new predictions, we can at any point calculate a loss function like $\delta\mathcal{L}(x[0:t],y[t-s:t])$ and use this to update the cell states of the LSTM using gradient chains like C15.

## Appendix D: Data Assimilation Loss Function

The loss function used for assimilation (i.e., $\mathcal{L}(\cdot)$ in ~~Eqs. C15or ??~~Eq. C15) do not need to be the same loss functions used for training (i.e., $L$ in Eq. C13). The derivatives that result from gradient chains in a deep learning tensor network can be calculated with respect to any loss function. Additionally, any loss function can be augmented with regularization – for example, to ensure that the updated cell states do not deviate too much from the values that are estimated by the trained model. The $R$ and $B$ matrices in Eq. C5 are an example of this type of regularization, and it is trivial to use this (or any other) type or regularization in the data assimilation loss function.

$$\mathcal{J}(\boldsymbol{c}[t-s]) = \alpha_c(\boldsymbol{c}[t-s]-\boldsymbol{c_b}[t-s])^T(\boldsymbol{c}[t-s]-\boldsymbol{c_b}[t-s]) + \alpha_y(\boldsymbol{y}[t-s:t]-\widehat{\boldsymbol{y}}[t-s:t])^T(\boldsymbol{y}[t-s:t]-\widehat{\boldsymbol{y}}[t-s:t]) \tag{D1}$$

Coefficients $\alpha_c$ and $\alpha_y$ are constants that are analogous to the $B$ and $R$ terms in Eq. C5. Since, in this form, these are mixing parameters, our experiments assume that $\alpha_y = 1-\alpha_c$. The assimilation window, ~~and $s$is an integer number of timesteps that defines an assimilation windowso that the cell state at time $t$ is updated based on observations through time $t+s$.~~ , is as in Eq. C15. Gradient chains like Eq. C15 do not look forward in time in the sense that the derivative of $\mathcal{L}$ with respect to $\boldsymbol{c}[t]$ does not depend on any observation prior to time $t$. This means that the assimilation loss Eq. D1 is general in $s$.

**Table F1.** Data assimilation hyperparameter tuning grid search and final values.

| Hyperparameter | Grid Search | Best Value |
|---|---|---|
| Initial Learning Rate | [~~0.05, 0.01~~1e-4, 1e-3, 1e-2, 5e-2, 1e-1] | ~~0.05~~ 0.1 |
| Learning Rate Epochs Drop | [1, 5, 10, 50, inf] | inf |
| Learning Rate Drop Factor | [~~0.9~~0.1, 0.5, ~~0.1~~0.9] | 0.1 |
| Assimilation Window$^{i}$ | [1, 3, 5, 20] | 5 |
| Assimilation History | [1, 5, 20, 50] | 20 |
| Epochs | [5, 10, 100, 1000] | 100 |
| Regularization$^{ii}$ | [0, 0.01, 0.1, 1, 2] | 0 |

$^{i}$: *This is s from Eq. D1*

$^{i}i$: *This is $\alpha_c$ from Eq. D1, and it is assumed that $\alpha_y = 1 - \alpha_c$*

## Appendix E: Description of ~~Peak-Timing Metrics~~Missed Peaks Metric

~~Peak timing metrics were~~ The missed peaks metric is calculated by first locating all peaks in the observation and simulation time series that satisfy the following two criteria: (1) observed and simulated peaks must be at least 30 days apart, and (2) peaks must be above the ~~$95^t h$~~ 80th flow percentile in a given basin. ~~We report four statistics: The fraction of observed peaks that the model predicts (above the $95^t h$ flow percentile)~~ Any peak in the observed time series that meets these two criteria and for which there is not a peak in the simulated time series that also meets these criteria within 1 day ~~of an observed peak. The average timing error (in unit days) calculated only on peaks that the model predicts. The average absolute timing error (in unit days) calculated only on peaks that the model predicts. The absolute percent bias of peak flow calculated only on peaks that the model predicts.~~ (before or after) is considered a missed peak. We report the fraction of observed peaks that are missed.

## Appendix F: Hyperparameter Tuning

Hyperparameter tuning for data assimilation was done with a validation period (1980-1989) that is distinct from both the training (1999-2008) and test periods (1989-1999) outlined in Sect. ~~??~~2.2.1. Due to computational expense, hyperparameter tuning was done on a subset of ~~20~~ fifty-three (53) basins out of the 531 used for the rest of the study ~~. The grid search~~(approximately 10%). We used a simple grid search, which is outlined in Table F1. This grid search resulted in the final data assimilation hyperparameters listed in the right-most column of Table F1.

We ~~tested~~ used a learning rate scheduler that dropped the learning rate every $N$ epochs. This is the *Learning Rate Epochs Drop* hyperparameter in Table F1 and did not improve performance. Because we used sequence-to-one prediction~~(Sect. 2.2.1)~~, we did not perform assimilation through the entire time series~~. The~~ , and the *Assimilation* ~~*Window*~~*History* hyperparameter determines how far back in each sequence (before time of prediction) we start assimilation. This parameter is a multiple of the *Assimilation Window*. The *Assimilation Window* itself is $s$ in Equation C15.

In our setup $\alpha_c$ (see Equation D1) was set as a hyperparameter and not trained directly, although learning this parameter through backpropagation is possible. Our hyperparameter search returned a value of $\alpha_c = 0$ (and ~~and~~ implied value of $\alpha_y = 1$), meaning that regularizing the loss function was not helpful.

## Appendix G: Performance by Basin Attributes

We tested whether it was possible to predict where DA or AR might offer the most benefit by using CAMELS catchemnt attributes (Addor et al., 2017). These attributes and their abbreviations are listed in Table G1. We used random forest models trained with static catchment attributes as inputs using k-fold cross validation to measure the predictability of the increase or decrease in test-period NSE scores at individual basins. The objective was to determine which types of hydrological characteristics determine the value or information content of lagged streamflow data.

Results for this analysis are given in Figure G1. The top subplots of Figure G1 illustrate the ability to predict test-period NSE scores from basin attributes in the three models (simulation, AR, DA), and the bottom subplots illustrate the ability to predict differences between test-period NSE scores from the different models. Kratzert et al. (2019c) found that forest fraction was strongly correlated with a predictor of the basin similarity mapping in an LSTM, and we see a similar effect in the top left subplot of Figure G1 (forest fraction is the second highest predictor of the skill of the simulation model).

Feng et al. (2020) reported that AR was less informative in flashy basins, and we see some evidence of that effect here: in particular, the frequency of low precipitation days was the strongest predictor of AR skill such that lower fractions of rainfall occurring in low intensity events corresponds with higher AR skill. Similarly, basin area was the third strongest predictor of AR skill, with AR being better in larger basins.

Snow fraction was the second strongest predictor of skill for both AR and DA, whereas this was not a strong predictor of skill in the pure simulation model. Kratzert et al. (2019a) showed that LSTMs can learn to store and release snow (without seeing snow data), however snowpack introduces correlations in streamflow time series that are exploited by both DA and AR.

The basin attributes that were the most important for determining NSE *improvements* due to both DA and AR (bottom center and bottom right subplots of Figure G1) were (i) mean annual precipitation and (ii) high precipitation frequency. High precipitation frequency was positively correlated with both AR and DA performance improvements. The reason for this is error in the rainfall data – AR and DA allow the model to effectively "see" streamflow events that occur due to unobserved or under-observed rainfall, although it takes one timestep for the model to register that a large event happened. This helps the model to avoid large errors for events with long recession curves. In general, average precipitation was negatively correlated with improvements due to incorporating lagged streamflow data, since precipitation events in general reduce autocorrelation in the

**Table G1.** Table of static catchment attributes. Descriptions taken from Addor et al. (2017)

| | |
|---|---|
| p_mean | Mean daily precipitation. |
| pet_mean | Mean daily potential evapotranspiration. |
| aridity | Ratio of mean PET to mean precipitation. |
| p_seasonality | Seasonality and timing of precipitation. Estimated by representing annual precipitation and temperature as sin waves. Positive (negative) values indicate precipitation peaks during the summer (winter). Values of approx. 0 indicate uniform precipitation throughout the year. |
| frac_snow_daily | Fraction of precipitation falling on days with temperatures below $0°C$. |
| high_prec_freq | Frequency of high precipitation days (>= 5 times mean daily precipitation). |
| high_prec_dur | Average duration of high precipitation events (number of consecutive days with >= 5 times mean daily precipitation). |
| low_prec_freq | Frequency of dry days (< 1 mm/day). |
| low_prec_dur | Average duration of dry periods (number of consecutive days with precipitation < 1 mm/day). |
| elev_mean | Catchment mean elevation. |
| slope_mean | Catchment mean slope. |
| area_gages2 | Catchment area. |
| forest_frac | Forest fraction. |
| lai_max | Maximum monthly mean of leaf area index. |
| lai_diff | Difference between the max. and min. mean of the leaf area index. |
| gvf_max | Maximum monthly mean of green vegetation fraction. |
| gvf_diff | Difference between the maximum and minimum monthly mean of the green vegetation fraction. |
| soil_depth_pelletier | Depth to bedrock (maximum 50m). |
| soil_depth_statsgo | Soil depth (maximum 1.5m). |
| soil_porosity | Volumetric porosity. |
| soil_conductivity | Saturated hydraulic conductivity. |
| max_water_content | Maximum water content of the soil. |
| sand_frac | Fraction of sand in the soil. |
| silt_frac | Fraction of silt in the soil. |
| clay_frac | Fraction of clay in the soil. |
| carb_rocks_frac | Fraction of the catchment area characterized as "Carbonate Sedimentary Rocks". |
| geol_permeability | Surface permeability (log10). |

streamflow time series. This effect appears worse for DA than AR (bottom left subplot), although this is a weak signal because

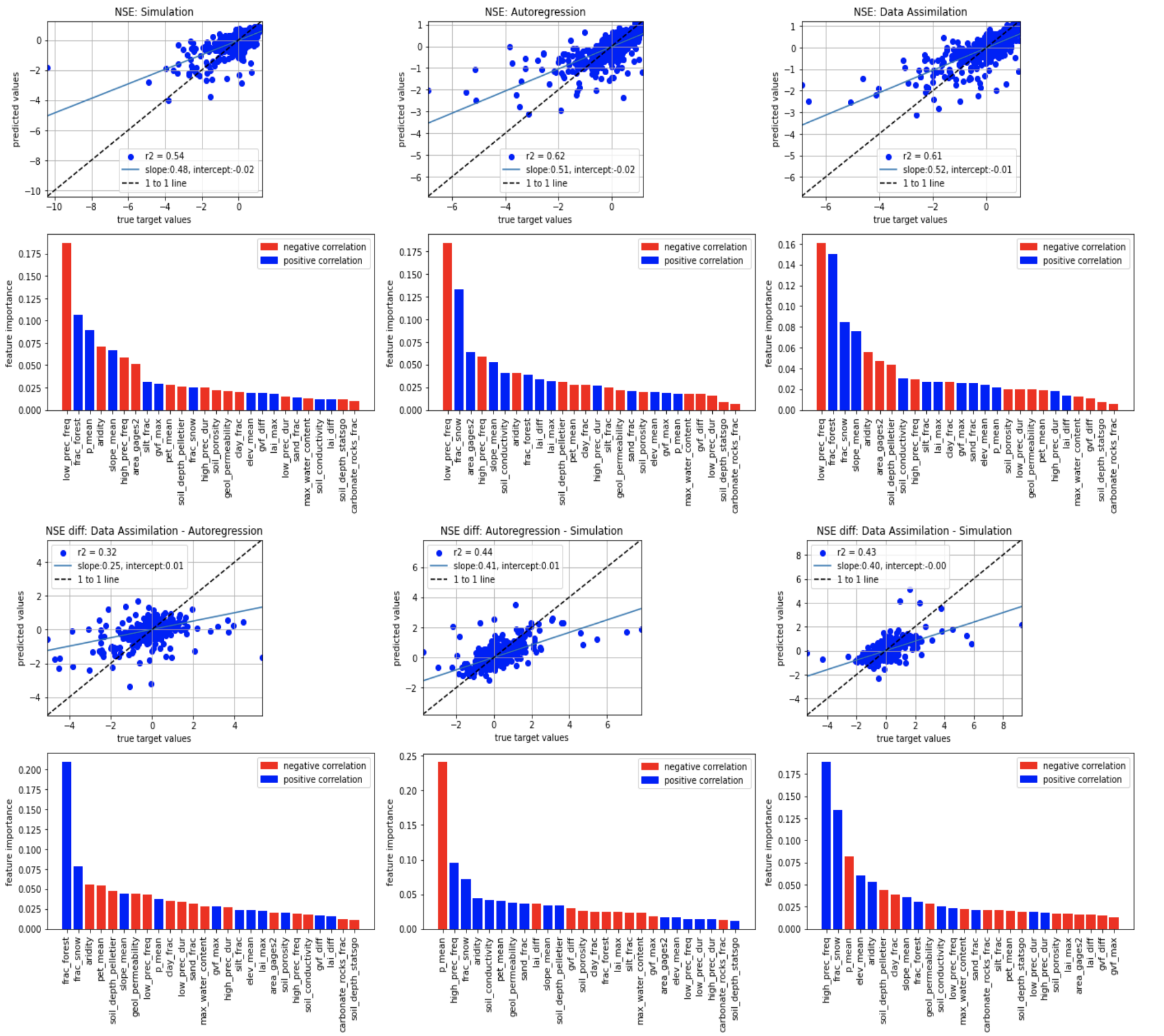


**Figure G1.** Scatterplots, $r^2$ metrics, and feature importances for predicting test-period NSE scores using different models (top subplots), as well as for predicting differences between models (bottom subplots). Bar charts show the feature importance for these predictions with blue indicating positive correlations between a given basin attribute and the NSE (or delta-NSE) and red indicating a negative correlation.

we were generally unable to predict differences between the NSE scores of DA and AR ($r^2 = 0.12$; bottom right subplot in Figure G1).

Figure G2 shows the spatial distribution of DA and AR NSE improvements relative to simulation. In both cases – but especially for DA – there is a group of basins in the Midwest and Southeast United States where performance was harmed by adding lagged streamflow data. We are unsure of the reason for this, but it warrants further exploration.

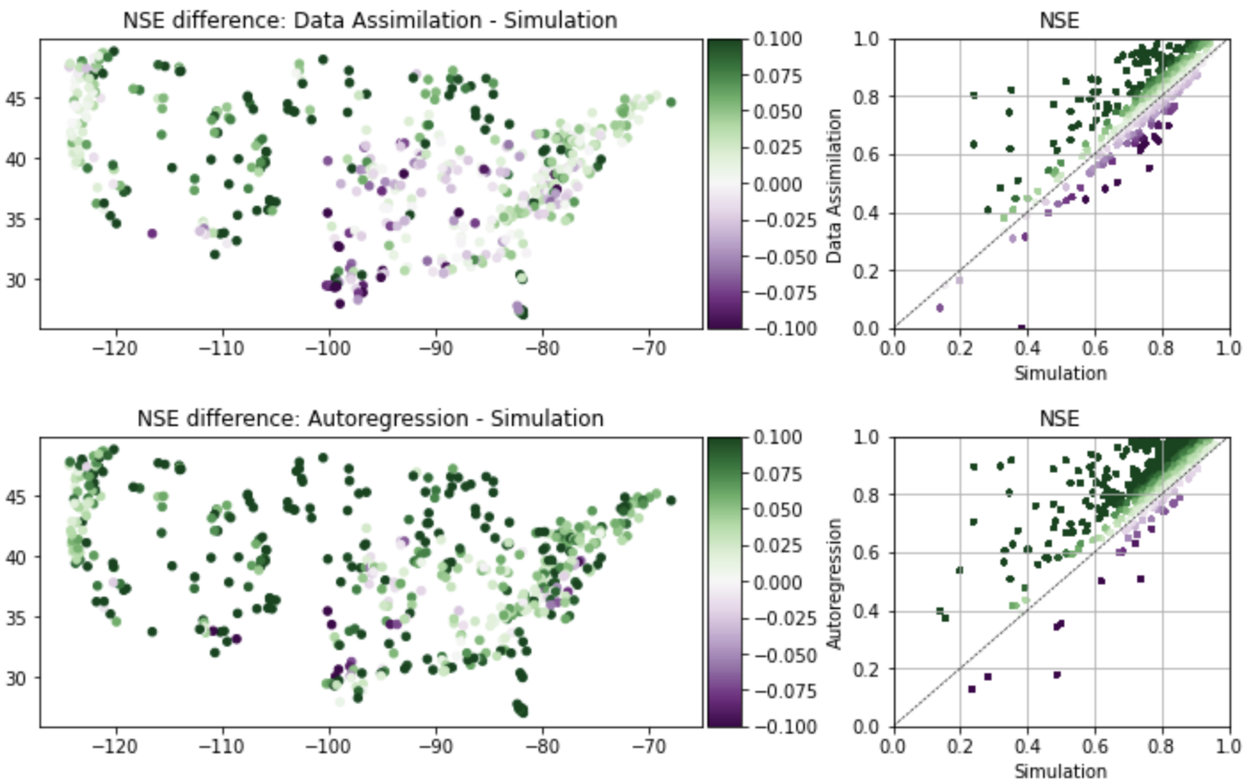


**Figure G2.** The performance difference (NSE score) between top: autoregression and the baseline simulation, and bottom: data assimilation and the baseline simulation.

## Appendix H: All Metrics Figures

This appendix contains figures similar to ~~Fig.~~ Figure 4 for all metrics listed in Table 1. These figures compare the median (over 531 basins) performance of four models (simulation, autoregression trained with and without holdout data, and data assimilation) as a function of lag time in days and fraction of missing lagged streamflow data in the test period.

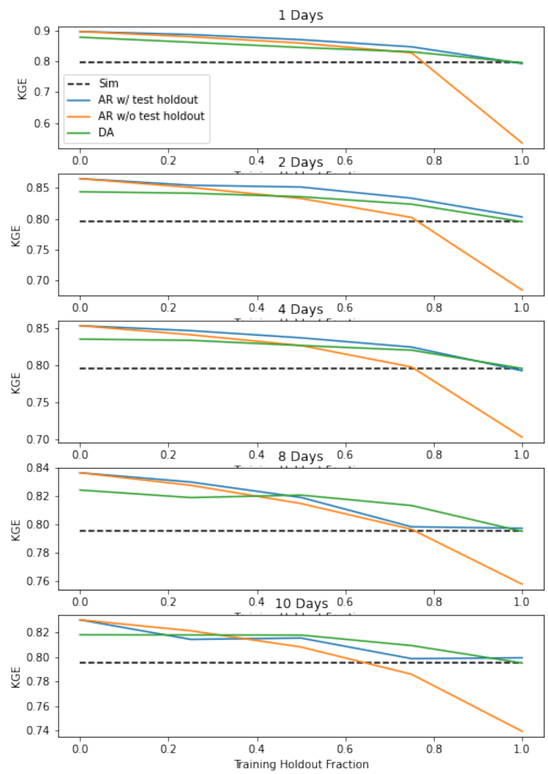


**Figure H1.** Same as ~~Fig.~~ Figure 4 but for Kling-Gupta Efficiency.

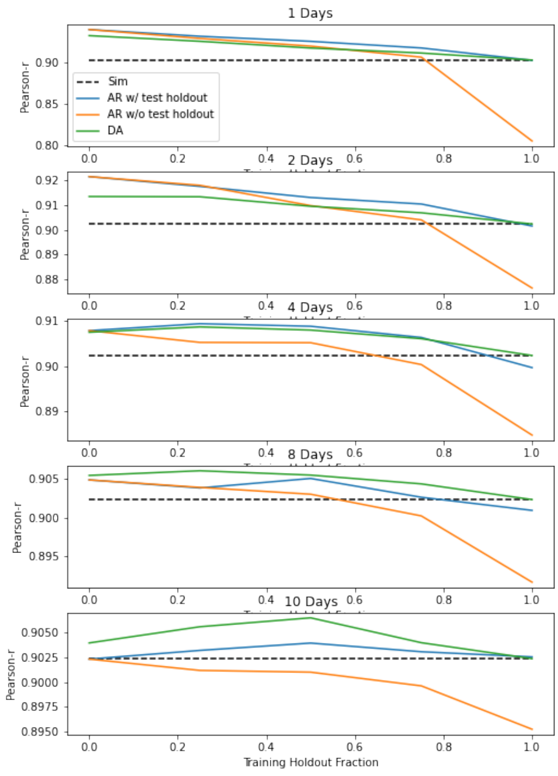


**Figure H2.** Same as ~~Fig.~~ Figure 4 but for the Pearson correlation coefficient.

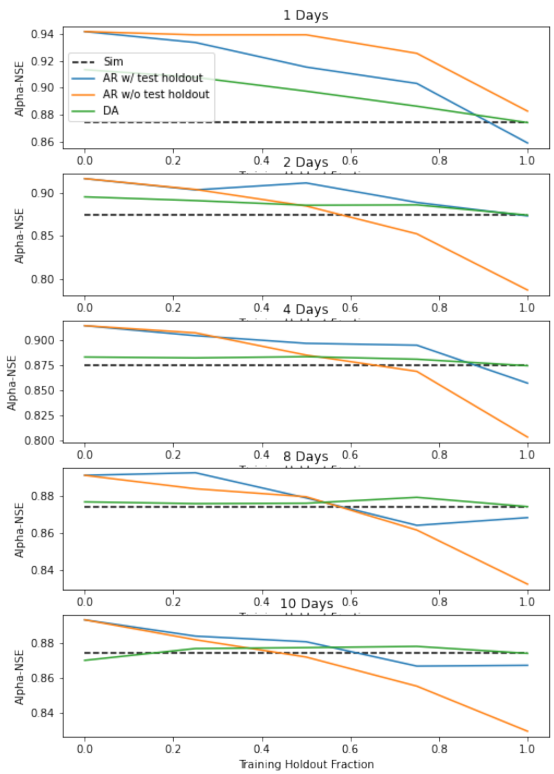


**Figure H3.** Same as Fig. Figure 4 but for $\alpha - NSE$, which is the ratio of the standard deviation of the observed vs modeled hydrographs.

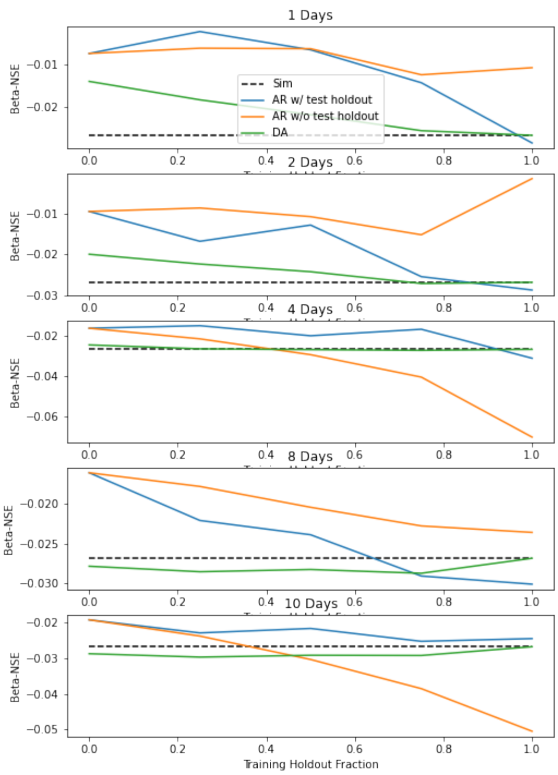


**Figure H4.** Same as ~~Fig.~~ Figure 4 but for $\beta - NSE$, which is the ratio of the means of the observed vs modeled hydrographs.

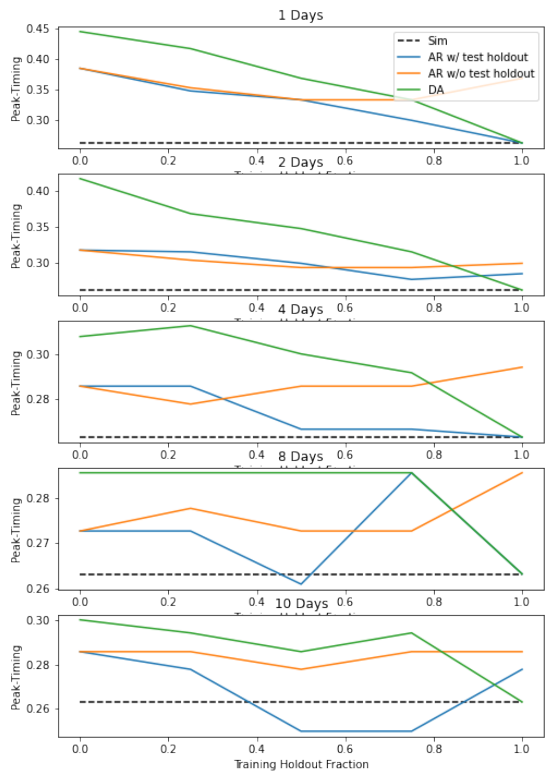


**Figure H5.** Same as ~~Fig.~~ Figure 4 but for peak timing error (Appendix ~~??~~E).

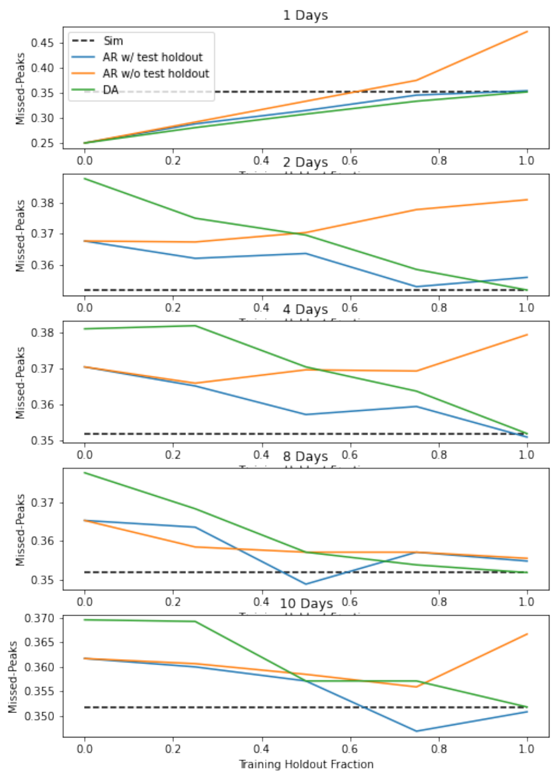


**Figure H6.** Same as ~~Fig.~~ Figure 4 but for ~~absolute~~ the fraction of missed peak ~~timing error~~ flow events within 1 day above 80th flow percentile (Appendix ~~??~~E).

~~Same as Fig. 4 but for the fraction of missed peak flow events (Appendix ??).~~

~~Same as Fig. 4 but for percent absolute bias (Appendix ??).~~

## References


Abrahart, R. J. and See, L.: Comparing neural network and autoregressive moving average techniques for the provision of continuous river flow forecasts in two contrasting catchments, Hydrological processes, 14, 2157–2172, 2000.

Addor, N., Newman, A. J., Mizukami, N., and Clark, M. P.: The CAMELS data set: catchment attributes and meteorology for large-sample studies, Hydrology and Earth System Sciences (HESS), 21, 5293–5313, 2017.

Bannister, R.: A review of operational methods of variational and ensemble-variational data assimilation, Quarterly Journal of the Royal Meteorological Society, 143, 607–633, 2017.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N.: Scheduled sampling for sequence prediction with recurrent neural networks, arXiv preprint arXiv:1506.03099, 2015.

Cameron, D., Kneale, P., and See, L.: An evaluation of a traditional and a neural net modelling approach to flood forecasting for an upland catchment, Hydrological Processes, 16, 1033–1046, https://doi.org/10.1002/hyp.317, 2002.

Child, R.: Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images, in: International Conference on Learning Representations, 2020.

Chollet, F.: Deep learning with Python, Simon and Schuster, 2017.

De Fauw, J., Dieleman, S., and Simonyan, K.: Hierarchical autoregressive image models with auxiliary decoders, arXiv preprint arXiv:1903.04933, 2019.

Del Moral, P.: Nonlinear filtering: Interacting particle resolution, Comptes Rendus de l'Académie des Sciences-Series I-Mathematics, 325, 653–658, 1997.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I.: Jukebox: A generative model for music, arXiv preprint arXiv:2005.00341, 2020.

Dong, W., Fong, D. Y. T., Yoon, J.-s., Wan, E. Y. F., Bedford, L. E., Tang, E. H. M., and Lam, C. L. K.: Generative adversarial networks for imputing missing data for big data clinical research, BMC medical research methodology, 21, 1–10, 2021.

Dosovitskiy, A. and Brox, T.: Inverting visual representations with convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4829–4837, 2016.

Evensen, G.: The ensemble Kalman filter: Theoretical formulation and practical implementation, Ocean dynamics, 53, 343–367, 2003.

Feng, D., Fang, K., and Shen, C.: Enhancing streamflow forecast and extracting insights using long-short term memory networks with data integration at continental scales, Water Resources Research, 56, e2019WR026 793, 2020.

Fernandez, B. and Salas, J. D.: Periodic gamma autoregressive processes for operational hydrology, Water Resources Research, 22, 1385–1396, 1986.

Frame, J., Nearing, G., Kratzert, F., and Rahman, M.: Post processing the US National Water Model with a Long Short-Term Memory network, 2020.

Frame, J. M., Kratzert, F., Klotz, D., Gauch, M., Shalev, G., Gilon, O., Qualls, L. M., Gupta, H. V., and Nearing, G. S.: Deep learning rainfall-runoff predictions of extreme events, Hydrology and Earth System Sciences Discussions, 2021.

Gauch, M., Kratzert, F., Klotz, D., Nearing, G., Lin, J., and Hochreiter, S.: Rainfall–runoff prediction at multiple timescales with a single Long Short-Term Memory network, Hydrology and Earth System Sciences, 25, 2045–2062, 2021a.

Gauch, M., Mai, J., and Lin, J.: The proper care and feeding of CAMELS: How limited training data affects streamflow prediction, Environmental Modelling & Software, 135, 104 926, 2021b.

Gaume, E. and Gosset, R.: Over-parameterisation, a major obstacle to the use of artificial neural networks in hydrology?, Hydrology and Earth System Sciences, 7, 693–706, https://doi.org/10.5194/hess-7-693-2003, 2003.

Gers, F. A., Schmidhuber, J., and Cummins, F.: Learning to forget: Continual prediction with LSTM, Neural computation, 12, 2451–2471, 2000.

Graves, A.: Generating sequences with recurrent neural networks, arXiv preprint arXiv:1308.0850, 2013.

Gregor, K., Danihelka, I., Graves, A., Rezende, D., and Wierstra, D.: Draw: A recurrent neural network for image generation, in: International Conference on Machine Learning, pp. 1462–1471, PMLR, 2015.

Gupta, H. V., Kling, H., Yilmaz, K. K., and Martinez, G. F.: Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling, Journal of hydrology, 377, 80–91, 2009.

Hsu, K.-l., Gupta, H. V., and Sorooshian, S.: Artificial neural network modeling of the rainfall-runoff process, 31, 2517–2530, 1995.

Kim, J., Tae, D., and Seok, J.: A survey of missing data imputation using generative adversarial networks, in: 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pp. 454–456, IEEE, 2020.

Klotz, D., Kratzert, F., Gauch, M., Keefe Sampson, A., Brandstetter, J., Klambauer, G., Hochreiter, S., and Nearing, G.: Uncertainty Estimation with Deep Learning for Rainfall–Runoff Modelling, Hydrology and Earth System Sciences Discussions, pp. 1–32, 2021.

Kratzert, F., Klotz, D., Brenner, C., Schulz, K., and Herrnegger, M.: Rainfall–runoff modelling using long short-term memory (LSTM) networks, Hydrology and Earth System Sciences, 22, 6005–6022, 2018.

Kratzert, F., Herrnegger, M., Klotz, D., Hochreiter, S., and Klambauer, G.: Neuralhydrology–interpreting lstms in hydrology, in: Explainable ai: Interpreting, explaining and visualizing deep learning, pp. 347–362, Springer, 2019a.

Kratzert, F., Klotz, D., Herrnegger, M., Sampson, A. K., Hochreiter, S., and Nearing, G. S.: Toward Improved Predictions in Ungauged Basins: Exploiting the Power of Machine Learning, Water Resources Research, 55, 11 344–11 354, https://doi.org/https://doi.org/10.1029/2019WR026065, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019WR026065, 2019b.

Kratzert, F., Klotz, D., Shalev, G., Klambauer, G., Hochreiter, S., and Nearing, G.: Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets, Hydrology and Earth System Sciences, 23, 5089–5110, https://doi.org/10.5194/hess-23-5089-2019, https://hess.copernicus.org/articles/23/5089/2019/, 2019c.

Kratzert, F., Klotz, D., Hochreiter, S., and Nearing, G. S.: A note on leveraging synergy in multiple meteorological data sets with deep learning for rainfall–runoff modeling, Hydrology and Earth System Sciences, 25, 2685–2703, https://doi.org/10.5194/hess-25-2685-2021, 2021.

Lamb, A. M., Goyal, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y.: Professor forcing: A new algorithm for training recurrent networks, in: Advances in neural information processing systems, pp. 4601–4609, 2016.

Mahendran, A. and Vedaldi, A.: Understanding deep image representations by inverting them, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5188–5196, 2015.

Mai, J., Shen, H., Tolson, B. A., Gaborit, É., Arsenault, R., Craig, J. R., Fortin, V., Fry, L. M., Gauch, M., Klotz, D., et al.: The Great Lakes Runoff Intercomparison Project Phase 4: The Great Lakes (GRIP-GL), Hydrology and Earth System Sciences Discussions, pp. 1–54, 2022.

Matalas, N. C.: Mathematical assessment of synthetic hydrology, Water Resources Research, 3, 937–945, 1967.

Moshe, Z., Metzger, A., Elidan, G., Kratzert, F., Nevo, S., and El-Yaniv, R.: Hydronets: Leveraging river structure for hydrologic modeling, arXiv preprint arXiv:2007.00595, 2020.

Nash, J. E. and Sutcliffe, J. V.: River flow forecasting through conceptual models part I—A discussion of principles, Journal of hydrology, 10, 282–290, 1970.

Nearing, G., Yatheendradas, S., Crow, W., Zhan, X., Liu, J., and Chen, F.: The efficiency of data assimilation, Water resources research, 54, 6374–6392, 2018.

Nearing, G. S., Gupta, H. V., and Crow, W. T.: Information loss in approximately Bayesian estimation techniques: A comparison of generative and discriminative approaches to estimating agricultural productivity, Journal of hydrology, 507, 163–173, 2013.

Nearing, G. S., Kratzert, F., Sampson, A. K., Pelissier, C. S., Klotz, D., Frame, J. M., Prieto, C., and Gupta, H. V.: What role does hydrological science play in the age of machine learning?, Water Resources Research, p. e2020WR028091, 2020.

Nevo, S., Anisimov, V., Elidan, G., El-Yaniv, R., Giencke, P., Gigi, Y., Hassidim, A., Moshe, Z., Schlesinger, M., Shalev, G., et al.: ML for flood forecasting at scale, arXiv preprint arXiv:1901.09583, 2019.

Nevo, S., Morin, E., Gerzi Rosenthal, A., Metzger, A., Barshai, C., Weitzner, D., Voloshin, D., Kratzert, F., Elidan, G., Dror, G., et al.: Flood forecasting with machine learning models in an operational framework, Hydrology and Earth System Sciences Discussions, pp. 1–31, 2021.

Newman, A., Clark, M., Sampson, K., Wood, A., Hay, L., Bock, A., Viger, R., Blodgett, D., Brekke, L., Arnold, J., et al.: Development of a large-sample watershed-scale hydrometeorological data set for the contiguous USA: data set characteristics and assessment of regional variability in hydrologic model performance, Hydrology and Earth System Sciences, 19, 209, 2015.

Newman, A. J., Mizukami, N., Clark, M. P., Wood, A. W., Nijssen, B., and Nearing, G.: Benchmarking of a physically based hydrologic model, Journal of Hydrometeorology, 18, 2215–2225, 2017.

Olah, C., Mordvintsev, A., and Schubert, L.: Feature Visualization, Distill, https://doi.org/10.23915/distill.00007, https://distill.pub/2017/feature-visualization, 2017.

Rabier, F. and Liu, Z.: Variational data assimilation: theory and overview, in: Proc. ECMWF Seminar on Recent Developments in Data Assimilation for Atmosphere and Ocean, Reading, UK, September 8–12, pp. 29–43, 2003.

Reichle, R. H.: Data assimilation methods in the Earth sciences, Advances in water resources, 31, 1411–1418, 2008.

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T.: DeepAR: Probabilistic forecasting with autoregressive recurrent networks, International Journal of Forecasting, 36, 1181–1191, 2020.

Snyder, C., Bengtsson, T., Bickel, P., and Anderson, J.: Obstacles to high-dimensional particle filtering, Monthly Weather Review, 136, 4629–4640, 2008.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R.: Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199, 2013.

Uria, B., Murray, I., and Larochelle, H.: RNADE: the real-valued neural autoregressive density-estimator, in: Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2, pp. 2175–2183, 2013.

van Leeuwen, P. J.: Nonlinear data assimilation in geosciences: an extremely efficient particle filter, Quarterly Journal of the Royal Meteorological Society, 136, 1991–1999, 2010.

Van Oord, A., Kalchbrenner, N., and Kavukcuoglu, K.: Pixel recurrent neural networks, in: International Conference on Machine Learning, pp. 1747–1756, PMLR, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I.: Attention is all you need, arXiv preprint arXiv:1706.03762, 2017.

Williams, R. J. and Zipser, D.: A learning algorithm for continually running fully recurrent neural networks, Neural computation, 1, 270–280, 1989.

Wunsch, A., Liesch, T., and Broda, S.: Groundwater level forecasting with artificial neural networks: a comparison of long short-term memory (LSTM), convolutional neural networks (CNNs), and non-linear autoregressive networks with exogenous input (NARX), Hydrology and Earth System Sciences, 25, 1671–1687, https://doi.org/10.5194/hess-25-1671-2021, https://hess.copernicus.org/articles/25/1671/2021/, 2021.